On Algorithmic Verification Methods for Probabilistic Systems

Habilitationsschrift
zur Erlangung der venia legendi
der Fakultät für Mathematik und Informatik
der Universität Mannheim

vorgelegt von
Christel Baier
aus Karlsruhe

1998

To my parents, Gerda and Mausi

Acknowledgements

First of all, I would like to thank to Mila Majster-Cederbaum who is by far more than just a supervisor. Many thanks for introducing me in the theory of parallel systems, various fruitful discussions about research topics and countless helpful advices in vocational and private affairs. Without her support and encouragement, I would never have started nor finished my Ph.D.Thesis or this habilitation thesis. In case I will be some time a good researcher or teacher it is due to her. It is impossible to express so many thanks that I owe to her.

When I visited Marta Kwiatkowska in November 1995, she convinced me that research on probabilistic processes is an important and interesting (even so interesting that I wrote a thesis with more than 300 pages about them!) task. Most of the main results in this thesis were developed in collaboration with her. Even our contact is almost exclusive by electronic mail I esteem her as a source of inspiration as well as a friend. Thank you so much.

Many thanks to Holger Hermanns, Joost Katoen and Pedro d'Argenio for being friends and for the various helpful discussions that we had via electronic mail, on the phone, at conferences or in a more relaxed scenario with beer, tequila or champagne.

I thank all my coauthors, discussion partners and all those who have commented on the papers that have served as basis for this work. Especially, I would like to thank Ed Clarke whose support and suggestions helped me so much, not only for this thesis.

Many thanks to Jürgen Jaap and Martin Trampler for their patience and assistence whenever I had a problem with our computer system, UNIX or LATEX, to Rita Sommer without her help I would not have survived in the german bureaucracy jungle and to Alexandra Schubert for all the time she spended for copying papers for me.

Even though research on parallel systems is nice, I am lucky that there is a life outside my office. Special thanks to my best friends Hans Helm, Mark Schad, Petra Gramlich and Petra Bullerkotte for being there whenever I need them, listening to my problems and bearing me even when I am in a bad mood. It might be a strange combination to be a vegetarian and smoker and being addicted in doing sports. However, in the past few years, the "Squash Lagune" turned out to be my second home. Many thanks to all my friends there; especially to Thomas Schmidt who never gave up in trying to teach me playing squash and Heike Stecher and Anne Luck for giving outstanding "power hours".

This thesis is dedicated to my parents who supported me in all areas of life, my sister Gerda and our dog Mausi, the only creature where I am definitely sure that she always looks forward to see me.

Contents

Chapter 1

Introduction

Parallel systems (such as operating systems, telecommunication systems, aircraft controlling systems, banking systems, etc.) arise in many industrial applications. For applications where errors might be expensive and lead to dangerous or even catastrophal situations a precise analysis of the possible system behaviours is an important task. The quality of a parallel system depends on several properties, typically classified into *safety properties* which state that "nothing bad happens" (e.g. mutual exclusion, deadlock freedom or the computation of sufficiently exact numerical values) and *liveness* or *progress properties* which assert that "something good will eventually happen" (e.g. termination, starvation freedom) [OwLa82]. In realistic applications, not only the functionality of a parallel system is important, also *quantitative* aspects (such as time or probabilities) play a crucial role. For instance, in practice, it is useless to establish a property like "each request will eventually be answered" as there is no bound on how much time will pass between a request and the response. Typically, one aims at a property like "each request will eventually be answered within the next 5 seconds". Whether or not a property of this type can be established not only depends on the design of the system but also on the reliability of the interface with the environment or the resources that the system uses. For instance, if the response is transmitted via an uncertain medium that might loose messages a property as above can never hold. However, the cases where a physical error happens might be rare. If the failure rates are known (or can be estimated by experimental results), it makes sense to reason about the frequencies for certain events, i.e. to deal with quantitative properties like "there is a 95% chance that the request will be answered within the next 5 seconds".

Traditionally, the modelling of parallel systems focusses on the *functional* behaviour but abstracts from *quantitative* aspects like time, performance or informations about the frequency of certain system behaviours. In this thesis, we shrink our attention to probabilistic phenomena and consider methods for specifying and validating *probabilistic systems*, i.e. parallel systems where probabilities are used e.g. to model uncertainties or randomized behaviour.

1.1 Verification methods

We first give a brief summary over the general techniques for analyzing parallel systems. These techniques are commonly used for reasoning about the functional behaviour. Suitable adaptions of these methods can be used to treat various types of quantitative behaviour; in particular, they can be applied to analyze probabilistic systems.

A widespread technique for analyzing the properties of a program is *testing* which means to observe the program during the execution with certain well-chosen inputs and to compare the reaction with the desired behaviour. Since testing covers only a (small) subset of the possible instances of system behaviours it can only detect the presence of errors but not the absence of errors. In this thesis, we consider the complementary technique: *verification* which aims at a formal proof for the correctness of a program.[1] This is typically achieved either by *deductive methods* or by *model checking*. The deductive methods are based on a manual composition of a program and a correctness proof using axioms and inference rules for an appropriate specification formalism. In general, these methods require user intervention to a large degree and are very time consuming. Model checking means an algorithmic verification method that takes (an abstract description of) a program and its specification as its input and returns the answer "yes" or "no" depending on whether or not the program meets the specification [ClEm81, QuSi82, CES83]. Thus, the methods based on model checking automate the task of validating programs; and hence, they are user independent to a large extent. While the deductive methods are applicable to systems of arbitrary size (even infinite systems), the model checking approach is (in most cases) limited to finite-state systems.[2] Nevertheless, a large class of parallel processes that appear in realistic applications can be described – with the help of several abstraction techniques – by a system with a finite state space.[3]

Specification formalisms: Any verification method requires a precise description of the desirable system behaviour by a formal *specification*. Two general frameworks to specify the required properties can be identified:

- the specification by a *model* that tells how the system should behave,
- the formalization of the desirable properties by formulas of some *logic*.

The first framework is based on a *homogenous* technique where the program and specification are described in the same formalism and compared via an *implementation relation*, i.e. a binary relation on the objects of that formalism (the "models"). The logical framework focusses on a *heterogenous* technique where different formalisms are used to represent the program and the specification. The program is described by a model (as in the homogenous approach) while the specification is a formula of some program logic.

Branching time versus linear time:[4] Both the homogenous and the heterogenous

[1] While testing is performed by exercising the (real) implementation the verification methods work with an abstraction (a model) of the program. Hence, the verification methods can only assure that the abstract model fulfills the required properties; thus, they can only be as good as the abstraction is.

[2] This observation is clear from the fact that a wide range of verification problems for systems of arbitrary size is undecidable (consider e.g. the halting problem); and hence, cannot be solved automatically.

[3] The benefits of the model checking approach have been documented from the reports on implemented tools, see e.g. [CPS90, McMil92, HoPe94, CCM+95, Camp96, HHW97, LPY97, HarG98].

[4] A detailed discussion about the branching time and linear time view can be found in [Lamp80,

framework reason about the "behaviours" of parallel systems. In the *linear time* view, the behaviour is determined by the possible executions ignoring the possible branches in the intermediate states. The *branching time* view takes the branching structure (the possible steps) of the intermediate states into account and observes a process by means of a "push-button-experiment".[5]

1.1.1 Transition systems

Models describe an abstraction of a system by representing the states and the possible transitions between them. One of the standard models are *transition systems* [Kell76, Plot81] that describe the system behaviour by a directed graph. The nodes represent the states; edges stand for the possible state changes (transitions). The branches (edges) in a state (node) represent the possible steps in that state.[6] The executions (sequences of states) are given by the paths through this graph. In the literature, several types of transition systems are proposed. For instance, the states can be labelled by assertions (e.g. propositional or first order logical formulas that state something about the values of the program and control variables), the transitions can be equipped with action names or boolean guards. In transition systems, asynchronous parallelism is modelled by *interleaving* and *fairness*. The use of interleaving can be motivated by the observation that the effect of the parallel execution $a\|b$ of two "independent" actions a and b (each of them on its own processor) is the same as if a and b are executed in any order on one processor. Hence, from the interleaving point of view, we have the "equality" $a\|b = a;b + b;a$.[7] In

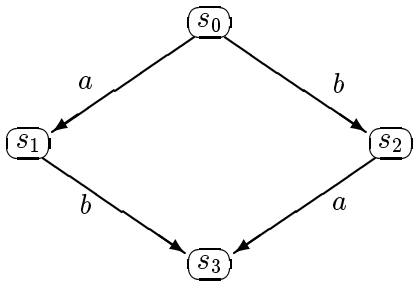


Figure 1.1: $a\|b = a;b + b;a$

other words, interleaving reduces parallelism to the non-deterministic choice that decides which subprocess performs the next step. One might think of this choice to be resolved by the "environment" (e.g. another program that runs in parallel or a user) whose decisions are (in some appropriate sense) *fair* with any subprocess. This kind of fairness is often called *process fairness*. Intuitively, process fairness rules out the pathological possibility that some subprocess is permanently denied to perform the next step. In the literature, various types of fairness are considered.[8] All fairness notions have in common that they

EmHa86, dBdRR88].

[5] For this, the possible steps in a state are viewed as buttons. The observer selects one of these buttons, the system executes the corresponding step and the "push-button-experiment" restarts in the new state.

[6] In the classical approach, the branches stand for non-deterministic alternatives. When dealing with probabilistic phenomena, the branches can also stand for the alternatives of a probabilistic choice.

[7] Here, ; denotes sequential composition and + non-deterministic choice.

[8] For a survey of fairness notions see e.g. [LSP81, QuSi83, Fran88, Kwia89].

make restrictions concerning the non-deterministic choices in the infinite executions (infinite paths in the transition system). They cannot affect the safety properties but might be essential for establishing liveness properties.

1.1.2 Specifying parallel systems with process calculi

Homogenous techniques are mostly used in the context with composition operators provided by some *process calculus* (also often called *process algebra*). Process calculi are specification languages that describe the reactive behaviour of parallel systems. The main ingedrients of such process calculi are operators for modelling parallel composition $\|$, non-deterministic choice $+$, sequential composition ; and recursion.[9] The main distinction mark for the process calculi proposed in the literature is the type of parallelism. Some are *asynchronous* calculi, such as Milner's *CCS* [Miln80], Hoare's *CSP* [Hoar85] or Bergstra & Klop's *ACP* [BeKl84], where the components work time-independently and communicate via certain channels. Others, such as Milner's *SCCS* [Miln83] or *ESTEREL* [BeGon92], are based on *synchronous* parallelism where the steps of the parallel composition are composed by "one-time-steps" of its subprocesses. The "one-time-steps" can either by single atomic steps (as in the case of *SCCS*) or sequences of atomic steps (as in the case of *ESTEREL*).

Implementation relations: Typically, process algebras are supplied with an operational semantics based on transition systems[10] together with an *implementation relation*, a binary relation on transition systems that formalizes what is meant for a program to correctly implement another one. The implementation relation makes it possible to compare a program (the implementation) with its specification. For this, the implementation $\mathcal{P}$ and the specification $\mathcal{Q}$ are described by terms of the process algebra and $\mathcal{P}$ is said to be correct with respect to the specification $\mathcal{Q}$ iff $\mathcal{P} \preceq_{impl} \mathcal{Q}$ for the chosen implementation relation $\preceq_{impl}$.[11] Process algebras equipped with a *congruence* (i.e. an implementation relation $\preceq_{impl}$ that is preserved by the composition operators of the calculus) play a crucial role for the design and analysis of parallel systems. Congruences are useful for the design by *stepwise refinement* since they allow the replacement of the modules $\mathcal{P}_1, \ldots, \mathcal{P}_n$ of a "higher-level" process $\mathcal{P}$ by "lower-level" modules $\mathcal{Q}_1, \ldots, \mathcal{Q}_n$ (provided that $\mathcal{Q}_i \preceq_{impl} \mathcal{P}_i$, $i = 1, \ldots, n$). Moreover, congruences can serve as basis for *modular verification*, i.e. the separate verification of the program modules from which the correctness of the composed process is derived using just the correctness of the modules but not any other knowledge about the modules. Typically, such implementation relations are either equivalences or preorders.[12]

[9] To reason about quantitative properties (e.g. time or probabilities), such calculi can be extended e.g. by operators that specify timeouts or delays or a probabilistic choice operator [GJS90, HaJo90, NRS+90, Hans91, Yi91]. Further references to probabilistic process calculi are given in Section 1.2, page 22.

[10] Often, the terms of a process algebra are identified with the associated transition systems. In particular, the composition operators of the process algebra can also be viewed as operators for composing transition systems.

[11] Thus, verification amounts showing that $\mathcal{P} \preceq_{impl} \mathcal{Q}$.

[12] Recall that a preorder is a reflexive and transitive relation. Both reflexity and transitivity seem to be natural conditions that a relation which formalizes what is meant by "a process implements another one" should have.

- The equivalences can be interpreted in such a way that equivalent programs exhibit the same "behaviour" with respect to an appropriate notion of behaviour.
- In most cases, the use of preorders is motivated by the assumption that the specification just tells which "behaviours" are allowed (but does not prescribe the exact behaviour)[13] but they can also serve as basis to compare the quantitive behaviour of two systems, e.g. if they yield notions of "faster than" or "more reliable".

Among the implementation relation that have proved most useful are *bisimulation* [Miln80, Park81, Miln89] and *simulation* [Miln89, AbLa88, Jons91, LyVa91] relations, *trace equivalence* [Hoar85], *failure equivalence* [BHR84] and *testing preorders* [dNHe83]. Bisimulation and simulation are based on the branching time view and establish a step-by-step correspondence between two systems. As a classical representative for the linear time relations, trace equivalence establishs a correspondence between the executions, but abstracts from the possible branches in the intermediate states. The basic idea behind the testing preorders is to define the process behaviour by means of its ability to pass tests. The tests are special programs (described in terms of the underlying process calculus) that are executed in parallel with the given process. Especially the equivalences that are preserved by the composition operators of the underlying process calculus (i.e. congruences) are of great importance for the analysis since they can be used to reduce the state space by *abstraction*. For this, equivalent states are identified and replaced by a single state. The resulting quotient space might be much smaller[14] and may even be finite for infinite systems.

Strong and weak relations: Weak implementation relations are those that abstract from internal computations while the strong implementation relations do not. Being sensitive with respect to internal steps, in general, strong relations can only be established for systems on the same level of abstraction (e.g. two implementations) while the weak relations are appropriate to compare systems on different levels of abstraction (e.g. an implementation and its specification).

Denotational semantics: Because of its declarative nature, the above mentioned operational semantics (which assigns to each term of the process calculus a transition system) is often the one that a designer has in mind. While the operational semantics focusses on the *stepwise behaviour* the main concepts of denotational semantics are *compositionality* and the use of *fixed point equations* for modelling recursion.[15] In many cases, the use of fixed point theory requires methods of several mathematical disciplines (e.g. topology, domain theory, category theory) and lead to a semantics that is hard to understand for a non-mathematician. However, the denotational approach provides a much more elegant technique to define the meanings of recursive (or repetitive) programs. Often, denotational semantics are used to obtain a characterization of the implementation relation associated with the operational semantics by means of a *full abstraction* result. Full abstraction means that the denotational semantics of a program contains exactly the information that is relevant for the chosen implementation relation (but abstracts from all other details about the program). Full abstraction results can serve as basis for ver-

[13] In this case, the use of preorders can be viewed as a proof technique for establishing safety properties.

[14] See e.g. [CGT+96] for an expressive example.

[15] The fixed point equations reflect our intuition that a recursive procedure and their body have the same behaviour.

ification methods[16] or just help for a better understanding of the operational semantics and the implementation relation. Moreover, being compositional, denotational semantics allow for proofs by structural induction which is often a useful concept to establish a link between several specification formalisms (e.g. some kind of logic or operational models).[17]

1.1.3 The temporal logical approach

In the logical framework where the specification is a formula (or the conjunction of formulas) and the system is described by a model (e.g. a transition system) verification amouts showing that the formula ϕ evaluates to true when interpreted over the system. In the literature, several logics are proposed to reason about parallel systems, such as dynamic [Prat76], temporal [Pnue77] or modal [HeMi85, Koze85] logic. In this thesis, we concentrate on the use of *propositional temporal logic* with future temporal modalities like "eventually" $\Diamond$ or "always" $\Box$. We briefly sketch the basic ideas where we mainly concentrate on those aspects that are relevant for the results of this thesis. Further details can be found e.g. in [Emer90, MaPn92, CGL93, Lamp94, MaPn95].

Linear time logic *LTL*: In the linear time approach, formulas describe properties of executions. Linear time formulas are built from atomic propositions (that make assertions about the states, e.g. about the current values of the program or control variables), the usual boolean combinators $\vee$, $\wedge$, $\neg$ and temporal operators. For instance, if $crit_1$, $crit_2$ are atomic propositions stating that certain subprocesse $\mathcal{P}_1$ and $\mathcal{P}_2$ are in their critical section then $\Box(\neg crit_1 \wedge \neg crit_2)$ stands for the safety property stating mutual exclusion. To reason about quantitative aspects, e.g. time, special modalities like "sometimes within the next k steps" $\Diamond^{\leq k}$ can be used, see e.g. [HaJo89, ACD90].[18] For example, the formula $\Box(request \rightarrow \Diamond^{\leq 5} response)$ might be interpreted as the liveness property stating that any request will answered within the next 5 time units. Linear time formulas are interpreted over the executions (i.e. the paths in a transition system). For a process, a linear time formula is viewed to be fulfilled if it holds even in a worst case (but realistic) scenario, i.e. if it holds for all "possible" executions. In general, not all executions are viewed to be possible, but only those that obey certain fairness conditions.

Branching time logic *CTL*: Branching time logics allow quantification over the possible futures which leads to formulas stating e.g. the existence or non-existence of an execution with a certain property.[19] Computation tree logic *CTL* introduced by Clarke & Emerson [ClEm81] is the classical representative for branching time logics. *CTL* distinguishes between state and path formulas. The state formulas subsume the propositional connectives

[16] It should be pointed out that the denotational approach can also be of importance for other practical applications. Especially in the field of sequential programs (but also for other types of programs), the procedural nature of denotational semantics can serve as a basis for a compiler. See e.g. [BCH98].

[17] For example, in this thesis, we apply the denotational framework for showing that two implementation relations coincide for a certain kind of processes.

[18] The interpretation of a "step" depends on the underlying system. A step might be one unit of time or – if the system under consideration arises from the asynchronous parallel composition of several subsystems – one can think of a step as the time taken by the slowest component to perform an atomic action.

[19] Here, we assume the traditional (non-probabilistic) approach. In a probabilistic scenario, branching time formulas might also reason about the probability of certain events. See Section 1.2.3.

and basic temporal operators of the form "a path quantifier followed by a single temporal modality" where the path quantifiers are $\forall$ or $\exists$ that range over all paths (executions) and the temporal modalities are as in linear time logic.[20] The logic *CTL** [EmHa86] extends *CTL* by allowing arbitrary linear time formulas to serve as path formulas; thus, it subsumes *LTL* and *CTL*.

Model checking: For finite systems, model checking algorithms are developed for both linear and branching time logic. While *CTL* model checking can be done in polynomial time (even in time linear in the size of the system and in the length of the formula [CES83]) model checking for *LTL* and *CTL** is *PSPACE*-complete [SiCl86] and can be done in time exponential in the length of the formula and linear in the size of the system [LiPn85].[21] On the basis of decision procedures for the satisfiability problem of (linear or branching time) temporal logic, the task for synthezising parallel systems from a given temporal logical specification can be automated, see e.g. [EmCl82, MaWo84, AtEm89, PnRo89].

1.1.4 State explosion problem

The size (number of states in the transition system) of a parallel system $\mathcal{P} = \mathcal{P}_1 \| \ldots \| \mathcal{P}_n$ grows exponentially in the number n of subprocesses. This explains why any algorithmic verification method that works with an explicit representation of transition systems (e.g. by adjacency lists) fails for systems with very much components. In the last decade, several methods have been developed to attack the "state explosion problem". Some are based on a *symbolic* representation of the system using *binary decison diagrams* [BCM+90, McMil92], others are based on the concept of *partial order reduction* [Pele93, Valm94, Gode94]. The basis idea behind the BDD-based approach goes back to Ken McMillan who proposed to handle very large systems by representing their transition relation implicitly by an ordered BDD [Brya86]. The BDD-approach has proved to be very successful for various types of verification problems for parallel systems, including the verification against branching and linear time temporal logical specifications and establishing a branching time relation between two systems [BCM+90, McMil92, EFT93, CGL93, CGH94]. Partial order reduction is based on the observation that the interleaved execution of independent actions allows one to investigate only a representative fragment of the state space. It is applicable for proving linear time temporal properties and for the process algebraic approach.[22] Both techniques have been implemented in tools and successfully applied to very large systems, see e.g. [McMil92, HoPe94, Camp96, GPS96].

1.2 Probabilistic systems

In the literature, a variety of extensions of the above mentioned verification methods are proposed that are appropriate to reason about quantitative aspects, e.g. for verifying real-

[20] To handle fairness the semantics of *CTL* has to be modified by taking $\forall$ and $\exists$ as quantifiers that range over the fair executions [EmLei85].

[21] See [VaWo86, CGH94, GPV+95] for other *LTL* model checking algorithms.

[22] More details about the partial order approach can be found in several papers in the proceedings [PPH96].

time conditions, for performance analysis or for computing the probabilities for certain system behaviours. In this thesis, we concentrate on probabilistic phenomena and consider parallel systems with probabilities for the state transitions (in the sequel called *probabilistic systems* or *probabilistic processes*).[23] There are several situations where probabilistic aspects have to be taken into account. The ones that we have in mind when speaking about a probabilistic system are the following two:

- To get a realistic model of a parallel system that reacts on the stimuli of the environment, one has to take into consideration the interfaces with the environment. These are often based on physical processes that are *probabilistic in nature*.
- The system (or one or more of its subsystems) might be based on a randomized algorithm, i.e. uses the concept of *randomization* ("tossing a fair coin").

In the former case, probabilities are used to model uncertainties (e.g. the failure rate of an unreliable medium that transmits messages). In the second case, the probabilities are determined by the frequencies of the possible outcomes of a probabilistic choice. The benefits of randomization are clear from the literature.[24] Randomization has been shown to be a elegant technique that might lead to simpler and more efficient algorithms than their non-randomized counterparts. Moreover, as observed by Lehmann & Rabin [LeRa81], in the field of parallel algorithms, the use of randomization makes it possible to solve problems that are not solvable with deterministic algorithms.

Probabilistic choice: The characteristic feature of probabilistic systems is that they work with the concept of *probabilistic choice*. This refers to any activity that chooses between several alternative behaviours where the frequencies of the possible outcomes of that choice are given by probabilities (i.e. values in the unit interval $[0,1]$ that sum up to 1). The interpretation of this probabilistic choice depends on the concrete process. As mentioned above, the probabilities might be obtained from failure rates of certain unreliable resources or might stem from a "truly randomized" action like "tossing a fair coin". In any case, probabilistic choice can be specified by a term of the form

$$random(p_1 : \mathcal{P}_1, \ldots, p_l : \mathcal{P}_l) \quad \text{often written as} \quad [p_1]\mathcal{P}_1 \oplus \ldots \oplus [p_l]\mathcal{P}_l$$

that we interpret as the process that chooses randomly to behave as one of the processes $\mathcal{P}_i$. Here, $p_1, \ldots, p_l \in [0,1]$ such that $p_1 + \ldots + p_l = 1$. Assuming *internal* probabilistic choice (which is resolved independent on the environment), the value p_i denotes the probability that the process $\mathcal{P}_i$ is selected. This stands in contrast to *external* probabilistic choice which assumes that the environment determines which of the processes $\mathcal{P}_1, \ldots, \mathcal{P}_l$ are enabled. For this, let us assume that $\mathcal{P}_1, \ldots, \mathcal{P}_k$ are available while $\mathcal{P}_{k+1}, \ldots, \mathcal{P}_l$ are not. Then, the external probabilistic choice selects one of the processes $\mathcal{P}_1, \ldots, \mathcal{P}_k$ according to the *conditional probabilities* $\frac{p_i}{p_1+\ldots+p_k}$, $i = 1, \ldots, k$.

1.2.1 Modelling probabilistic behaviour

Most of the models that are used for the representation of probabilistic systems are extensions of transition systems but there are also other models such as "true concurrency"

[23] In this chapter, we use the notions "system" and "process" as synonyms.

[24] See e.g. the papers by Rabin [Rabi76a, Rabi76b, Rabi80], the survey papers [Karp91, GSB94] or the books [MoRa95, Lync95].

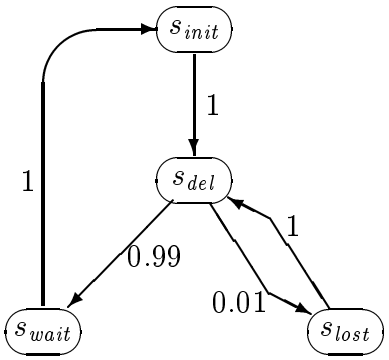
models (e.g. event structures with probabilities [KLL94, Kato96]). In this thesis, we concentrate on the use of *probabilistic transition systems*. To reason about probabilities, several extensions of transition systems have been proposed.[25] They all have in common, that they endow the transitions with probabilities in an appropriate way. The resulting models can be classified with respect to their treatment of non-determinism.

Fully probabilistic models: Several authors consider models based on *Markov chains* (MCs) where the concept of non-determinsm is replaced by probabilistic choice, e.g. "generative transition systems" [vGSST90], "sequential Markov chains" [LeSh82, HaSh84, Vard85, CoYa88, CoYa95] or "fully probabilistic automata" [SeLy94, Sega95a]. In these models, each state s is associated with a probabilistic choice; that is, the transitions are labelled by probabilities (values in the unit interval), such that, for each state s, the probabilities for the outgoing transitions sum up to 1.[26]

Example 1.2.1 [Simple communication protocol: the sender] We consider a simple communication protocol similar to that in [HaJo94]. The system consists of two enitities: a sender that works with an unreliable medium which might loose messages and a receiver. The sender, having produced a message, transmits the message to the medium, which in turn tries to deliver the message to the receiver. With probability 1/100, the messages gets lost and the medium retries to deliver the message. With probability 99/100, the message is delivered correctly, in which case the sender waits for the acknowledgement by the receiver and then returns to the initial state. For simplicity, we assume that the acknowledgement cannot be corrupted or lost. We describe the behaviour of the sender by the following Markov chain.

We use the following four states:

- s_{init}: the state in which the sender produces a message and passes the message to the medium
- s_{del}: the state in which the medium tries to deliver the message
- s_{lost}: the state reached when the message is lost
- s_{wait}: the state reached when the message is delivered correctly and in which the system waits for the acknowledgement by the receiver.



For instance, the transition $s_{wait} \to s_{init}$ stands for the case where the sender gets the acknowledgement of the receipt of the message; $s_{del} \to s_{lost}$ for the case where the medium looses the messages. ■

Probabilistic models with non-determinism: On the other hand, there is a variety of models based on *Markov decision processes* (MDPs) which allow for both probabilistic and non-deterministic branching. For the MDP-based models, there are different ways of associating probabilities to the transitions. One possibility is to dis-

[25] The "probabilistic automaton" à la Rabin [Rabi63] (that were introduced as language acceptors) can be viewed as a precursor of this approach.

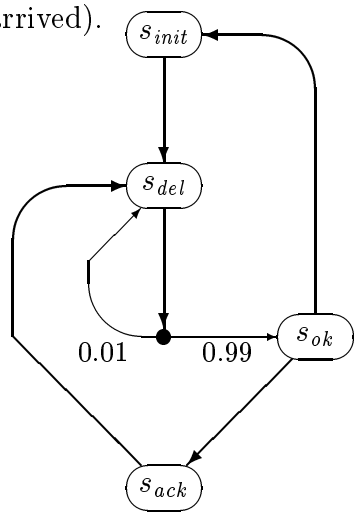
[26] Of course, there might also be terminal states without any outgoing transitions. Moreover, many authors allow for "substochastic states" where the probabilities of the outgoing transitions sum up to a value $p \in]0,1[$. In this case, the remaining value $1-p$ can be interpreted as the probability for deadlock.

tinguish between probabilistic and non-probabilistic states.[27] Representatives of such models are "concurrent Markov chains" [Vard85, CoYa88, CoYa95] and "alternating systems" [HaJo90, Hans91].[28] Another possibility is to allow each state to behave non-deterministically where each of the non-deterministic alternatives is associated with a probabilistic choice. Examples for such systems are the models (just called "probabilistic programs") considered in [HSP83, Pnue83, PnZu86a, PnZu86b, PnZu93], the "probabilistic automaton" of [SeLy94, Sega95a], "probabilistic non-deterministic systems" of [BidAl95, dAlf97a, dAlf97b] and "real-time probabilistic programs" of [ACD91a].

Example 1.2.2 [Simple communication protocol: Sender ‖ Receiver] We consider a variant of the simple communication protocol of Example 1.2.1 (page 19) where we specify the behaviour of the parallel composition of the sender and the receiver by a probabilistic system with non-determinism.[29] For simplicity, we assume that both the sender and the receiver work with mailing boxes that cannot hold more than one message at any time. Thus, if the sender has produced a message m then the next message cannot be produced before m is delivered correctly; similarly, the medium cannot be activiated as long as there is an unread message in the mailing box of the receiver (i.e. as long as the acknowledgement for the last message is not yet arrived).

We use the following four states:

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



The state s_{ack} is reached in the case where the sender has already produced the next message while there is still an unread message in the mailing box of the receiver. Thus, the only possible step in s_{ack} is the one where the receiver "consumes" the message and acknowledges the receipt. In state s_{ok}, the sender and the receiver can work *in parallel* (simultaneously): the sender may produce the next message while the receiver may consume the last message. The parallelism in state s_{ok} is described by *interleaving*, i.e. the non-deterministic choice that decides which process performs the next step: either the sender produces the next message or the receiver consumes the last message. The interleaving

[27] Probabilistic states are those where a probabilistic choice is resolved while non-probabilistic states behave purely non-probabilistic, possibly non-deterministic.

[28] The idea of separating the probabilistic branches from non-probabilistic activities is also realized in the "stratified transition systems" of [vGSST90]. These are introduced as operational model for a language with probabilistic choice but lacks for non-deterministic choice. Thus, in the stratified systems of [vGSST90], non-determinism is not present. However, non-determinism could be easily added to the language and the model.

[29] We use the model where any state is associated with a set of non-deterministic alternatives and where each of these alternatives is represented by a probabilistic choice.

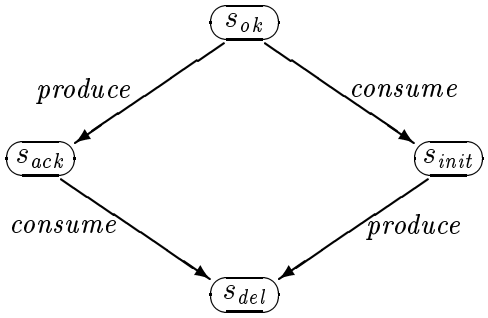


Figure 1.2: The "diamond" obtained by interleaving

of the actions *produce* and *consume* in state s_{ok} leads to the classical "diamond" shown in Figure 1.2 stating that the effect of the parallel execution of *produce* and *consume* is the same as if *produce* and *consume* are executed in any order: in either case, we reach the state s_{del}. ■

Of course, the classification MC-based versus MDP-based models is too coarse to capture all models proposed in the literature. Several authors introduced models that can be classified between MCs and MDPs such as "reactive systems" [LaSk89, vGSST90] or "probabilistic I/O automaton" [WSS94].

Internal vs external probabilistic choice: The formal definition of these models does not depend on whether internal or external probabilistic choice is assumed. The difference between internal and external probabilistic choice becomes visible in the context of composition operators of a process calculus. Especially the *restriction operator* (that specifies the processes or actions that are enabled in a certain state) is affected from the chosen type of probabilistic choice.

Specifying probabilistic systems: Which of these models should be used depends on the concrete application. Roughly speaking, the models based on MCs are suitable to formalize the behaviour of sequential randomized algorithms or processes of probabilistic calculi with synchronous parallel composition or probabilistic shuffle operators while the models based on MDPs can be used to describe the behaviour of distributed randomized algorithms or processes of an asynchronuous probabilistic calculus.

The need of non-determinism: When modelling distributed randomized algorithms or asynchronous probabilistic systems by MDP-based models, non-determinism is used to model interleaving (cf. Example 1.2.2, page 20). As observed by several other authors, e.g. [JHY94, JoYi95, Sega95a], there are also other situations where the concept of non-determinism might be helpful. The non-determinism might be useful to represent *underspecification* which can be (totally or partly) resolved in further refinement steps (cf. [JoYi95]). This situation is well-known in the design of (sequential or distributed) algorithms. For example, in a high-level design one might use a statement like

"choose some index $i \in \{1, \ldots, n\}$ and put $x := a[i]$"

(e.g. in a high-level description of Quicksort the Pivot element might be chosen by a statement like that) while in the implementation one works e.g. with the assignment $x := a[1]$ (or a randomized assignment $x := \mathit{random}(a[1], \ldots, a[n])$). Another example is that

"non-determinism can be used to specify the allowed probabilities of failure of a medium where a refinement step is used to decrease the set of allowed failure rates [JoLa91]" (where we quote from [JoYi95]). Second, also observed in [JoYi95], non-determinism can be used to represent *incomplete information* on the parameters of system behaviour such as Milner's weather conditions [Miln89].

Example 1.2.3 [Roulette player] Figure 1.3 (page 22) shows the "one-day-behaviour" of an addicted roulette player. For simplicity, we assume that he is arbitrary rich and always chooses the simple risk "red" or "black" and that there is no limit on the allowed stake.[30] When entering the casino, the roulette player starts playing with the stake 1\$. Whenever he looses the last game, he doubles the stake for the next game. On the other hand, if he has won the last game, he decides *non-deterministically* to continue playing (in which case he restarts with the stake 1\$) or to leave the casino with one last game where he risks all his money. Here, the non-deterministic choice is used to describe the *incomplete information* about the "environment". The choice in state s_{won} between staying in or leaving the casino might be dependent on the well-being of the roulette player or on the mood of his wife or on other unknown factors. ■

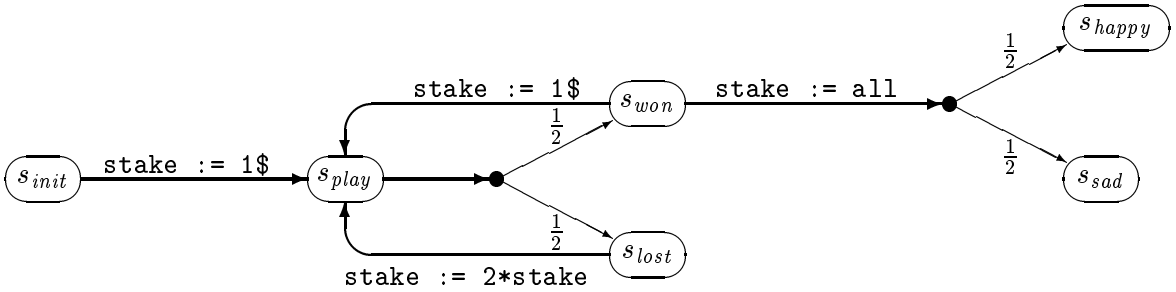


Figure 1.3: The "one-day-behaviour" of the roulette player

1.2.2 The process calculus approach for probabilistic systems

In the literature, a variety of *probabilistic process calculi* are proposed. They either replace the non-deterministic choice operator by a probabilistic choice operator or allow for both non-deterministic and probabilistic choice. See e.g. [GJS90, JoSm90, vGSST90, Toft90, LaSk92, Toft94] for synchronous and [HaJo90, Hans91, YiLa92, Yi94, Lowe93b, Seid95, BaKw97, Norm97] for asynchronous process calculi and [BBS92, SCV92, NúdF95, GLN+97, dAHK98] for calculi with probabilistic shuffle operators.[31] Some of these calculi can be used to reason about *priorities* [SmSt90, Toft94, Lowe95]. Typically, such process calculi are supplied with an operational semantics based on (some kind of) probabilistic transition systems. In absence of non-determinism, the calculi with synchronous parallelism or a probabilistic shuffle operator can be described by a fully probabilistic (MC-based) system [GJS90, vGSST90, BBS92, LaSk92]; but also other operational semantics (e.g. based on the reactive or stratified view) are possible [vGSST90, Toft94].

[30] Moreover, we neglect the possible outcome "Zero" (where the bank gets all stakes) and suppose that the probability for winning a game is 1/2.

[31] The probabilistic shuffle operators describe the interleaved execution of two processes with respect to a fixed scheduler that decides randomly which process performs the next step where the underlying random choice depends on the local states of the processes.

The operational semantics of probabilistic calculi that allow for non-deterministic choice and/or deal with asynchronous parallelism can be defined by means of MDP-based models (probabilistic transition systems with non-determinism), see e.g. [HaJo90, Hans91, YiLa92, Yi94, BaKw97].

Implementation relations for probabilistic processes: Several implementation relations for probabilistic processes are proposed, such as trace, failure and ready equivalence [JoSm90], bisimulation [LaSk89, HaJo90, Hans91, SeLy94, BaHe97][32], simulation-like preorders [JoLa91, Yi94, SeLy94, Sega95a] and various types of testing preorders [Chri90a, Chri90b, CSZ92, YiLa92, Chri93, YCDS94, JHY94, JoYi95, NúdF95, Sega96, Norm97, KwNo98a, KwNo98b].

Verification methods: Even though many implementation relations for probabilistic systems have been introduced, corresponding verification methods (i.e. methods for showing that one process implements another one with respect to an appropriate implementation relation) are relatively rare. For fully probabilistic systems (the MC-based models), both axiomatic [GJS90, JoSm90, LaSk92, BBS92] and algorithmic [Chri90a, ChCh91, HuTi92, Chri93, BaHe97] methods have been developed. All the above mentioned algorithmic methods run in *polynomial time*. Especially in the case of trace and failure equivalence [HuTi92], this fact is of interest since decidability of the corresponding relations for non-probabilistic systems is *PSPACE*-complete [KaSm83]. In the case of (strong or weak) bisimulation or simulation, the time complexities are polynomial in the non-probabilistic [KaSm83, PaTa87, BoSm87, GroVa90, HHK95] as well as the probabilistic [HuTi92, BaHe97] case. For the models with non-determinism (the MDP-based models), verification methods for the branching time relations (bisimulation and simulation) are proposed so far (see [HaJo90, Hans91, Yi94, Toft94] for axiomatizations and [Bai96, PSS98] for algorithmic verification methods) while – as far as the author knows – the literature lacks for methods for other implementation relations (such as testing equivalence à la [JoYi95] or any weak linear time relation).

Denotational semantics: The work by Kozen [Koze79] on denotational semantics for sequential programs with random assignment and while-loops can be seen as a precursor of the denotational approach. Jones & Plotkin [JoPl89, Jone90] introduce the probabilistic powerdomain of *evaluations* to provide a denotational semantics for a programming language with while-loops and a probabilistic concurrency operator. Roughly speaking, for semantical purposes, evaluations are used to decorate *sets* of behaviours with probabilities rather than single behaviours. The concept of evaluations is often used in denotational semantics for randomized programs; e.g. for probabilistic predicate transformers [Jone90, MMS96, HMS97] but also in the field of probabilistic process algebras. Evaluations are used in [MMS+94] to give a failure/divergence semantics for *CSP* with probabilistic choice and in [BaKw97] to obtain denotational semantics for a probabilistic extension of *CCS* that are shown to be fully abstract with respect to bisimulation and simulation. Other denotational characterizations for probabilistic variants of *CSP* (that do not use evaluations) are proposed by Lowe [Lowe93a, Lowe93b, Lowe95] and Seidel [Seid95]. Denotational models and related full abstraction results for certain types of testing preorders are presented by Christoff [Chri90a, Chri90b], Jonsson & Yi [JoYi95]

[32] See also [dViRu97, BDE+97, DEP98] where bisimulation equivalence for "continuous" probabilistic systems are introduced.

and Kwiatkoswka & Norman [KwNo96, Norm97, KwNo98a, KwNo98b]. [Hart98] presents several denotational semantics for a *CCS*-like language with probabilistic choice and discusses the use of internal or external probabilistic and non-deterministic choice. The above mentioned semantics for the asynchronous calculi are all based on the interleaving view. A denotational "true concurrency" semantics for a variant of *LOTOS* with time and probabilities by means of event structures is given by Katoen [Kato96].

1.2.3 Probabilistic temporal logic

Several authors proposed extensions of program logics to reason about qualitative or quantitative temporal properties of probabilistic systems. In this introduction, we only explain the main ideas behind the temporal logical framework.[33] *Qualitative* properties assert that a certain event φ holds with probability 0 or 1 while *quantitive* properties guarantee that the probability for a certain event φ meets given lower or upper bounds.[34] In most applications, the quantitive properties deal with an upper bound ϵ for some small ϵ and assert that the probability for a "bad event" is sufficiently small (i.e. $< \epsilon$ or $\leq \epsilon$) or use a lower bound $1 - \epsilon$ and state that a certain safety or liveness condition is satisfied with some sufficiently large probability (i.e. with a probability in the interval $]1 - \epsilon, 1]$ or $[1 - \epsilon, 1]$).[35] In the temporal logical framework, the event φ describes a property for the executions and is specified by a *path formula* built from standard temporal operators like $\Diamond$, $\Box$ and $\Diamond^{\leq k}$ (see Section 1.1.3, page 16).

Linear time logics: The linear time framework (see e.g. [Vard85, VaWo86, PnZu86a, PnZu86b, CoYa88, PnZu93, CoYa95]), uses classical "non-probabilistic" linear time temporal logics (where formulas are path formulas) with an interpretation over the states of a probabilistic systems. The truth value of a formula φ in a state s is a value $p_s(\varphi)$ in the interval [0,1] which can be viewed as the probability that φ holds for an execution starting in s. Satisfaction of a quantitative linear time specification (consisting of a formula φ and a lower or upper bound for the "acceptable" probabilities) means that the truth value $p_s(\varphi)$ meets the given bound.[36]

Branching time logics: In contrast to these Fuzzy logic like interpretations of linear time formulas (with truth values in the unit interval), the branching time framework deals with *state formulas* that might hold for a state or not (i.e. that are equipped with an interpretation over the states of a probabilistic system by the usual truth values 0 or 1). [LeSh82, HaSh84, ACD91b] propose branching time logics for specifying qualitative temporal properties by using state formula that assert that a certain event holds with probability 1. Branching time logics that allow to express quantitative properties

[33] For other program logics for specifying probabilistic systems see e.g. [Feld83, FeHa84, Koze85] for dynamic and [LaSk89, LaSk92, ChCh92, Chri93, HuKw97, MoMcI97, McIv98, HuKw98] for modal logics.

[34] In fully probabilistic systems, there is a natural probability measure on the executions. This is different in probabilistic transition systems with non-determinism where it makes no sense to speak about probabilities unless the non-determinism is resolved. However, the "probability" for an event can be defined as the minimal or maximal probability measures for this event ranging over the possible resolutions of the non-deterministic choices.

[35] Clearly, the qualitative properties can be viewed as special instances of the quantitive properties; we just have to deal with $\epsilon = 0$.

[36] In particular, satisfaction of a qualitative linear time specifications means that $p_s(\varphi)$ is 0 or 1.

(see e.g. [HaJo89, Hans91, HaJo94, SeLy94, ASB+95, BidAl95, dAlf97a]) integrate the lower/upper bounds for the acceptable probabilities into the syntax and use formulas e.g. of the form $\mathsf{Prob}_{\geq p}(\varphi)$ that state that the probability for the event φ is at least p.

Verification methods: Proving the correctness of a probabilistic process against qualitative properties expressed in the temporal logical framework amounts showing that the given event φ holds with probability 0 or 1. For finite systems, it has been realized that this is completely independent on the precise transition probabilities and just depends on the "topology" of the underlying directed graph. This observation was first made by Hart, Sharir & Pnueli [HSP83] for proving termination with probability 1 and later used in several verification methods for establishing qualitative temporal properties; see e.g. [LeSh82, Pnue83, HaSh84, PnZu86a] for deductive methods and [Vard85, VaWo86, PnZu86b, CoYa88, ACD91a, ACD91b, PnZu93, CoYa95] for algorithmic methods. Establishing quantitative temporal properties requires the computation of the exact probabilities for the given event φ; see e.g. [LSS94, PoSe95, Sega95a] for proof rules, [CoYa88, HaJo94, ASB+95, CoYa95, IyNa96] for algorithmic methods for fully probabilistic systems and [CoYa90, Hans91, BidAl95, dAlf97a, dAlf97b] for algorithmic methods for probabilistic systems with non-determinism. The main concepts for the handling of formulas involving the "eventually operator" $\Diamond$ is the use of *linear equation systems* in the case of fully probabilistic systems [CoYa88, HaJo94] and *linear optimization problems* in the case of probabilistic systems with non-determinism [CoYa90, BidAl95]. The time complexities of the model checking algorithms for branching time logics are polynomial [HaJo94, BidAl95]. For linear time logic, model checking is *PSPACE*-complete in the case of fully probabilistic systems and complete for double exponential time in the case of probabilistic systems with non-determinism [Vard85, CoYa95].

Fairness: For non-probabilistic parallel systems, it is well-known that fairness assumptions about the resolutions of the non-deterministic choices might be essential for proving certain liveness properties. Clearly, this observation carries over to probabilistic systems with non-determinism and concerns qualitative as well as quantitative properties. As an example, consider the *randomized dining philosophers* [LeRa81]: when two philosophers are simultaneously ready to flip a fair coin in order to decide which fork to pick up, one can think of this as two probability distributions, each respectively with probability $\frac{1}{2}$ of obtaining heads or tails, enabled in the same state. If the scheduler never selects a given philosopher for execution even though he is ready to proceed (e.g. to flip the coin) the run thus produced would be unfair, and as a result one could not guarantee the qualitative property that asserts lack of starvation. As an example for a situation where fairness assumptions are essential for establishing quantitative properties, consider a communication protocol which attempts to deliver a message to the recipient if one is received on the input channel from the environment, and loops back to the initial state otherwise. In a realistic scenario, the outcome of the delivery is probabilistic, and will result in a message being delivered correctly with some suitably high probability, say 0.999, or an error state being reached if a fault has occurred in the transmitting medium. Then, the property "the message is eventually delivered with probability 0.9" can only be established on condition that the protocol does not loop back to the initial state forever. Hence, also in the probabilistic case, it is desirable to have methods for proving (quantitative or qualitative) temporal properties under fairness constraints. Establishing temporal properties under fairness constraints (for a probabilistic system with non-determinism) amounts showing

that an event φ holds with some sufficiently small or large probability (or with probability 0 or 1 in the case of a qualitative property), provided that the non-deterministic choices are resolved in a fair manner.

Even though the verification of qualitative properties under fairness assumptions is well-understood (see e.g. [HSP83, Vard85, PnZu86b, PnZu93] for algorithmic methods) only a few research has been done so far in the field of verification methods for establishing quantitative properties under fairness constraints. [LSS94, PoSe95, Sega95a] present proof rules for establishing quantitative (timed) progress properties for randomized distributed systems which can be combined with several notions of fairness. As far as the author knows, [BaKw98, dAlf97a] are the first attempts to formulate algorithmic methods for verifying quantitative properties of probabilistic systems with non-determinism which take fairness into account.

1.3 The topics of this thesis

This thesis investigates several aspects of formal reasoning about probabilistic systems.[37]

(I) **The process algebra approach:** We consider asynchronous and synchronous probabilistic process calculi, operational and denotational semantics for them and homogenous algorithmic verification methods. The main contributions are:

- denotational characterizations of bisimulation and simulation (Chapter 5),
- algorithms for establishing a branching time relation (bisimulation or simulation) between probabilistic systems with non-determinism (Chapter 6),
- the definition of weak bisimulation for fully probabilistic systems together with a corresponding verification algorithm (Chapter 7) and the definition of a lazy synchronous parallel composition operator that preserves weak bisimulation equivalence (Section 4.3).

(II) **The temporal logic approach:** We consider the linear and branching time framework for establishing qualitative and quantitative temporal properties. The main contributions are:

- a technique for proving qualitative linear time properties with well-known non-probabilistic methods (Chapter 8),
- algorithms for establishing quantitive temporal properties of a probabilistic system with non-determinism and fairness by means of a model checking algorithm for a probabilistic temporal logic $PCTL^*$ with a satisfaction relation that involves fairness of non-deterministic choice (Chapter 9).

(III) **Symbolic verification:** Chapter 10 presents verification algorithms for probabilistic systems that use multi-terminal BDDs (MTBDDs) as data structure. The main idea is the development of a "language" for manipulating MTBDDs in which several verification problems for probabilistic systems can be embedded. This yields symbolic model checking algorithms for $PCTL$ (interpreted over fully probabilistic

[37] Most results are published with coauthors. The corresponding reference can be found in the introduction of each chapter.

systems or probabilistic systems with non-determinism and fairness) and MTBDD-based methods for checking strong and weak bisimulation equivalence for fully probabilistic systems.

1.3.1 Related work

In the literature, a lot of work has been done in the field of formal methods for probabilistic systems; see the references mentioned before.[38] In the authors opinion, it would make little sense to list all related work here and to explain in which way this thesis is related. This is done in the respective chapter. At this place, the author just wants to refer to the *thesis*' [Chri90a, Jone90, Seid92, Hans91, Chri93, Lowe93a, Sega95a, dAlf97a, Norm97, HarG98] that are all about specification formalisms and/or verification methods of probabilistic systems and hence related to this thesis at a large degree.[39] Especially the excellent work by Hans Hansson [Hans91], Roberto Segala [Sega95a] and Luca deAlfaro [dAlf97a] (and several papers that they wrote with coauthors) had great influence on the development of this thesis.[40]

- The model for concurrent probabilistic systems that we use here essentially agrees with the one of [Sega95a, dAlf97a] and is a variant of the one of [Hans91].
- The process calculus *PCCS* of Chapter 4 is a variant of the process calculus (also called *PCCS*) introduced by Hansson & Jonsson [HaJo90].
- The bisimulation equivalence and the simulation preorder that we consider in Chapters 5 and 6 were introduced by Segala & Lynch [SeLy94].
- The main concepts of the logic $PCTL^*$ that we consider in Chapter 9 are taken from papers by each of the three, namely [HaJo89, Hans91, HaJo94, SeLy94, BidAl95, dAlf97a, dAlf97b]. Moreover, the idea of using ω-automaton for our $PCTL^*$ model checking algorithm was suggested by Luca deAlfaro.
- The symbolic *PCTL* model checking algorithm of Chapter 10 take the methods of Hansson & Jonsson [HaJo94] and Bianco & deAlfaro [BidAl95] as basis.

1.3.2 How to read this thesis

Chapter 2 collects our notations concerning sets, relations, functions and distributions. The reader is not supposed to read this chapter but he/she should keep in mind that he/she has a fair chance to find the explanations for our notations in Chapter 2. Chapter 3 serves as basis for all other chapters because it introduces (and tries to motivate) the models and explains the notations that are used in almost all parts of this thesis. A reader not familiar with probabilistic systems should read this chapter first while a reader

[38] Clearly, also any work on formal methods to reason about other quantitative aspects is related to the topic of that thesis. In particular, the field of performance analysis, see e.g. [Herz90, GHR93, GiHi94, Hill94, Pria96, dAKB98, BeGor98, Herm98, HHM98], (where continuous time Markov chains and stochastic Petri nets [Moll82, MBC84] belong to the standard models) is close to the approach here.

[39] This list of thesis' might be far from being complete. It contains only those thesis' that treat probabilistic systems as their main topic and that had influences to this thesis.

[40] It should be pointed out that each of the three thesis' also considers real-time aspects while this thesis does not.

who is familiar with probabilistic processes might skip this chapter keeping in mind that the notations specific to this thesis can be found there. To support a reader who is only interested in certain parts of this thesis the author tried to make the remaining chapters as independent as possible. In those cases where a result of one chapter is used in another chapter the reader will find a (page) reference. The appendix (Chapter 12) recalls some definitions/concepts presented somewhere in the literature; the notations introduced there are always used in connection with a reference to the relevant part of Chapter 12.

Proofs: For the sake of readability, in most chapters the main results are presented without proofs (but with a page reference to the place where the proof can be found). The proofs are given in the last section of the respective chapter.[41] A reader not interested in the theoretical development of the results might skip the appended "proof-sections".

Examples: The main concepts are illustrated by simple toy examples. These are either abstract examples (without any concrete meaning) or extremely simplified examples with a realistic background. Examples of the former type should just demonstrate a certain technique. Although unrealistic, examples of the latter type should give some insights how to apply the proposed framework in realistic situations.

The symbols ■ and ⌋: We use the symbol ■ to denote the end of a proof, remark or example. Some proofs are devided into subclaims. The symbol ⌋ denotes the end of the proof of such a subclaim.

Background: Even not necessary, some familiarity with the basic concepts of formal methods for parallel systems might be helpful. Elementary notions of several mathematical disciplines (such as numerical analysis, linear algebra, probability and measure theory, topology and graph theory) are used without any explanation. However, a reader not familiar with them (but interested in the topics of this thesis) should not immediately give up to read this thesis; an intuitive understanding of e.g. the notion "probability" or the knowledge what a linear equation system or optimization problem is should be sufficient to understand the main ideas. We do not recall the basic notions of the above mentioned mathematical disciplines here and refer to any standard book about the respective discipline.[42]

[41] In a few cases, the proof of a certain theorem is given in the "proof-section" of another chapter. This is only done in those cases where a simple proof can be derived from the results of a further chapter.

[42] For instance, see [Halm50, Rudi66, Fell68, GrWe86] for measure and probability theory, [Dugu66, Suth77, Enge89] for topology and [Even79, Goul88] for graph theory. Basic knowledge about the theory of Markov (decision) processes, see e.g. [Derm70, Ross83, Pute94], might be helpful but is not necessary.

Chapter 2

Preliminaries

In this chapter, we briefly explain some notations that are used throughout the thesis. For a first reading, the reader might skip this chapter, but should keep in mind that our notations concerning sets, relations, partitions, functions and distributions are explained here.

2.1 Sets, relations, partitions and functions

Sets: For X to be a set, 2^X is the powerset of X. id_X denotes the *identity* on X, i.e. the function $id_X : X \to X$, $id_X(x) = x$ for all $x \in X$. The *characteristic function* of a subset X' of X is the boolean-valued function $X \to \{0,1\}$, $x \mapsto 1$ iff $x \in X'$. If X is finite then $|X|$ denotes the number of elements of X. If X is infinite then we put $|X| = \infty$. $\uplus$ denotes disjoint union.

Relations: Let R, R_1, R_2 be binary relations on X. We also write $x_1 R x_2$ to denote that $(x_1, x_2) \in R$. We define $R^{-1} = \{(x_2, x_1) \in X \times X : (x_1, x_2) \in R\}$ and $R_1 \circ R_2 = \{(x_1, x_2) \in X \times X : \exists\, x \in X\ ((x_1, x) \in R_1 \ \wedge\ (x, x_2) \in R_2)\}$. We often write $R_1 R_2$ rather than $R_1 \circ R_2$. R^* denotes the transitive, reflexive closure of R.

Equivalences and partitions: If R is an equivalence relation on a set X then X/R denotes the *quotient space* (i.e. the set of equivalence classes) and, for $x \in X$, $[x]_R$ the equivalence class of x with respect to R. A *partition* of X is a set $\mathcal{X}$ consisting of pairwise disjoint nonempty subsets of X such that $\bigcup_{B \in \mathcal{X}} B = X$. We often refer to the elements of a partition as *blocks*. Clearly, for each equivalence relation R on X, the quotient space X/R is a partition of X. Vice versa, each partition of X induces an equivalence relation on X: For $\mathcal{X}$ to be a partition of X, $R_{\mathcal{X}}$ denotes the induced equivalence relation, i.e. $R_{\mathcal{X}}$ consists of all pairs $(x_1, x_2) \in X \times X$ where $x_1,\ x_2 \in B$ for some $B \in \mathcal{X}$. We often write $[x]_{\mathcal{X}}$ (instead of $[x]_{R_{\mathcal{X}}}$) to denote the unique block $B \in \mathcal{X}$ that contains x. A partition $\mathcal{X}$ is called *finer* than a partition $\mathcal{X}'$ (and $\mathcal{X}'$ is called *coarser* than $\mathcal{X}$) iff the induced equivalence relation $R_{\mathcal{X}}$ is finer than $R_{\mathcal{X}'}$ (i.e. iff each $B \in \mathcal{X}$ is contained in some $B' \in \mathcal{X}'$). We say $\mathcal{X}$ is *strictly finer* than $\mathcal{X}'$ (or $\mathcal{X}'$ *strictly coarser* than $\mathcal{X}$) iff $\mathcal{X}$ is finer than $\mathcal{X}'$ and $\mathcal{X} \neq \mathcal{X}'$.

Functions: For X and Y to be sets, $X \to Y$ denotes the function space of all functions from X to Y. If $f : X \to Y$ is a function and $X' \subseteq X$ then $f|_{X'}$ denotes the restriction of f

on X', i.e. $f|_{X'}$ denotes the function $f|_{X'} : X' \to Y$ which is given by $f|_{X'}(x) = f(x)$. For $Y' \subseteq Y$, $f^{-1}(Y')$ denotes the inverse image of Y' under f, i.e. $f^{-1}(Y') = \{x \in X : f(x) \in Y'\}$. For $y \in Y$, we put $f^{-1}(y) = f^{-1}(\{y\})$. Similarly, for $X' \subseteq X$, $f(X')$ denotes the image of X' under f, i.e. $f(X') = \{f(x) : x \in X'\}$. We define $\mathit{Range}(f) = f(X)$ to denote the range (image) of f. $g \circ f$ denotes the usual function composition, i.e. if $g : Y \to Z$ and $f : X \to Y$ are functions then $g \circ f : X \to Z$ is given by $(g \circ f)(x) = g(f(x))$. If $f : X \to X$ is a function then $f^0 = id_X$ and, for $n = 0, 1, 2, \ldots$, $f^{n+1} = f \circ f^n$.

2.2 Distributions

Distributions: Let X be a set. A *distribution* on X is a function $\mu : X \to [0, 1]$ such that $\{x \in X : \mu(s) > 0\}$ is countable and $\sum_{x \in X} \mu(x) = 1$. If $x \in X$ then μ^1_x denotes the unique distribution on X with $\mu^1_x(x) = 1$. $\mathit{Supp}(\mu)$ denotes the *support* of μ, i.e. the set of elements $x \in X$ with $\mu(x) > 0$. For $\emptyset \neq A \subseteq S$, we write $\mu[A]$ to denote $\sum_{x \in A} \mu(x)$. In particular, $\mu[\emptyset] = 0$. $\mathit{Distr}(X)$ denotes the collection of all distributions on X.

The composition $\mu * \nu$: Let μ, ν be distributions on X and Y respectively. The *composition* $\mu * \nu$ is the distribution on $X \times Y$ which is given by $(\mu * \nu)(x, y) = \mu(x) \cdot \nu(y)$.

Weight functions (cf. [SeLy94, Sega95a]): Let μ, ν distributions be on X and Y respectively and $R \subseteq X \times Y$. A *weight function* for (μ, ν) with respect to R is a function $\mathit{weight} : X \times Y \to [0, 1]$ which satisfies:

1. $\mathit{weight}(x, y) \neq 0$ for at most countably many $(x, y) \in X \times Y$.
2. For all $x \in X$, $y \in Y$:

$$\sum_{y \in Y} \mathit{weight}(x, y) \;=\; \mu(x), \qquad \sum_{x \in X} \mathit{weight}(x, y) \;=\; \nu(y)$$

3. If $\mathit{weight}(x, y) > 0$ then $(x, y) \in R$.

We write $\mu \preceq_R \nu$ if there exists a weight function for (μ, ν) with respect to R.[1] Clearly, if $R_1 \subseteq R_2$ then each weight function with respect to R_1 is also a weight function with respect to R_2. Hence, $\mu \preceq_{R_1} \nu$ implies $\mu \preceq_{R_2} \nu$.

Remark 2.2.1 Let μ_X, μ_Y and μ_Z be distributions on X, Y, Z respectively, and let $R_{X,Y} \subseteq X \times Y$, $R_{Y,Z} \subseteq Y \times Z$ and $\mathit{weight}_{X,Y} : X \times Y \to [0, 1]$ a weight function for (μ_X, μ_Y) with respect to $R_{X,Y}$, $\mathit{weight}_{Y,Z} : Y \times Z \to [0, 1]$ a weight function for (μ_Y, μ_Z) with respect to $R_{Y,Z}$. Then, $\mathit{weight}_{X,Z} : X \times Z \to [0, 1]$,

$$\mathit{weight}_{X,Z}(x, z) \;=\; \sum_{y \in \mathit{Supp}(\mu_Y)} \frac{\mathit{weight}_{X,Y}(x, y) \cdot \mathit{weight}_{Y,Z}(y, z)}{\mu_Y(y)},$$

is a weight function for (μ_X, μ_Z) with respect to $R_{X,Y} \circ R_{Y,Z}$. Thus, if $\mu_X \preceq_{R_{X,Y}} \mu_Y$ and $\mu_Y \preceq_{R_{Y,Z}} \mu_Z$ then $\mu_X \preceq_R \mu_Z$ where $R = R_{X,Y} \circ R_{Y,Z}$. In particular, if X is a set and

[1] Intuitively, the weight function *weight* shows how to split the probabilities $\mu(x)$ and $\nu(y)$ such that the relation R is preserved: if $\mu(x)$, $\nu(y) > 0$ then we "combine" the $\mathit{weight}(x, y)/\mu(x)$-part of x with the $\mathit{weight}(x, y)/\nu(y)$-part of y. Then, the whole of each $x \in \mathit{Supp}(\mu)$ is combined with certain parts of elements $y \in \mathit{Supp}(\nu)$ where $(x, y) \in R$.

$R \subseteq X \times X$ is a transitive relation then $\preceq_R$ is a transitive relation on $Distr(X)$. From this, if R is a preorder on X then $\preceq_R$ is a preorder on $Distr(X)$. ■

Remark 2.2.2 Let $R \subseteq X \times Y$ and $\mu \in Distr(X)$, $\mu' \in Distr(Y)$ such that $\mu \preceq_R \mu'$. Then, $\mu' \preceq_{R^{-1}} \mu$. ■

The function $Distr(f)$**:** For $f : X \to Y$ to be a function, the function $Distr(f) : Distr(X) \to Distr(Y)$ is given by $Distr(f)(\mu)(y) = \mu[\ f^{-1}(y)\ ]$.

Remark 2.2.3 Let $f : X \to Y$ be a function. Then,

$$weight : X \times Y \to [0,1], \quad weight(x,y) \ = \ \begin{cases} \mu(x) & : \text{ if } f(x) = y \\ 0 & : \text{ otherwise} \end{cases}$$

is a weight function for $(\mu, Distr(f)(\mu))$ with respect to $R = \{(x, f(x)) : x \in X\}$. Thus, $\mu \preceq_R Distr(f)(\mu)$. ■

Chapter 3

Modelling probabilistic behaviour

In this chapter we introduce the models for probabilistic systems (together with some related notations) that are used throughout the thesis. We shrink our attention to those models that are extensions of (non-probabilistic) *transition systems* which have been established as one of the standard models for non-probabilistic systems.

The main distinctive mark for the probabilistic models proposed in the literature is the treatment of non-determinism. On the one hand, there are various extensions of Markov chains (MCs) that allow for probabilistic (but not for non-deterministic) choice; these can be used to analyze the behaviour of sequential randomized algorithms or processes of a calculus with synchronous parallel composition. On the other hand, there are several extensions of Markov decision processes (MDPs) that are suitable to specify both probabilistic and non-deterministic behaviour; these are suitable to describe the behaviour of distributed randomized algorithms or processes of a calculus with asynchronous parallel composition.

The formal definition of a model for probabilistic systems does not depend on whether *internal* or *external probabilistic choice* is assumed.[1] Internal probabilistic choice is resolved independent on the environment while the resolution of an external probabilistic choice depends on the processes/actions that are enabled in a certain state.[2] In this thesis, we assume internal probabilistic choice. This will only be important in Chapter 4 where the process algebra approach is considered.

Organization of that chapter: In the first three sections we give the formal definitions of the models that we use in that thesis. We start with the basic models without any labellings where Section 3.1 deals with the MC-based models (called *fully probabilistic systems*) and Section 3.2 with the MDP-based models (called *concurrent probabilistic systems*). These "stripped" models are extended in Section 3.3 by action labels for the transitions and by proposition labels for the states. In Section 3.4, we recall the definition of probabilistic bisimulation à la Larsen & Skou [LaSk89] and probabilistic simulation as introduced by Segala & Lynch [SeLy94]. In the remainder of the thesis, we distinguish between "systems" and "processes". The notion "system" is used to describe a structure

[1] See page 18 for the motivation behind internal and external probabilistic choice.

[2] The underlying type of probabilistic choice influences the semantics of the composition operators of a process calculus; in particular, the restriction operator. See Remark 4.2.4 (page 81).

consisting of a state space and a transition relation (possibly extended by certain labels) while a "process" denotes a "pointed system" (i.e. a system with an initial state). This will be explained in Section 3.5. In Section 3.6, we sketch how our models fit into the hierarchy of models studied in the literature.

3.1 Fully probabilistic systems

This section introduces the basic concepts for the MC-based models. We follow the notations of Segala & Lynch [SeLy94, Sega95a] and use the adjective "fully probabilistic".

Definition 3.1.1 [Fully probabilitistic systems] *A* fully probabilistic system *is a tuple* $\mathcal{S} = (S, \mathbf{P})$ *where* S *is a set of* states *and* $\mathbf{P} : S \times S \to [0,1]$ *is a function (called the* transition probability function*) such that, for all* $s \in S$, $\mathbf{P}(s,t) > 0$ *for at most countably many* $t \in S$ *and* $\sum_{t \in S} \mathbf{P}(s,t) \in \{0,1\}$.

Example 3.1.2 [Simple communication protocol: the sender] The behaviour of the sender in the simple communication protocol of Example 1.2.1 (page 19) can be formalized by the fully probabilistic system $(S, \mathbf{P})$ where $S = \{s_{init}, s_{del}, s_{lost}, s_{wait}\}$ and $\mathbf{P}(s_{init}, s_{del}) = \mathbf{P}(s_{lost}, s_{del}) = \mathbf{P}(s_{wait}, s_{init}) = 1$, $\mathbf{P}(s_{del}, s_{init}) = 0.01$, $\mathbf{P}(s_{del}, s_{wait}) = 0.99$ and $\mathbf{P}(\cdot) = 0$ in all other cases. ■

Let $\mathcal{S} = (S, \mathbf{P})$ be a fully probabilistic system. $\mathcal{S}$ is said to be *finite* iff S is finite. For finite systems, we also refer to $\mathbf{P}$ as the *transition probability matrix*. If $C \subseteq S$ and $s \in S$ then we put $\mathbf{P}(s, C) = \sum_{t \in C} \mathbf{P}(s,t)$. A state s of $\mathcal{S}$ is called *terminal* iff $\mathbf{P}(s, S) = 0$.

An *execution fragment* or *finite path* in $\mathcal{S}$ is a nonempty finite "sequence" $\sigma = s_0 \to s_1 \to \ldots \to s_k$ where $k \geq 0$, $s_0, s_1, \ldots, s_k \in S$ and $\mathbf{P}(s_i, s_{i+1}) > 0$, $i = 0, 1, \ldots, k-1$.

- $|\sigma|$ denotes the *length* of σ, i.e. we put $|\sigma| = k$.
- $first(\sigma)$ denotes the first state of σ, i.e. $first(\sigma) = s_0$.
- $last(\sigma)$ denotes the last state of σ, i.e. $last(\sigma) = s_k$.
- $\sigma(i)$ denotes the $(i+1)$-st state of σ, i.e. $\sigma(i) = s_i$, $i = 0, 1, \ldots, k$,
- $\sigma^{(i)}$ denotes the i-th prefix of σ, i.e. $\sigma^{(i)} = s_0 \to s_1 \to \ldots \to s_i$, $i = 0, 1, \ldots, k$. If $i > k = |\sigma|$ then we put $\sigma^{(i)} = \sigma$.
- If $k = |\sigma| = 0$ then we put $\mathbf{P}(\sigma) = 1$. For $k \geq 1$, we define

$$\mathbf{P}(\sigma) = \mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdot \ldots \cdot \mathbf{P}(s_{k-1}, s_k).$$

- σ is called *maximal* iff $last(\sigma)$ is terminal.

A state t is called *reachable* from s if there exists a finite path σ with $first(\sigma) = s$ and $last(\sigma) = t$.

Example 3.1.3 The system in Example 1.2.1 (page 19) has no finite maximal execution fragment as there are no terminal states. $\sigma = s_{init} \to s_{del} \to s_{lost} \to s_{del} \to s_{wait}$ is an execution fragment (finite path) with $|\sigma| = 4$, $first(\sigma) = s_{init}$, $last(\sigma) = s_{wait}$, $\sigma(2) = s_{lost}$, $\sigma^{(2)} = s_{init} \to s_{del} \to s_{lost}$ and $\mathbf{P}(\sigma) = 1 \cdot 0.01 \cdot 1 \cdot 0.99 = 0.0099$. ■

An *execution* or *fulpath* is either a maximal execution fragment or an infinite "sequence" $\pi = s_0 \to s_1 \to s_2 \to \ldots$ where $s_0, s_1, \ldots \in S$ and $\mathbf{P}(s_{i-1}, s_i) > 0$, $i = 1, 2, \ldots$. For π to be an infinite execution, $\pi(i)$, $\pi^{(i)}$ and $first(\pi)$ are defined as for execution fragments. For infinite executions we put $|\pi| = \infty$ and define

$$inf(\pi) = \{s \in S : \pi(i) = s \text{ for infinitely many indices } i \geq 0\}.$$

If σ is a finite path then we put $inf(\sigma) = \emptyset$. A *path* denotes a finite path or a fulpath.

- $Path^{\mathcal{S}}_{ful}$ denotes the set of fulpaths in $\mathcal{S}$,
- $Path^{\mathcal{S}}_{ful}(s)$ the set of fulpaths starting in s,
- $Path^{\mathcal{S}}_{fin}$ the set of finite paths in $\mathcal{S}$,
- $Path^{\mathcal{S}}_{fin}(s)$ the set of finite paths starting in s,
- $Reach^{\mathcal{S}}(s)$ denotes the set of states which are reachable from s.

If the underlying fully probabilistic system $\mathcal{S}$ is clear from the context we abbreviate $Path^{\mathcal{S}}_{ful}$ to $Path_{ful}$, $Path^{\mathcal{S}}_{ful}(s)$ to $Path_{ful}(s)$, $Path^{\mathcal{S}}_{fin}$ to $Path_{fin}$, $Path^{\mathcal{S}}_{fin}(s)$ to $Path_{fin}(s)$ and $Reach^{\mathcal{S}}(s)$ to $Reach(s)$.

- If Π is a set of fulpaths in $\mathcal{S}$ and $s \in S$ then $\Pi(s) = \Pi \cap Path_{ful}(s)$.
- If Σ is a set of finite paths then $\Sigma(s) = \Sigma \cap Path_{fin}(s)$.

$\leq_{prefix}$ denotes the *prefix relation* on paths, i.e. if γ_1, γ_2 are (finite or infinite) paths then $\gamma_1 \leq_{prefix} \gamma_2$ iff γ_1 is a prefix of γ_2 (iff $\gamma_1 = \gamma_2$ or $\gamma_1 = \gamma_2^{(k)}$ for some k). $<_{prefix}$ denotes the *proper prefix relation* on paths, i.e. $\sigma <_{prefix} \gamma$ iff $\sigma = \gamma^{(i)}$ for some $i < |\gamma|$.

Let $\sigma = s_0 \to \ldots \to s_k$ be a finite path and $\gamma = t_0 \to t_1 \to \ldots$ a (finite or infinite) path. If $last(\sigma) = first(\gamma)$ (i.e. $s_k = t_0$) then we define $\sigma \circ \gamma$ to be the path $s_0 \to \ldots \to s_k \to t_1 \to t_2 \to \ldots$ that arises by appending γ at the end of σ where the last state of σ and the first state of γ are identified. We define:

- $\sigma\uparrow$ denotes the *basic cylinder* induced by σ, i.e. $\sigma\uparrow = \{\pi \in Path_{ful}(s) : \sigma \leq_{prefix} \pi\}$.
- $\sigma\uparrow_{fin} = \{\sigma' \in Path_{fin}(s) : \sigma \leq_{prefix} \sigma'\}$.
- $\sigma\downarrow = \{\sigma' \in Path_{fin}(s) : \sigma' \leq_{prefix} \sigma\}$.
- For Σ to be a set of finite paths,

$$\Sigma\uparrow = \bigcup_{\sigma\in\Sigma} \sigma\uparrow, \quad \Sigma\downarrow = \bigcup_{\sigma\in\Sigma} \sigma\downarrow, \quad \Sigma\uparrow_{fin} = \bigcup_{\sigma\in\Sigma} \sigma\uparrow_{fin}.$$

For each state s, $\mathbf{P}$ induces a probability space on $Path_{ful}(s)$ as follows. We define $SigmaField^{\mathcal{S}}(s)$ to be the smallest sigma-field on $Path_{ful}(s)$ which contains all basic cylinders $\sigma\uparrow$ where σ ranges over all finite paths starting in s (i.e. $\sigma \in Path_{fin}(s)$). The probability measure $Prob$ on $SigmaField^{\mathcal{S}}(s)$ is the unique measure with $Prob(\sigma\uparrow) = \mathbf{P}(\sigma)$.

Lemma 3.1.4 *Let $(S, \mathbf{P})$ be a fully probabilistic system and $\Sigma \subseteq Path_{fin}$ such that σ, $\sigma' \in \Sigma$, $\sigma \neq \sigma'$ implies $\sigma \not\leq_{prefix} \sigma'$. Then, for all $s \in S$,*

$$Prob(\Sigma(s)\uparrow) = \sum_{\sigma\in\Sigma(s)} \mathbf{P}(\sigma).$$

Proof: easy verification. Uses the fact that $\sigma\uparrow$, $\sigma \in \Sigma$, are pairwise disjoint. ■

We now turn our attention how to compute the probabilities to reach a state of a certain set S_2 via a path leading through states of a certain set S_1 only. More precisely, we consider a fully probabilistic system $(S, \mathbf{P})$ and two subsets S_1, S_2 of S. Let $\Sigma \subseteq Path_{fin}$ be the set of all finite paths σ such that

- $\sigma(i) \in S_1 \setminus S_2$, $i = 0, 1, \ldots, |\sigma| - 1$,
- $last(\sigma) \in S_2$.

and let $\Pi = \Sigma\uparrow$. Then, our aim is to compute the probabilities $Prob(\Pi(s))$.

Lemma 3.1.5 *Let $(S, \mathbf{P})$ be a fully probabilistic system and let S_1, S_2, Σ and Π as above. For all $s \in S$,*

$$\sum_{\sigma \in \Sigma(s)} \mathbf{P}(\sigma) \;=\; Prob(\Pi(s)).$$

Proof: follows immediately from Lemma 3.1.4 (page 35). ■

The following result characterizes the probabilities $Prob(\Pi(s))$ as the least fixed point of a certain monotonic operator on the function space $S \mapsto [0,1]$.

Theorem 3.1.6 *Let $(S, \mathbf{P})$ be a fully probabilistic system and let S_1, S_2, Σ and Π as above. Then,*

$$p : S \to [0,1],\ p(s) = Prob\,(\Pi(s)),$$

is the least fixed point *of the operator $F : (S \to [0,1]) \to (S \to [0,1])$ which is given by $F(f)(s) = 1$ if $s \in S_2$, $F(f)(s) = 0$ if $s \in S \setminus (S_1 \cup S_2)$ and, if $s \in S_1 \setminus S_2$,*

$$F(f)(s) \;=\; \sum_{t \in S} \mathbf{P}(s,t) \cdot f(t).$$

Proof: see Section 3.7, Corollary 3.7.2 (page 65). ■

Proposition 3.1.7 (cf. [CoYa88, HaJo94, CoYa95]) *Let $(S, \mathbf{P})$ be a finite fully probabilistic system and let S_1, S_2, Σ, Π and p be as in Theorem 3.1.6. Moreover, let $S^{NO} = \{s \in S : \Sigma(s) = \emptyset\}$, S^{YES} a subset of S with $S_2 \subseteq S^{YES} \subseteq \{s \in S : \Sigma(s)\uparrow = Path_{ful}(s)\}$ and $S^? = S \setminus (S^{NO} \cup S^{YES})$. Then, p is the* unique fixed point *of the operator $F' : (S \to [0,1]) \to (S \to [0,1])$ which is given by*

$$F'(f)(s) \;=\; \begin{cases} F(f)(s) & : \text{ if } s \in S^? \\ 1 & : \text{ if } s \in S^{YES} \\ 0 & : \text{ if } s \in S^{NO}. \end{cases}$$

Here, F is as in Theorem 3.1.6 (page 36).

Proof: The claim is a slight generalization of the results established in [CoYa88, HaJo94, CoYa95] and can be shown in a similar way. ■

Remark 3.1.8 [Computing the probabilities $p(s)$] If S is finite then Theorem 3.1.6 (page 36) yields that the probabilities $p(s) = Prob(\Sigma(s)\uparrow)$ can be obtained by *iteration*:

We take $p_n(s) = 1$ if $s \in S_2$ and $p_n(s) = 0$ if $s \in S \setminus (S_1 \cup S_2)$, $n = 0, 1, 2, \ldots$. For $s \in S_1 \setminus S_2$, we define $p_0(s) = 0$ and, for $n = 0, 1, 2, \ldots$,

$$p_{n+1}(s) \;=\; \sum_{t \in S} \mathbf{P}(s,t) \cdot p_n(t).$$

Then, $\lim p_n(s) = p(s)$ for all $s \in S$. This iterative method can be reformulated as follows. Let $\mathbf{p}_n$ be the vector $(p_n(s))_{s \in S}$ and let $\mathbf{Q}$ be the matrix $(q_{s,t})_{s,t \in S}$ where

$$q_{s,t} \;=\; \begin{cases} \mathbf{P}(s,t) & : \text{ if } s \in S_1 \setminus S_2 \\ 1 & : \text{ if } s = t \in S_2 \\ 0 & : \text{ otherwise.} \end{cases}$$

Then, $\mathbf{p}_n = \mathbf{Q} \cdot \mathbf{p}_{n-1} = \mathbf{Q}^2 \cdot \mathbf{p}_{n-2} = \ldots = \mathbf{Q}^n \cdot \mathbf{p}_0$. In particular, the vector $\mathbf{p} = (p(s))_{s \in S}$ is given by

$$\mathbf{p} \;=\; \lim_{i \to \infty} \mathbf{Q}^{2^i} \mathbf{p}_0$$

where the matrices $\mathbf{Q}^{2^i}$ are obtained by *iterative squaring* (i.e. by successively computing $\mathbf{Q}^{2^{i+1}} = \mathbf{Q}^{2^i} \cdot \mathbf{Q}^{2^i}$, $i = 0, 1, 2, \ldots$). Another possibility (used in [CoYa88, HaJo94, CoYa95]) for computing the function $p(\cdot)$ is based on Proposition 3.1.7 (page 36). First, one computes the sets S^{NO} and S^{YES} by a graph analysis. Second, one solves the regular *linear equation system* $\mathbf{x} = \mathbf{A} \cdot \mathbf{x} + \mathbf{b}$ (or equivalently, $(\mathbf{I} - \mathbf{A}) \cdot \mathbf{x} = \mathbf{b}$) where $\mathbf{b} = (b_s)_{s \in S^?}$ with $b_s = \mathbf{P}(s, S^{YES})$, $\mathbf{x} = (x_s)_{s \in S^?}$, $\mathbf{A} = (\mathbf{P}(s,t))_{s,t \in S^?}$ and $\mathbf{I}$ the $|S^?| \times |S^?|$-identity matrix. ■

Example 3.1.9 For the simple communication protocol of Example 1.2.1 (page 19) we compute the probabilities that the message is eventually delivered correctly by taking $S_1 = S$, $S_2 = \{s_{wait}\} = S^{YES}$, $S^{NO} = \emptyset$ and solving the linear equation system

$$x_{init} = 1 \cdot x_{del}, \quad x_{lost} = 1 \cdot x_{del}, \quad x_{del} = \tfrac{1}{100} \cdot x_{lost} + \tfrac{99}{100}$$

which yields $x_{init} = x_{lost} = x_{del} = 1$. ■

In the following lemma, we give a graph-theoretical criteria for $Prob(\Pi(s)) = 1$ where we assume that $S_1 = S$ and $S_2 = U$. In that case, $Prob(\Pi(s))$ is the probability for the "progress property" stating that, from state s, the system will eventually reach a U-state.

Lemma 3.1.10 *Let $(S, \mathbf{P})$ be a finite fully probabilistic system and $U \subseteq S$. Let Σ and Π be as in Theorem 3.1.6 (page 36) where $S_1 = S$ and $S_2 = U$. Let $s \in S$ and $T = \{last(\sigma) : \sigma \in Path_{fin}(s), \sigma \notin \Sigma \uparrow_{fin}\}$. Then, we have:*

$$\Sigma(t) \neq \emptyset \text{ for all states } t \in T \;\; \textit{iff} \;\; Prob(\Pi(s)) = 1.$$

Proof: see Section 9.5, Corollary 9.5.5 (page 242). ■

Lemma 3.1.10 yields that whether or not a qualitative progress property of the type "with probability 1, the system will eventually reach a U-state" holds does not depend on the exact probabilities but only the topology of the underlying directed graph.[3]

Example 3.1.11 In Example 3.1.9 (page 37), we computed the probabilities $x_* = 1$ for the states of the communication protocol of Example 1.2.1 (page 19) to reach the state s_{wait} by solving a linear equation system. Alternatively, we could apply Lemma 3.1.10. ■

[3] This result is not surprising since a similar result for concurrent systems was established by Hart, Sharir & Pnueli [HSP83] (see Chapter 9, page 227).

Definition 3.1.12 [Boundedness, cf. [LeSh82, HaSh84, LaSk89]] *A fully probabilistic system $(S, \mathbf{P})$ is called* bounded *iff there exists a real number c with $0 < c < 1$ such that, for all s, $t \in S$, if $\mathbf{P}(s,t) > 0$ then $\mathbf{P}(s,t) \geq c$.*

Clearly, each finite fully probabilistic system is bounded. Moreover, whenever $(S, \mathbf{P})$ is bounded and s a state in S then there are only finitely many states t where $\mathbf{P}(s,t) > 0$. I.e. in bounded systems, the set of (immediate) successors of a state is always finite.

3.2 Concurrent probabilistic systems

The concept of non-determinism is necessary to describe the interleaving behaviour of parallel (non-probabilistic or probabilistic) systems whose components work asynchronously. In this section we present the basic concepts of the MDP-based models that allow for both probabilistic and non-deterministic branching. Thus, the MDP-based models are suitable to specify the interleaving behaviour of randomized distributed systems (cf. Example 1.2.2, page 20); but they are also appropriate for all other situations where the concept of non-determinism might be useful, e.g. to represent underspecification or incomplete information about the environment [JHY94, JoYi95, Sega95a] (cf. Example 1.2.3, page 22).

Like the models considered e.g. in [HSP83, PnZu93, SeLy94, Sega95a, BidAl95, dAlf97a], our basic model – called "concurrent probabilistic systems" – assigns to each state a set of non-deterministic alternatives where each of them stands for a randomized step of the system. On the other hand, there are models, e.g. those considered in [Vard85, Hans91, CoYa95], that capture the branching structure of the purely probabilistic choices and distinguish between probabilistic and non-probabilistic states. These two kinds of MDP-based models have equivalent "power". We define the models of the latter type as special instances of our basic model. Vice versa, we briefly explain how the behaviour of a system described by our basic model can be specified by a model that distinguishes between probabilistic and non-probabilistic states.

Definition 3.2.1 [Concurrent probabilistic system] *A* concurrent probabilistic system *is a pair $\mathcal{S} = (S, \mathit{Steps})$ where S is a set of states and Steps a function which assigns to each state $s \in S$ a set $\mathit{Steps}(s)$ of distributions on S.*[4]

Let $\mathcal{S} = (S, \mathit{Steps})$ be a concurrent probabilistic system. A state s is called *terminal* iff $\mathit{Steps}(s) = \emptyset$. $\mathcal{S}$ is called *finite* iff S and $\bigcup_{s \in S} \mathit{Steps}(s)$ are finite. We write $s \longrightarrow \mu$ iff $s \in S$ and $\mu \in \mathit{Steps}(s)$ and refer to $s \longrightarrow \mu$ as a *transition* or a *step* of s. If μ is of the form μ_t^1 then we also write $s \to t$ rather than $s \to \mu_t^1$. Intuitively, *Steps* represents the non-deterministic alternatives in each state: given a state $s \in S$, a scheduler chooses some transition $s \longrightarrow \mu$ which represents a randomized step of the system, i.e. if $s \longrightarrow \mu$ is the chosen step then, with probability $\mu(t)$, the system reachs the state t afterwards.

We depict concurrent probabilistic systems as follows. We use circles for the states. Thick lines stand for the outgoing transitions from a state. The thick line corresponding to a transition $s \longrightarrow \mu$ is directed and ends in a small circle that represents the probabilistic

[4] For the definition of a "distribution" and related notations, see Section 2.2, page 30.

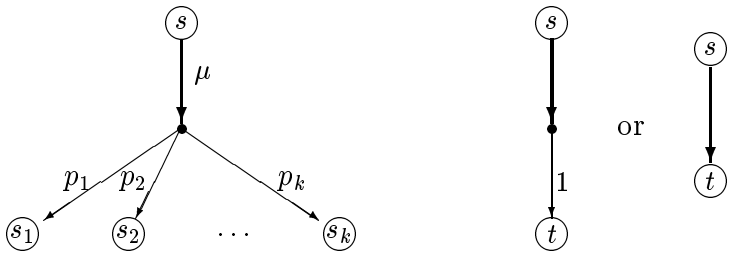
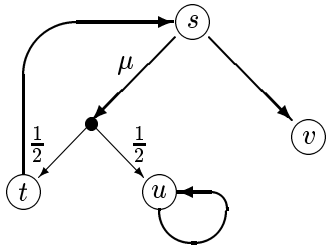


Figure 3.1: Pictures for the transition $s \longrightarrow \mu$

choice. The picture on the left of Figure 3.1 stands for a step $s \longrightarrow \mu$ where $Supp(\mu) = \{s_1, \ldots, s_k\}$ and $\mu(s_i) = p_i > 0$, $i = 1, \ldots, k$. Transitions of the form $s \longrightarrow \mu_t^1$ are depict as shown in the pictures on the right.

Example 3.2.2 The picture on the right shows a simple example for a concurrent probabilistic system where non-deterministic choice is present only in state s. The states t and u are "deterministic" in the sense that t and u have unique successor states (as $Steps(\cdot)$ consists of a single distribution of the form μ_x^1 for some state x). The state v is terminal (as $Steps(v) = \emptyset$). ■



Further examples for concurrent probabilistic systems are given in the introduction: the simple communication protocol $Sender \| Receiver$ of Example 1.2.2 (page 20) and the roulette player of Example 1.2.3 (page 22).[5]

Some MDP-based models (such as *stratified transition systems* [vGSST90], the *alternating model* [HaJo89, Hans91] or *concurrent Markov chains* [Vard85, CoYa88]) capture the branching structure of the purely probabilistic choices and distinguish between probabilistic and non-probabilistic states. The behaviour in a probabilistic state is "purely probabilistic" which is described by a distribution on the state space while the behaviour in each other state s is "purely non-probabilistic" in the sense that none of the possible steps in s is randomized, i.e. $Steps(s)$ consists of distributions μ_t^1, $t \in S$. Formally, these models can be defined as special instances of concurrent probabilistic systems. We follow the notations of [vGSST90] and use the adjective "stratified" for these models.[6]

Definition 3.2.3 [Stratified system] *A* stratified system *is a concurrent probabilistic system $(S, Steps)$ such that for all $s \in S$:*

$$Steps(s) \subseteq \{\mu_t^1 : t \in S\} \text{ or } |Steps(s)| = 1.$$

Let $(S, Steps)$ be a stratified system. A state s is called *probabilistic* iff $Steps(s) = \{\mu\}$ for some distribution $\mu \notin \{\mu_t^1 : t \in S\}$. Otherwise, s is called *non-probabilistic*. Note that the system behaviour in a non-probabilistic state s might be non-deterministic (if

[5] Note that in the simple communication protocol of Example 1.2.2 (page 20), the states s_{init}, s_{del} and s_{ack} behave deterministically. Formally, the "non-deterministic" alternatives in these states can be described by singleton sets consisting of a distribution that returns the probability 1 for the unique successor state.

[6] See Section 3.6 (page 62) for the exact relation between our notion of a stratified system and the original notion by van Glabbeek et al [vGSST90].

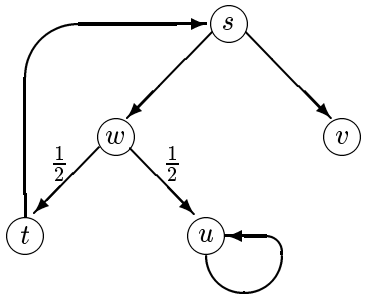
$|Steps(s)| \geq 2$) or deterministic (if $|Steps(s)| = 1$, in which case $Steps(s) = \{\mu_t^1\}$ for some state t) or s might be terminal (if $Steps(s) = \emptyset$).

Notation 3.2.4 [Stratified transition probabilities] *Let $(S, Steps)$ be a stratified system. Then, the transition probability function $\mathbf{P} : S \times S \to [0,1]$ is given by*

$$\mathbf{P}(s,t) \;=\; \begin{cases} \mu(t) & : \text{ if } s \text{ is probabilistic and } Steps(s) = \{\mu\} \\ 1 & : \text{ if } s \text{ is non-probabilistic and } s \longrightarrow t \\ 0 & : \text{ otherwise.} \end{cases}$$

We defined stratified systems as special instances of concurrent probabilistic systems. Nevertheless, the stratified view is as powerful as the concept of concurrent probabilistic systems where each state change involves the resolution of a non-deterministic choice (the choice for some $\mu \in Steps(\cdot)$) and a probabilistic choice (the randomized choice corresponding to the chosen distribution μ). If we devide these two choices into two "steps" then the behaviour of a concurrent probabilistic system can be described by a stratified system. Intuitively, in the stratified view, the small circle in the picture of a transition $s \longrightarrow \mu$ is viewed as a state where the system performs a randomized step. Formally, if $\mathcal{S} = (S, Steps)$ is a concurrent probabilistic system then $\mathcal{S}$ can be identified with the stratified system $\mathcal{S}' = (S', Steps')$ where $S' = S \cup \{(s,\nu) : s \in S, \nu \in Steps(s)\}$, $Steps'(s) = \{\mu^1_{(s,\nu)} : \nu \in Steps(s)\}$ and $Steps'(s,\nu) = \{\nu\}$. Of course, the resulting system $\mathcal{S}'$ can be simplified by removing states of the form (s, μ_t^1).[7]

Example 3.2.5 The system of Example 3.2.2 (page 39) can be modelled by the stratified system shown on the right. The state w represents the probabilistic choice that is resolved when in state s the transition μ is selected; i.e. w stands for the auxiliary state (s,μ). The states (t, μ_s^1), (u, μ_u^1), (s, μ_v^1) and (v, μ_v^1) are omitted. ■



3.2.1 Paths in concurrent probabilistic systems

Execution sequences (or paths) arise by resolving both the non-deterministic and probabilistic choices. Formally, an *execution fragment* or *finite path* in a concurrent probabilistic system $\mathcal{S} = (S, Steps)$ is a nonempty finite "sequence" $\sigma = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} s_2 \ldots \xrightarrow{\mu_k} s_k$ where $k \geq 0$ and $s_i \in S$, $\mu_i \in Steps(s_{i-1})$, $\mu_i(s_i) > 0$, $i = 1, 2, \ldots, k$.[8] σ is called *maximal* iff s_k is terminal. An *execution* or *fulpath* is either a maximal execution fragment or an infinite "sequence" $\pi = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} s_2 \xrightarrow{\mu_3} \ldots$ where $s_0, s_1, s_2, \ldots \in S$ and $\mu_i \in Steps(s_{i-1})$, $\mu_i(s_i) > 0$, $i = 1, 2, \ldots$. We use similar notations as in the fully probabilistic case (see page 34): If γ is a (finite or infinite) path in $\mathcal{S}$ then $\gamma(i)$ (the $(i+1)$-st state of γ), $\gamma^{(i)}$ (the i-th prefix of γ), $|\gamma|$ (the length of γ), $first(\gamma)$ (the first state of γ) and $inf(\gamma)$ (the set of states that occur infinitely often in γ) are defined as in the fully probabilistic case.

[7] Distributions of the form μ_t^1 do not really represent randomized steps as they yield a unique next state.

[8] Note that we write $s \to \mu$ to denote that μ is a possible step in s (i.e. $\mu \in Steps(s)$) and $s \xrightarrow{\mu} t$ to denote that μ is a possible step of s which leads (with non-zero probability) to the state t.

Similarly, for σ to be a finite path, $last(\sigma)$ denotes the last state of σ, $\sigma\uparrow$ the set of all fulpaths where σ is a prefix, $\sigma\uparrow_{fin}$ the set of finite paths σ' where σ is a prefix of σ' and $\sigma\downarrow$ the set of finite paths σ' where σ' is a prefix of σ. The prefix relation $\leq_{prefix}$, the proper prefix relation $<_{prefix}$, $\Sigma\uparrow$, $\Sigma\downarrow$, $\Sigma\uparrow_{fin}$ and "path concatenation" $\sigma\circ\gamma$ are defined as in the fully probabilistic case (see page 35). Moreover, for $\gamma = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} s_2 \xrightarrow{\mu_3} \ldots$ to be a (finite or infinite) path in $\mathcal{S}$ and $i < |\gamma|$, we put

$$step(\gamma, i) = \mu_{i+1}.$$

A state t is called *reachable* from s if there exists a finite path σ with $first(\sigma) = s$ and $last(\sigma) = t$. $Reach^{\mathcal{S}}(s)$ denotes the set of states which are reachable from s.

Example 3.2.6 For the system in Example 3.2.2 (page 39), $\sigma = s \xrightarrow{\mu} t \xrightarrow{\mu_s^1} s \xrightarrow{\mu_v^1} v$ is a fulpath with $first(\sigma) = \sigma(0) = s$, $last(\sigma) = \sigma(3) = v$, $\sigma(1) = t$, $\sigma(2) = s$, $step(\sigma, 0) = \mu$, $step(\sigma, 1) = \mu_s^1$, $step(\sigma, 2) = \mu_v^1$ and $|\sigma| = 3$. We have:

$$\sigma^{(2)} = s \xrightarrow{\mu} t \xrightarrow{\mu_s^1} s, \qquad \sigma^{(2)} \circ \left(s \xrightarrow{\mu} t\right) = \sigma^{(2)} \xrightarrow{\mu} t = s \xrightarrow{\mu} t \xrightarrow{\mu_s^1} s \xrightarrow{\mu} t$$

Moreover, $\sigma \circ v = \sigma$.[9] The states s, t, u, v are reachable from s and t while only v is reachable from v, only u is reachable from u. ■

$Path_{ful}^{\mathcal{S}}$ denotes the set of all fulpaths in $\mathcal{S}$, $Path_{fin}^{\mathcal{S}}$ the set of all finite paths in $\mathcal{S}$, and $Path_{ful}^{\mathcal{S}}(s)$ the set of fulpaths π with $first(\pi) = s$. When it is clear from the context what $\mathcal{S}$ is we abbreviate $Path_{ful}^{\mathcal{S}}$ by $Path_{ful}$, and similarly, $Path_{fin}^{\mathcal{S}}$ by $Path_{fin}$, $Path_{ful}^{\mathcal{S}}(s)$ by $Path_{ful}(s)$, and $Reach^{\mathcal{S}}(s)$ by $Reach(s)$. As in the fully probabilistic case, if Π is a set of fulpaths in $\mathcal{S}$ and $s \in S$ then $\Pi(s) = \Pi \cap Path_{ful}(s)$; if Σ is a set of finite paths then $\Sigma(s) = \Sigma \cap Path_{fin}(s)$.

3.2.2 Adversaries of concurrent probabilistic systems

We split a concurrent probabilistic system $\mathcal{S} = (S, Steps)$ into its *computation trees* (called *execution trees* in [HSP83] and *maximal resolutions* in [JoLa91]), with each component described as a fully probabilistic system. The computation trees arise by resolving the non-deterministic choices (but not the probabilistic choices). It is convenient to suppose that the "environment" (called *adversary* in [SeLy94, Sega95a], *policy* in the theory of MDPs, *scheduler* in [Vard85]) decides – based on the past history of the system – which of the possible steps to perform next. We follow the notations of [SeLy94, Sega95a] and use the word "adversary" to denote the instance that resolves the non-deterministic choices.[10]

[9] Here, the state v stands for a finite path of length 0.

[10] We only consider deterministic adversaries, i.e. those that schedule a unique next step. The notion of randomization of adversaries or probabilistic adversaries has been investigated in [HSP83] and [SeLy94, Sega95a], where it is shown that the probability of a measurable set Π with respect to a randomized adversary is a convex combination of the measure of Π with respect to non-randomized adversaries, and hence lies between the minimal and maximal measure of Π with respect to non-randomized adversaries. Since we are only interested in the maximal and minimal measures, we shall not need the randomized adversaries.

Definition 3.2.7 [Adversary, simple adversary] *Let $\mathcal{S} = (S, Steps)$ be a concurrent probabilistic system. An* adversary *of $\mathcal{S}$ is a function $A : Path_{fin} \to Distr(S)$ such that $A(\sigma) \in Steps(last(\sigma))$ for all $\sigma \in Path_{fin}$. An adversary A of $\mathcal{S}$ is called* simple *iff for every state $s \in S$ there exists $\mu_s \in Steps(s)$ with $A(\sigma) = \mu_{last(\sigma)}$ for all $\sigma \in Path_{fin}$ where $last(\sigma) = s$.*

$\mathcal{A}dv^{\mathcal{S}}$ denotes the set of all adversaries of $\mathcal{S}$ and $\mathcal{A}dv^{\mathcal{S}}_{simple}$ the set of simple adversaries. When clear from the context we write $\mathcal{A}dv$ and $\mathcal{A}dv_{simple}$ rather than $\mathcal{A}dv^{\mathcal{S}}$ and $\mathcal{A}dv^{\mathcal{S}}_{simple}$.

An adversary chooses for every finite path σ in $\mathcal{S}$ an outgoing transition from $last(\sigma)$. Simple adversaries resolve the non-determinism by selecting for every state a next step which is executed whenever the state s is reached – independent of the past history.[11]

Example 3.2.8 The system of Figure 3.2.2 (page 39) has exactly two simple adversaries A, B. These are given by $A(s) = \mu$, $B(s) = \mu^1_v$. Note that the other states do not behave non-deterministically because there is at most one outgoing transition. ■

Given an adversary A, the "behaviour" of $\mathcal{S}$ under A can be described by a bounded fully probabilistic system $\mathcal{S}^A$.

Notation 3.2.9 [The fully probabilistic system $\mathcal{S}^A$] *If $A \in \mathcal{A}dv$ then*

$$\mathcal{S}^A \;=\; \left(Path^{\mathcal{S}}_{fin}, \mathbf{P}^A\right)$$

is the fully probabilistic system where $\mathbf{P}^A$ is given by $\mathbf{P}^A(\sigma, \sigma \xrightarrow{A(\sigma)} s) = A(\sigma)(s)$ and $\mathbf{P}^A(\cdot) = 0$ in all other cases.

Note that, in general, $\mathcal{S}^A$ is infinite even if $\mathcal{S}$ is finite. If A a simple adversary then $\mathcal{S}^A$ can be identified with the fully probabilistic system $(S, \mathbf{A})$ where $\mathbf{A}(s,t) = A(s)(t)$ for all s, $t \in S$. For $\mathcal{S}$ to be finite and $A \in \mathcal{A}dv_{simple}$, the associated fully probabilistic system $\mathcal{S}^A = (S, \mathbf{A})$ is finite. For an adversary A of a concurrent probabilistic system $\mathcal{S} = (S, Steps)$ and σ to be an execution fragment, we define:

- $Path^A_{ful}$ denotes the set of all paths $\pi \in Path_{ful}$ with $step(\pi, i) = A(\pi^{(i)})$ for all $i \geq 0$.
- $Path^A_{fin}$ is the set of all finite paths $\sigma \in Path_{fin}$ with $step(\sigma, i) = A(\sigma^{(i)})$ for all $i < |\sigma|$.
- $\sigma \uparrow^A = \{\pi \in Path^A_{ful} : \sigma \leq_{prefix} \pi\}$ and $\sigma \uparrow^A_{fin} = \{\sigma' \in Path^A_{fin} : \sigma \leq_{prefix} \sigma'\}$
- If Π is a set of fulpaths in $\mathcal{S}$ then $\Pi^A = \Pi \cap Path^A_{ful}$.
- If Σ is a set of finite paths in $\mathcal{S}$ then

$$\Sigma^A = \Sigma \cap Path^A_{fin}, \quad \Sigma \uparrow^A = \bigcup_{\sigma \in \Sigma} \sigma \uparrow^A, \quad \Sigma \uparrow^A_{fin} = \bigcup_{\sigma \in \Sigma} \sigma \uparrow^A_{fin} .$$

- $Reach^A(s) = \{last(\sigma) : \sigma \in Path^A_{fin}(s)\}$

Note that, in the notations introduced on page 41, $Path^A_{ful}(s) = Path_{ful}(s) \cap Path^A_{ful}$, $Path^A_{fin}(s) = Path_{fin}(s) \cap Path^A_{fin}$, $\Pi^A(s) = \Pi \cap Path^A_{ful}(s)$ and $\Sigma^A(s) = \Sigma \cap Path^A_{fin}(s)$.

[11] In some sense, simple adversaries are extremely unfair and would be ruled out for practical purposes. We need them only for the sake of convenience.

We identify each (finite or infinite) path $\gamma = \sigma_0 \to \sigma_1 \to \ldots$ in $\mathcal{S}^A$ which starts in a state $s_0 \in S$ (i.e. $\sigma_0 = s_0$ is a path of length 0) with the path $last(\sigma_0) \xrightarrow{A(\sigma_0)} last(\sigma_1) \xrightarrow{A(\sigma_1)} \ldots$ in $\mathcal{S}$. Vice versa, if $\gamma \in Path^A_{ful} \cup Path^A_{fin}$ then we identify γ with the path $\gamma^{(0)} \to \gamma^{(1)} \to \gamma^{(2)} \to \ldots$ in $\mathcal{S}^A$. This yields a one-to-one correspondence between the paths $\gamma \in Path^A_{fin}(s) \cup Path^A_{ful}(s)$ and the paths in $\mathcal{S}^A$ that start in s. Hence, the probability measure $Prob$ on $Path^{\mathcal{S}^A}_{ful}(s)$ (defined as in Section 3.1 on page 35) turns $Path^A_{ful}(s)$ into a probability space. If $\Pi \subseteq Path_{ful}(s)$ and Π^A is measurable then we refer to $Prob(\Pi^A)$ as the measure of Π with respect to A.

Example 3.2.10 For the system of Example 3.2.2 (page 39) and the finite path σ of Example 3.2.6 (page 41), we have $\sigma \in Path^A_{fin}(s)$ for each adversary A with

$$A(s) = \mu \quad \text{and} \quad A\left(s \xrightarrow{\mu} t \xrightarrow{\mu^1_s} s\right) = \mu^1_v.$$

Moreover, for each such adversary A, the probability measure of $\sigma \uparrow^A$ is $1/2$. ■

Theorem 3.2.11 *Let $(S, Steps)$ be a concurrent probabilistic system, S_1, $S_2 \subseteq S$ and let Σ be the set of finite paths σ such that $\sigma(i) \in S_1$, $i = 0, 1, \ldots, |\sigma| - 1$, and $last(\sigma) \in S_2$. For $s \in S$ and $A \in \mathcal{A}dv$, let*

$$p^{min}(s) \;=\; \inf_{A \in \mathcal{A}dv} Prob\left(\Sigma^A(s) \uparrow^A\right), \quad p^{max}(s) \;=\; \sup_{A \in \mathcal{A}dv} Prob\left(\Sigma^A(s) \uparrow^A\right).$$

Then, p^{min} and p^{max} are the least fixed points of the operators

$$F^{min},\ F^{max} : (S \to [0,1]) \to (S \to [0,1])$$

that are defined as follows. If $s \in S_2$ then $F(f)(s) = 1$. If $s \in S \setminus (S_1 \cup S_2)$ then $F(f)(s) = 0$. If $s \in S_1 \setminus S_2$ then

$$F^{min}(f)(s) \;=\; \min\left\{ \sum_{t \in S} \mu(t) \cdot f(t) \;:\; \mu \in Steps(s) \right\},$$

$$F^{max}(f)(s) \;=\; \max\left\{ \sum_{t \in S} \mu(t) \cdot f(t) \;:\; \mu \in Steps(s) \right\}.$$

Proof: see Section 3.7, Corollary 3.7.4 (page 67). ■

Remark 3.2.12 [Computing the probabilities $p^{min}(s)$ and $p^{max}(s)$] Theorem 3.2.11 yields that the values $p^*(s)$ can be approximated with the following iterative method. We put $p^*_n(s) = 1$ if $s \in S_2$ and $p^*_n(s) = 0$ if $s \in S \setminus (S_1 \cup S_2)$, $n = 0, 1, 2, \ldots$. For $s \in S_1 \setminus S_2$, we define $p^*_0(s) = 0$ and, for $n = 0, 1, 2, \ldots$,

$$p^{min}_{n+1}(s) \;=\; \min\left\{\sum_{t \in S} \mu(t) \cdot p^{min}_n(t) : \mu \in Steps(s)\right\},$$

$$p^{max}_{n+1}(s) \;=\; \max\left\{\sum_{t \in S} \mu(t) \cdot p^{min}_n(t) : \mu \in Steps(s)\right\}.$$

Then, $\lim p^*_n(s) = p^*(s)$ for all $s \in S$. As proposed by [CoYa90, BidAl95], for S to be finite, the values $p^*(s)$ can also be computed by solving *linear optimization problems*

which can be solved in polynomial time with well-known methods of linear programming [Derm70, Bert87, Schr87].

Computation of $p^{max}(s)$: Let $S \setminus (S_1 \cup S_2) \subseteq S^{NO} \subseteq \{s \in S : \Sigma(s) = \emptyset\}$ and $S^? = S \setminus (S^{NO} \cup S_2)$. We define $y_s = 1$ if $s \in S_2$ and $y_s = 0$ if $s \in S^{NO}$. For each state $s \in S^?$, we choose a variable y_s. Then, the vector $(p^{max}(s))_{s \in S^?}$ is the unique solution of the linear minimization problem $0 \leq y_s \leq 1$ and

$$y_s \;\geq\; \sum_{t \in S} \mu(t) \cdot y_t, \quad \mu \in \mathit{Steps}(s)$$

where $\sum_{s \in S^?} y_s$ is minimal.

Computation of $p^{min}(s)$: Let $S^{NO} = \{s \in S : \Sigma(s) = \emptyset\}$ and $S^? = S \setminus (S^{NO} \cup S_2)$. We define $y_s = 1$ if $s \in S_2$ and $y_s = 0$ if $s \in S^{NO}$. Then, the vector $(p^{min}(s))_{s \in S^?}$ is the unique solution of the linear maximization problem $0 \leq y_s \leq 1$ and

$$y_s \;\leq\; \sum_{t \in S} \mu(t) \cdot y_t, \quad \mu \in \mathit{Steps}(s)$$

where $\sum_{s \in S^?} y_s$ is maximal.

The set $\{s \in S : \Sigma(s) = \emptyset\}$ can be obtained by a reachability analysis in the underlying directed graph. Note that for the computation of the values $p^{min}(s)$ it is essential that we deal with $S^{NO} = \{s \in S : \Sigma(s) = \emptyset\}$ (rather than e.g. $S^{NO} = S \setminus (S_1 \cup S_2)$ as it is possible for computing $p^{max}(\cdot)$). For instance, for the system $(\{s\}, \mathit{Steps})$ where $\mathit{Steps}(s) = \{\mu_s^1\}$ and $S_1 = \{s\}$, $S_2 = \emptyset$ we have $\Sigma = \emptyset$ (and hence $p^{min}(s) = 0$) while the optimization problem $0 \leq y_s \leq 1$ and

$$y_s \;\leq\; \sum_{t \in S} \mu(t) \cdot y_t, \quad \mu \in \mathit{Steps}(s)$$

where $\sum_t y_t$ is maximal yields $y_s = 1$ (because we just deal with the inequation $y_s \leq \sum_t \mu_s^1(t) \cdot y_t \;=\; y_s$). ■

Example 3.2.13 We consider the concurrent probabilistic system of Example 1.2.3 on page 22 (see Figure 1.3, page 22) that describes the one-day-behaviour of the roulette player. The maximal/minimal probabilities to reach the state s_{happy} (i.e. the state where the roulette player leaves the casino winning the last game) can be computed by taking $S_1 = \emptyset$ and $S_2 = \{s_{happy}\}$. By Theorem 3.2.11 (page 43), $p^* : S \to [0,1]$ is the least function $S \to [0,1]$ that satisfies $p^*(s_{happy}) = 1$, $p^*(s_{sad}) = 0$ and

$$p^*(s_{init}) = p^*(s_{play}) = p^*(s_{lost}) = \tfrac{1}{2} \cdot p^*(s_{won}) + \tfrac{1}{2} \cdot p^*(s_{lost}),$$

$$p^{min}(s_{won}) = \min\left\{p^{min}(s_{play}), \tfrac{1}{2}\right\}, \quad p^{max}(s_{won}) = \max\left\{p^{max}(s_{play}), \tfrac{1}{2}\right\}.$$

Thus, $p^{min}(s) = 0$ for all $s \in S \setminus \{s_{happy}\}$ and $p^{max}(s) = 1/2$ for all $s \in S \setminus \{s_{happy}, s_{sad}\}$. Note that the minimal probabilities $p^{min}(s) = 0$ are obtained by the simple adversary A that always chooses the transition $s_{won} \longrightarrow s_{play}$ (i.e. the pathological adversary which forces the roulette player to stay forever in the casino). For any other adversary A, the probability for s_{init} to reach s_{happy} is the maximal probability $p^{max}(s_{init}) = 1/2$. ■

3.2.3 Fairness of non-deterministic choice

In the verification of non-probabilistic concurrent systems, it is well-known that certain liveness properties can only be established when appropriate fairness assumptions about the resolution of the non-deterministic choices are made. Clearly, this also holds for concurrent probabilistic processes as they also allow for non-deterministic choice. Thus, as in the non-probabilistic case, certain (qualitative or quantitative) liveness properties cannot be established unless fairness of non-deterministic choice is imposed. For instance, for the roulette player of Example 1.2.3 on page 22 (see Figure 1.3, page 22) the quantitative liveness property stating that there is a 50% chance for the roulette player to leave the casino while winning the last game (i.e. eventually to reach the state s_{happy}) can only be established when fairness in the state s_{won} is assumed (see Example 3.2.13, page 44).

Fairness of non-deterministic choice (i.e. fairness of adversaries) of concurrent probabilistic systems was first introduced by Hart, Sharir & Pnueli [HSP83] and later considered by Vardi [Vard85] and several other authors. Fairness of non-deterministic choice requires that – in some sense – the environment (the adversary) resolves the non-deterministic choices in a fair manner. [HSP83] defines two types of fairness for adversaries: an adversary is *strictly fair* iff each of its fulpaths is fair, and it is *fair* if almost all execution sequences are fair (i.e. if the measure of its fair fulpaths is 1) where fairness of a fulpath can be defined as in the non-probabilistic case. [HSP83] deals with concurrent probabilistic systems which arise by the interleaving of sequential probabilistic processes and defines a fulpath π to be fair iff each sequential process is activated infinitely often in π (i.e. [HSP83] deals with "process fairness"). [Vard85] deals with "concurrent Markov chains" (stratified systems, see Definition 3.2.3, page 39) – which distinguish between non-deterministic and probabilistic states – and defines a fulpath π to be fair if all possible successor states of a non-deterministic state, in which fairness is required and which occur infinitely often in π, also occur infinitely often.

In this section, we follow the approachs of [HSP83, Vard85] and define fairness of adversaries. We adapt Vardi's notion of fairness to our model for concurrent probabilistic processes – which does not distinguish between non-deterministic and probabilistic states – and define an execution sequence π to be fair if none of the non-deterministic alternatives in a state occurring infinitely often in π is refused continuously. Moreover, we define W-fairness for a set W of states in which fairness is required.[12]

Definition 3.2.14 [Fairness for fulpaths] *Let $\mathcal{S} = (S, Steps)$ be a concurrent probabilistic system and π a fulpath in $\mathcal{S}$. π is called* fair *iff either π is finite or, for each $s \in inf(\pi)$ and each $\mu \in Steps(s)$, there are infinitely many indices i with $\pi(i) = s$ and $step(\pi, i) = \mu$.*

Remark 3.2.15 [Process fairness à la [HSP83]] Our notion of fairness of a fulpath is stronger than fairness of fulpaths in [HSP83]. In [HSP83] "process fairness" is considered, in the sense that all sequential processes (whose composition is the concurrent probabilistic system under consideration) are activated infinitely many times in fair fulpaths π. If $\mathcal{S}$ is a concurrent probabilistic system which arises through the interleaving of sequential

[12] An alternative notion of fairness for concurrent probabilistic systems and a discussion about the relation to our notion is presented in [dAlf97a].

processes without shared variables then fairness in the sense of Definition 3.2.14 (page 45) implies fairness in the sense of [HSP83]; to see this suppose that there are k sequential probabilistic processes $\mathcal{P}_1, \ldots, \mathcal{P}_k$ where each of them is described by a Markov chain $\mathcal{S}_i = (S_i, \mathbf{P}_i)$, $i = 1, \ldots, k$, and that $\mathcal{S} = (S, Steps)$ where $S = S_1 \times \ldots \times S_k$ and $Steps(s_1, \ldots, s_k) = \{\nu^i_{(s_1,\ldots,s_k)} : i = 1, \ldots, k\}$ where

$$\nu^i_{(s_1,\ldots,s_k)}(t_1, \ldots, t_k) \;=\; \begin{cases} \mathbf{P}_i(s_i, t_i) & : \text{ if } t_j = s_j,\ j = 1, \ldots, k,\ i \neq j \\ 0 & : \text{ otherwise.} \end{cases}$$

Then, whenever π is a fulpath in $\mathcal{S}$ that is fair in the sense of Definition 3.2.14 (page 45) then π is fair in the sense of [HSP83], which requires that for each $i \in \{1, \ldots, k\}$ there are infinitely many indices $j \geq 0$ with $step(\pi, j) = \nu^i_{\pi(j)}$. ■

Notation 3.2.16 [The set *Fair* of all fair fulpaths] *$Fair^{\mathcal{S}}$ (or shortly $Fair$) denotes the set of fair fulpaths in $\mathcal{S}$.*

As in [HSP83] we consider two kinds of fairness for adversaries: *strictly fair* adversaries, where *all* fulpaths are fair, and *fair* adversaries, where the set of fair paths has probability 1, i.e. where *almost all* fulpaths are fair.

Definition 3.2.17 [(Strict) fairness of adversaries, cf. [HSP83, Vard85]] *Let $\mathcal{S}$ be a concurrent probabilistic system and F an adversary for $\mathcal{S}$. F is called*

- strictly fair *iff* $Path^F_{ful} \subseteq Fair$,
- fair *iff* $Prob(Fair^F(s)) = 1$ *for all states s in $\mathcal{S}$.*

$\mathcal{A}dv^{\mathcal{S}}_{sfair}$ denotes the set of strictly fair adversaries, $\mathcal{A}dv^{\mathcal{S}}_{fair}$ the set of fair adversaries.

Clearly, strictly fair adversaries are fair. If F is a fair adversary then for each $\sigma \in Path^F_{fin}$ there exists $\pi \in Fair^F$ where σ is a prefix of π. This reflects "liveness" in the sense of Alpern & Schneider [AlSch84] which states that every finite computation can be extended to an infinite (fair) computation.

Example 3.2.18 For the system of Example 3.2.2 (page 39), the fulpath

$$\pi_0 \;=\; s \xrightarrow{\mu} t \xrightarrow{\mu^1_s} s \xrightarrow{\mu} t \xrightarrow{\mu^1_s} s \xrightarrow{\mu} \ldots$$

is not fair since $s \in inf(\pi_0)$ and $\mu^1_v \in Steps(s) \setminus \{step(\pi_0, i) : i \geq 0\}$. Every other fulpath $\pi \in Path_{ful}(s)$ is fair (as it "ends" in v or u). Thus,

$$Fair(s) \;=\; Path_{ful}(s) \setminus \{\pi_0\}.$$

The simple adversary B with $B(s) = \mu^1_v$ is strictly fair since $\pi_0 \notin Path^B_{ful}$. The simple adversary A with $A(s) = \mu$ is not strictly fair since $\pi_0 \in Path^A_{ful}(s)$. Nevertheless, A is fair. To see this, consider the set Π of all fulpaths $\pi \in Path_{ful}$ where $\pi(i) \in \{u, v\}$ for some i. Then, $\Pi^A(x) = Fair^A(x)$ for all states x and $Prob(Fair^A(u)) = Prob(Fair^A(v)) = 1$,

$$Prob\left(Fair^A(s)\right) \;=\; \sum_{i=0}^{\infty} \frac{1}{2} \cdot \left(\frac{1}{2}\right)^i \;=\; 1$$

and $Prob\left(Fair^A(t)\right) = Prob\left\{t \xrightarrow{\mu^1_s} \pi : \pi \in Fair^A(s)\right\} = 1$. Hence, A is fair. ■

Following Vardi [Vard85], the above definition of fair fulpaths or fair adversaries can be weakened by requiring fairness with respect to the non-deterministic choices only in certain states rather than in all states.

Definition 3.2.19 [W**-Fairness of fulpaths**] *Let $\mathcal{S} = (S, Steps)$ be a concurrent probabilistic system and $W \subseteq S$. A fulpath π in $\mathcal{S}$ is called* W-fair *iff, for all $s \in inf(\pi) \cap W$ and all $\mu \in Steps(s)$, there are infinitely many indices $j \geq 0$ with $step(\pi, j) = \mu$.*

Fairness with respect to $W = S$ (in the sense of Definition 3.2.19) is weaker than fairness of a fulpath in the sense of Definition 3.2.14 (page 45).[13] Vardi's notion of fairness of adversaries adapted to our model for concurrent probabilistic systems is the following.

Definition 3.2.20 [W**-Fairness of adversaries, cf. [Vard85]**] *Let $\mathcal{S}$ and W be as before. An adversary F is called* W-fair *iff, for all $s \in S$, the measure of the set of fulpaths $\pi \in Path^F_{ful}(s)$ which are W-fair is 1. $\mathcal{A}dv^{\mathcal{S}}_{Wfair}$ denotes the set of W-fair adversaries.*

When clear from the context, we write $\mathcal{A}dv_{sfair}$, $\mathcal{A}dv_{fair}$ or $\mathcal{A}dv_{Wfair}$ rather than $\mathcal{A}dv^{\mathcal{S}}_{sfair}$, $\mathcal{A}dv^{\mathcal{S}}_{fair}$ or $\mathcal{A}dv^{\mathcal{S}}_{Wfair}$. Clearly, $\mathcal{A}dv_{sfair} \subseteq \mathcal{A}dv_{fair} \subseteq \mathcal{A}dv_{Wfair}$.

3.3 Labelled probabilistic systems

Formal reasoning about the behaviour of programs requires additional informations about the states and/or the transitions. In the literature two kinds of labellings have been established: one uses *atomic propositions* (or, more general, first order logical formulas) as labels for the states, the other uses *action labels* for the transitions. Models based on the former type of labellings are often called *Kripke structures* and used in the context of temporal logic specifications while the models based on the latter type of labellings are often called *labelled transition systems* and used in the context of process algebras and implementation relations. Several authors proposed transformation techniques between proposition-labelled and action-labelled systems, see e.g. [JHP89, dNVa90]. Even though they are originally formulated for non-probabilistic systems they can also be applied in the probabilistic case. We follow these standard approachs and use action labels for the transitions in Chapters 4, 5, 6 and 7 where we work with process calculi and implementation relations and proposition labels for the states in Chapter 9 where we deal with temporal logic specifications.

3.3.1 Action-labelled probabilistic systems

In the action-labelled approach one usually deals with a set Act of abstract action symbols. Each action symbol a represents an activity of the program that is viewed to be "atomic" in the sense that it cannot be interleaved by actions of programs which run in parallel. Typical examples are communication actions like sending or receiving a message along a certain channel.

[13] Note that in Definition 3.2.19 we do not require that $step(\pi, j) = \mu$ and $\pi(j) = s$.

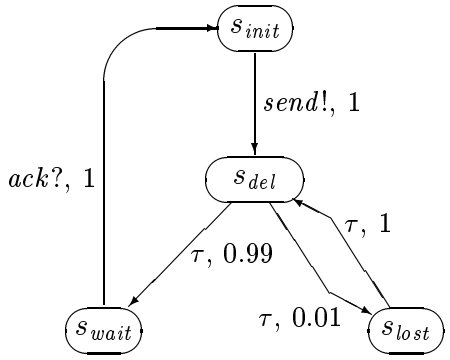


Figure 3.2: The sender with action labels

In (non-probabilistic) labelled transition systems, the possible steps in the states are described by a transition relation $\longrightarrow \subseteq S \times Act \times S$ (where Act stands for the underlying set of actions), i.e. the state changes are associated with action labels. Intuitively, $s \xrightarrow{a} t$ asserts that, in state s, it is possible to perform the action a and to reach state t afterwards. Hence, for fixed state s, the set $\{(a,t) : s \xrightarrow{a} t\}$ represents the non-deterministic alternatives in state s. This can be adapted for probabilistic systems as follows: In the concurrent case, the transitions are associated with action labels (i.e. one deals with $Steps(s)$ to be a set of pairs (a,μ) where a is an action label and μ a distribution on the state space.) In the fully probabilistic case, the arguments of the transition probability function $\mathbf{P}$ are extended by an action (i.e. one deals with the probabilities $\mathbf{P}(s,a,t)$ for state s to perform the action a and to reach t afterwards).

The action set Act**:** Throughout all sections, Act stands for a nonempty set of *actions*. For $L \subseteq Act$, L^* denotes the set of finite sequences over L. The empty sequence is denoted by ε. L^+ denotes the set of finite nonempty sequences over Act, i.e. $L^+ = L^* \setminus \{\varepsilon\}$. In Chapters 4 and 5, we assume that Act contains a special symbol τ. Intuitively, τ stands for any "internal" activity of the system which is invisible for an observer (or the environment of the system). We refer to τ as the *internal* action. The other actions are called *visible*. We use greek letters $\alpha, \beta, \ldots$ to denote visible actions and arabic letters $a, b, \ldots$ to range over all actions.

Definition 3.3.1 [Action-labelled fully probabilistic systems] *An* action-labelled fully probabilistic system *is a tuple* $(S, Act, \mathbf{P})$ *consisting of a set* S *of states, a nonempty set* Act *of actions and a transition probability function* $\mathbf{P} : S \times Act \times S \to [0,1]$ *such that, for each* $s \in S$, $\mathbf{P}(s,a,t) > 0$ *for at most countably many pairs* $(a,t) \in Act \times S$ *and* $\sum_{a,t} \mathbf{P}(s,a,t) \in \{0,1\}$.

Example 3.3.2 [The sender with action labels] Figure 3.2 (page 48) shows an action-labelled extension of the simple communication protocol of Example 1.2.1 (page 19). We use the visible actions $send!$ (an output action by which the sender passes the message to the medium) and $ack?$ (an input action which stands for the receipt of the acknowledgement). The other steps are supposed to be invisible and thus labelled by the special action symbol τ. ■

Let $(S, Act, \mathbf{P})$ be an action-labelled fully probabilistic system. For $s \in S$, $C \subseteq S$, $a \in Act$

and $L \subseteq Act$, we define

$$\mathbf{P}(s,a,C) \;=\; \sum_{t\in C} \mathbf{P}(s,a,t), \quad \mathbf{P}(s,a) = \mathbf{P}(s,a,S), \quad \mathbf{P}(s,L) = \sum_{a\in L} \mathbf{P}(s,a).$$

An *execution fragment* or *finite path* is a finite "sequence" $\sigma \;=\; s_0 \stackrel{a_1}{\rightarrow} s_1 \stackrel{a_2}{\rightarrow} s_2 \stackrel{a_2}{\rightarrow} \ldots \stackrel{a_k}{\rightarrow} s_k$ such that $s_0, s_1, \ldots, s_k \in S$, $a_1, \ldots, a_k \in Act$ and $\mathbf{P}(s_{i-1}, a_i, s_i) > 0$, $i = 1, \ldots, k$. Maximality, $\sigma(i)$, $\sigma^{(i)}$, $first(\sigma)$, $last(\sigma)$, $|\sigma|$, $\mathbf{P}(\sigma)$, $\sigma \uparrow$ are defined as in the unlabelled case (see page 34 ff). If σ is as above then we put $trace(\sigma) = a_1 a_2 \ldots a_k$. An *execution* or *fulpath* in $(S, Act, \mathbf{P})$ is either a maximal execution fragment or an infinite "sequence" $\pi \;=\; s_0 \stackrel{a_1}{\rightarrow} s_1 \stackrel{a_2}{\rightarrow} s_2 \stackrel{a_2}{\rightarrow} \ldots$ where $s_0, s_1, \ldots, \in S$, $a_1, a_2, \ldots \in Act$ and $\mathbf{P}(s_{i-1}, a_i, s_i) > 0$, $i = 1, 2, \ldots$. As before, a path denotes an execution fragment or execution. For π to be an infinite path, $\pi(i)$, $\pi^{(i)}$, $first(\pi)$ and $|\pi|$ are defined in the obvious way.

Example 3.3.3 For the system of Figure 3.2 (page 48),

$$\pi \;=\; s_{init} \stackrel{send!}{\longrightarrow} s_{del} \stackrel{\tau}{\longrightarrow} s_{lost} \stackrel{\tau}{\longrightarrow} s_{del} \stackrel{\tau}{\longrightarrow} s_{lost} \stackrel{\tau}{\longrightarrow} \ldots$$

is an execution (fulpath) with $\pi^{(2)} \;=\; s_{init} \stackrel{send!}{\longrightarrow} s_{del} \stackrel{\tau}{\longrightarrow} s_{lost}$, $first(\pi) = s_{init}$, $\pi(3) = s_{del}$, $\mathbf{P}(\pi^{(4)}) \;=\; 1 \cdot 0.01 \cdot 1 \cdot 0.01 = 0.0001$ and $trace(\pi^{(4)}) = send!\ \tau\ \tau\ \tau$. ■

The probabilities $Prob(s, \Omega, C)$**:** The sigma-field $SigmaField(s)$ and the probability measure $Prob$ are defined as in the unlabelled case (Section 3.1, page 35). For $s \in S$, $\Omega \subseteq Act^*$ and $C \subseteq S$, we define $Prob(s, \Omega, C)$ to be the probability for s to reach C via an execution fragment that is labelled by some string of Ω. The formal definition of $Prob(s, \Omega, C)$ is as follows. Let $Path_{fin}(s, \Omega, C)$ be the set of finite paths $\sigma \in Path_{fin}(s)$ $trace(\sigma) \in \Omega$ and $last(\sigma) \in C$. Let $Path_{ful}(s, \Omega, C) = \bigcup_{\sigma \in Path_{fin}(s,\Omega,C)} \sigma \uparrow$, $Path_{ful}(s, \Omega)$ $= Path_{ful}(s, \Omega, S)$ and $Path_{ful}(s, \Omega, t) = Path_{ful}(s, \Omega, \{t\})$. We define $Prob(s, \Omega, C) =$ $Prob(\ Path_{ful}(s, \Omega, C)\ )$, $Prob(s, \Omega, t) = Prob(s, \Omega, \{t\})$ and $Prob(s, \Omega) = Prob(s, \Omega, S)$.

Proposition 3.3.4 *Let $(S, Act, \mathbf{P})$ be an action-labelled fully probabilistic system and $C \subseteq S$. The function $S \times 2^{Act^*} \rightarrow [0,1]$, $(s, \Omega) \mapsto Prob(s, \Omega, C)$, is the least fixed point of the operator $F : \Big(S \times 2^{Act^*} \rightarrow [0,1]\Big) \rightarrow \Big(S \times 2^{Act^*} \rightarrow [0,1]\Big)$ which is defined as follows. $F(f)(s, \Omega) \;=\; 1$ if $s \in C$ and $\varepsilon \in \Omega$. If $s \notin C$ or $\varepsilon \notin \Omega$ then*

$$F(f)(s, \Omega) \;=\; \sum_{(a,t)\in Act\times S} \mathbf{P}(s,a,t) \cdot f(t, \Omega/a, C)$$

where $\Omega/a = \{x : ax \in \Omega\}$.[14] *If S is finite and $S^{NO} = \{s \in S : Path_{ful}(s, \Omega, C) = \emptyset\}$, $S^{YES} = C$ if $\varepsilon \in \Omega$, $S^{YES} = \emptyset$ if $\varepsilon \notin \Omega$ and $S^{?} = S \setminus (S^{NO} \cup S^{YES})$ then the function $(s, \Omega) \mapsto Prob(s, \Omega, C)$ is the unique fixed point of the operator $F' : \Big(S \times 2^{Act^*} \rightarrow [0,1]\Big) \rightarrow \Big(S \times 2^{Act^*} \rightarrow [0,1]\Big)$ which is defined by:*

$$F'(f)(s, \Omega) \;=\; \begin{cases} F(f)(s, \Omega) & : \ \text{if } s \in S^{?} \\ 1 & : \ \text{if } s \in S^{YES} \\ 0 & : \ \text{if } s \in S^{NO} \end{cases}$$

where F is defined as above.

[14] Recall that ε denotes the empty word in Act^*.

Proof: easy verification. Uses Theorem 3.1.6 (page 36). ■

The probabilities $Prob(s, \tau^*, C)$ and $Prob(s, \tau^*\alpha\tau^*, C)$: In what follows, we identify a regular expression (e.g. τ^*, $\tau^*\alpha$ or $\tau^*\alpha\tau^*$) with the corresponding set of traces. For instance, $Prob(s, \tau^*, t)$ denotes the probability to reach t from s via internal actions, $Prob(s, a_1a_2 \ldots a_k)$ stands for the probability for s to perform the trace $a_1a_2 \ldots a_k$. Clearly, for finite action-labelled fully probabilistic systems and regular expressions of the form τ^*, $\tau^*\alpha$ and $\tau^*\alpha\tau^*$, the second part of Proposition 3.3.4 yields that the probabilities $Prob(s, \Omega, C)$ can be computed by a reachability analysis in the underlying directed graph (which yields S^{NO} and $S^?$) and solving a linear equation system.

Example 3.3.5 Consider the simple communication protocol of Example 3.3.2 on page 48 (Figure 3.2, page 48). The probability $Prob(s_{init}, \tau^*\ send!\ \tau^*, s_{wait})$ can be computed by solving the linear equation system:

$$\begin{aligned} x_{init} &= 1 \cdot y_{del} \\ y_{del} &= 0.01 \cdot y_{lost} + 0.99 \cdot y_{wait} \\ y_{lost} &= 1 \cdot y_{del}, \quad y_{wait} = 1 \end{aligned}$$

(Here, $y_* = Prob(s_*, \tau^*, s_{wait})$.) We get $Prob(s_{init}, \tau^*\ send!\ \tau^*, s_{wait}) = x_{init} = 1$. ■

Next we extend concurrent probabilistic systems by action labels. For this, each transition in the system is associated with an action label. I.e. we deal with a function *Steps* that assigns to each state s a set of pairs (a, μ) where a is an action and μ a distribution on the state space. Thus, action-labelled concurrent probabilistic systems are associated with a transition relation $\longrightarrow \subseteq S \times Act \times Distr(S)$.

Definition 3.3.6 [Action-labelled concurrent probabilistic system] *An* action- labelled concurrent probabilistic system *is a tuple $(S, Act, Steps)$ where S is a set of states, Act a nonempty set of actions and $Steps : S \to 2^{Act \times Distr(S)}$ a function which assigns to each state s a set $Steps(s)$ of pairs $(a, \mu) \in Act \times Distr(S)$.*

Let $\mathcal{S} = (S, Act, Steps)$ be an action-labelled concurrent probabilistic system. $\mathcal{S}$ is called *finite* iff S, Act and $\bigcup_{s \in S} Steps(s)$ are finite. We write $s \xrightarrow{a} \mu$ iff $s \in S$, $a \in Act$ and $(a, \mu) \in Steps(s)$ and refer to $s \xrightarrow{a} \mu$ as a *transition* or a *step* of s. If μ is of the form μ_t^1 then we also write $s \xrightarrow{a} t$ rather than $s \xrightarrow{a} \mu_t^1$. As in the unlabelled case, for each state s, the elements of $Steps(s)$ represent the non-deterministic alternatives in the state s. Given a state s, an adversary chooses some outgoing transition $s \xrightarrow{a} \mu$. Then, the action a is performed and the next state is chosen randomly according to the distribution μ.

Example 3.3.7 [The communication protocol with action labels] The simple communication protocol of Example 1.2.2 (page 20) can be extended by action labels as shown in Figure 3.3 on page 51. Here, we assume $Act = \{produce, consume, try\}$ where *produce* stands for the action by which the sender generates a message and passes the message to the medium, *consume* for the action by which the receiver reads and works up the message and acknowledges the receipt while *try* represents the actions by which the medium tries to deliver the message. ■

Paths and adversaries of action-labelled concurrent probabilistic systems are defined as in the unlabelled case (see Sections 3.2.1 and 3.2.2) where the action labels are added.

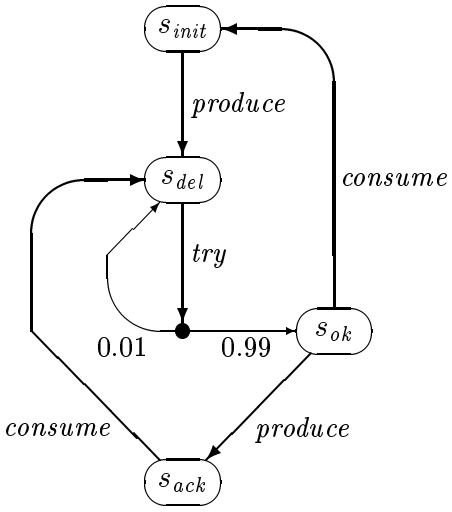


Figure 3.3: The simple communication protocol with action labels

For instance, a path is of the form $s_0 \xrightarrow{a_1,\mu_1} s_1 \xrightarrow{a_2,\mu_2} \ldots$, adversaries are functions that take a finite path σ as their input and return a step of the last state of σ, i.e. a pair $A(\sigma) = (a, \mu) \in \mathit{Steps}(\mathit{last}(\sigma))$.

Non-probabilistic labelled transition systems (where the transition relation $\longrightarrow$ is a subset of $S \times \mathit{Act} \times S$) arise as special cases of action-labelled concurrent probabilitistic systems by identifying each "non-probabilistic transition" $s \xrightarrow{a} t$ with the "probabilistic transition" $s \xrightarrow{a} \mu_t^1$. I.e. the non-probabilistic labelled transition system $(S, \mathit{Act}, \longrightarrow)$ corresponds to the probabilistic system $(S, \mathit{Act}, \mathit{Steps})$ where $\mathit{Steps}(s) = \left\{(a, \mu_t^1) : s \xrightarrow{a} t\right\}$.

Let $\mathcal{S} = (S, \mathit{Act}, \mathit{Steps})$ be an action-labelled concurrent probabilistic system.

Notation 3.3.8 [The sets $\mathit{Steps}_a(s)$ and $\mathit{act}(s)$] *For $s \in S$ and $a \in \mathit{Act}$, let*

$$\mathit{Steps}_a(s) = \{\mu : s \xrightarrow{a} \mu\}, \quad \mathit{act}(s) = \{a \in \mathit{Act} : \mathit{Steps}_a(s) \neq \emptyset\}.$$

Definition 3.3.9 [Finitely branching, image-finite systems] *$\mathcal{S}$ is called*

- finitely branching *iff, for each $s \in S$, $\mathit{Steps}(s)$ is finite,*
- image-finite *iff, for each $s \in S$ and $a \in \mathit{Act}$, $\mathit{Steps}_a(s)$ is finite.*

Definition 3.3.10 [Reactive systems, cf. [LaSk89, vGSST90]] *$\mathcal{S}$ is called* reactive *iff, for each $s \in S$ and $a \in \mathit{Act}$, $|\mathit{Steps}_a(s)| \leq 1$.*

The use of reactive systems is motivated by the assumption that the system "reacts" on the stimuli of the environment which offers the communication on certain actions. The choice between several (different) actions is not under the control of the system (hence, no probabilistic assumptions are – or can be – made about the resolution of the choice between the actions) while the choice between the several branches of the same action is resolved randomly according to a certain distribution. For further details about the reactive view see [LaSk89, vGSST90].

In what follows, we often describe reactive systems as tuples $(S, \mathit{Act}, \mathbf{P})$ where $\mathbf{P} : S \times \mathit{Act} \times S \to [0, 1]$ returns the probability $\mathbf{P}(s, a, t)$ for the transition $s \xrightarrow{a} t$ (if it exists).

Notation 3.3.11 [Reactive transition probabilities] *If $\mathcal{S}$ is reactive then the induced transition probability function* $\mathbf{P}$ *is given by* $\mathbf{P}(s,a,t) = 0$ *if* $Steps_a(s) = \emptyset$ *and* $\mathbf{P}(s,a,t) = \mu(t)$ *if* $Steps_a(s) = \{\mu\}$.

For the extension of stratified systems (Definition 3.2.3, page 39) by action labels we make the requirement that the transition of the probabilistic states are labelled by a special action symbol a_{random} that stands for any activity that resolves a probabilistic choice (e.g. "tossing a fair coin").

Definition 3.3.12 [Action labels in stratified systems] *An* action-labelled stratified system *is an action-labelled concurrent probabilistic system* $(S, Act, Steps)$ *such that* Act *contains the special action* a_{random} *and, for all* $s \in S$*:*

- *either* $a_{random} \notin act(s)$ *and* $Steps(s) \subseteq \{(a, \mu_t^1) : t \in S, a \in Act\}$
- *or* $\{a_{random}\} = act(s)$ *and* $|Steps(s)| = 1$.

Thus, for any probabilistic state s of an action-labelled stratified system, $Steps(s) = \{(a_{random}, \mu)\}$ for some distribution μ.

3.3.2 Proposition-labelled probabilistic systems

In the proposition-labelled approach, a state is viewed as a function which assigns values to the program and control variables. In many applications, it suffices to abstract from the exact values of certain (or all) variables and just to work with assertions about the values of certain variables. Typically, these assertions are formulated in a first order or propositional logical framework. For instance, we can use atomic propositions of the form $a_{x=v}$ which states that the current value of variable x is v or $a_{x<v}$ which states that the current value of variable x is less than v. In many applications, it suffices to use a finite nonempty set AP of atomic propositions and to deal with a labelling function $\mathcal{L} : S \to 2^{AP}$ that assigns to each state s the set $\mathcal{L}(s)$ of atomic propositions that are satisfied in s.

Definition 3.3.13 [Proposition-labelled probabilistic systems] *A* proposition-labelled probabilistic system *is a tuple* $(\mathcal{S}, AP, \mathcal{L})$ *consisting of a (fully or concurrent) probabilistic system* $\mathcal{S}$*, a finite nonempty set* AP *of atomic propositions and a labelling function* $\mathcal{L} : S \to 2^{AP}$ *which assigns to each state* $s \in S$ *a set* $\mathcal{L}(s)$ *of atomic propositions.*

Example 3.3.14 [Sock selection problem] We briefly explain how to use proposition-labelled probabilistic systems to specify the behaviour of randomized algorithms. We consider the "sock-selection problem" of [GSB94]. The starting point is a dresser drawer with $2n$ socks, n read socks and n blue socks. The problem is to extract a matching pair of socks (i.e. two red socks or two blue socks) where it is not allowed to have at hand more than two socks at any time and where a sock, once extracted from the drawer, cannot be put back in the drawer. The randomized method of [GSB94] can be sketched as follows. We extract the first two socks. If we do not have a matching pair then we choose randomly one of the two socks, threw it away and replace it by the next sock from the drawer. We proceed in this way until we have a matching pair or no more socks are in the drawer. As shown in [GSB94], the expected number of socks that have to be extracted

from the drawer until a matching pair is obtained is approximately 4. Nevertheless, there is a small chance that the algorithm fails (i.e. does not return a matching pair of socks). This can be seen by analyzing the induced Markov chain: Given a fixed sequence $sock_1, sock_2, \ldots, sock_{2n}$ of socks that represents the order in which the socks are extracted from the drawer the behaviour of the algorithm can be described by a fully probabilistic system. We use the state space $S = \{red, blue\} \times \{red, blue\} \times \{0, 1, \ldots, 2n-2\}$ where the first two components stand for the colors of the two socks we have in hand while the last component is the number of socks that are still in the drawer. The terminal states are those states $\langle c_1, c_2, k\rangle$ where either $c_1 = c_2$ (the states where we have a matching pair) or $k = 0$ (the states where the drawer is empty). If $c_1 \neq c_2$ and $k \geq 1$ then we have the transition probabilities

$$\mathbf{P}(\langle c_1, c_2, k\rangle, \langle c, c_2, k-1\rangle) \;=\; \mathbf{P}(\langle c_1, c_2, k\rangle, \langle c_1, c, k-1\rangle) \;=\; \tfrac{1}{2}$$

where $c = color(sock_{2n-k})$ is the color of $sock_{2n-k}$. Figure 3.4 (page 53) shows the fully probabilistic system that we obtain for $n = 2$ and the sequence $red, blue, red, blue$. For

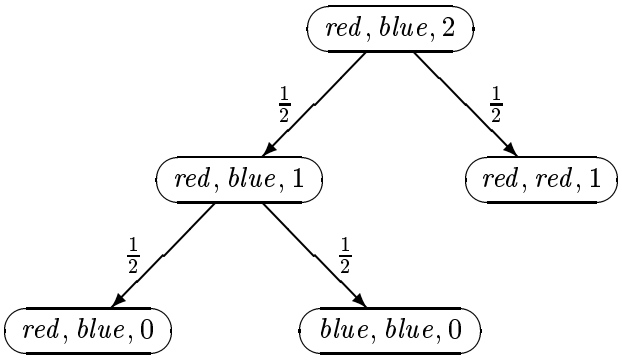


Figure 3.4: The fully probabilistic system for $n = 2$ and the sequence $red, blue, red, blue$

analyzing the correctness of the algorithm one might use a single atomic proposition *success* that characterizes the successful states (i.e. the states where a matching pair is found). Hence, we deal with the labelling function $\mathcal{L}$ where $success \in \mathcal{L}(\langle c_1, c_2, k\rangle)$ iff $c_1 = c_2$. Using Theorem 3.1.6 (page 36) or Proposition 3.1.7 (page 36), it can be shown that the probability to reach a *success*-labelled state from $\langle c_1, c_2, k\rangle$ (where $c_1 \neq c_2$) is $1 - 1/2^k$. By considering the initial state $s_{init} = \langle color(sock_1), color(sock_2), 2n-2\rangle$ we obtain that the probability to get a matching pair is $1 - 1/2^{2n-2}$. ■

3.4 Bisimulation and simulation

Bisimulation equivalence [Miln80, Park81] is one of the standard concepts to obtain a natural notion of "process equality", i.e. a notion of "behaves like". While bisimulation is "bi-directed" and asserts that each step of one process can be simulated by a step of the other process, simulation is "uni-directed" and states that for each step of the first process (the "implementation") there is a corresponding one of the second process (the "specification").

In this section, we recall the definition of bisimulation equivalence as introduced by Larsen & Skou [LaSk89] for reactive systems (and its modifications for action-labelled

concurrent probabilistic systems [SeLy94] and for action-labelled fully probabilistic systems [vGSST90]). Section 3.4.2 presents the notion of a simulation à la Segala & Lynch [SeLy94] for action-labelled concurrent probabilistic systems. Moreover, we show how to adapt this notion of a simulation for fully probabilistic systems with action labels.[15]

3.4.1 Bisimulation

In [LaSk89], Larsen & Skou introduce probabilistic bisimulation for reactive systems as an elegant extension of bisimulation for non-probabilistic systems [Miln80, Park81]. Van Glabbeek et al [vGSST90] reformulate probabilistic bisimulation for action-labelled fully probabilistic; Segala & Lynch [SeLy94] for action-labelled concurrent systems (which – when applied to reactive systems – yields the original definition by Larsen & Skou).

Definition 3.4.1 [Bisimulation (fully probabilistic case), cf. [vGSST90]] *A* bisimulation *on an action-labelled fully probabilistic system $(S, Act, \mathbf{P})$ is an equivalence relation R on S such that $\mathbf{P}(s,a,C) = \mathbf{P}(s',a,C)$ for all $(s,s') \in R$, all $a \in Act$ and $C \in S/R$.*

Definition 3.4.2 [Bisimulation (concurrent case), cf. [SeLy94]] *A* bisimulation *on an action-labelled concurrent probabilistic system $(S, Act, Steps)$ is an equivalence relation R on S such that for all $(s,s') \in R$:*

If $s \xrightarrow{a} \mu$ then there is a transition $s' \xrightarrow{a} \mu'$ with $\mu[C] = \mu'[C]$ for all $C \in S/R$.

Definition 3.4.3 [Bisimulation equivalence $\sim$] *Two states s_1 and s_2 of an action-labelled (fully or concurrent) probabilistic system are called* bisimilar *(denoted by $s_1 \sim s_2$) iff there exists a bisimulation which contains (s_1, s_2).*

Clearly, the above notion of a bisimulation equivalence applied to a non-probabilistic system $(S, Act, \longrightarrow)$ (identified with the concurrent probabilistic system $(S, Act, Steps)$ where $Steps_a(s) = \{\mu_t^1 : s \xrightarrow{a} t\}$) coincides with the classical bisimulation equivalence à la [Miln80, Park81].[16] Jonsson & Larsen [JoLa91] give an alternative description of bisimulation for fully probabilistic systems with proposition labels which is based on *weight functions* for distributions.[17] The following proposition reformulates this result (Theorem 4.6 in [JoLa91]) for concurrent probabilistic systems with action labels. A similar observation was made by de Vink & Rutten [dViRu97] for reactive systems (using a categorical characterization of what we call weight functions).

Proposition 3.4.4 *Let $(S, Act, Steps)$ be an action-labelled concurrent probabilistic system and s, $s' \in S$. Then, s and s' are bisimilar iff there exists a binary relation R on S such that $(s,s') \in R$ and, for all $(t,t') \in R$:*[18]

[15] Bisimulation equivalence and the simulation preorder can also be defined for proposition-labelled probabilistic systems (see e.g. [JoLa91, ASB+95]). These definitions are omitted here.

[16] This simple observation should not be confused with the more delicate result by Bloom & Meyer [BlMe89] who have shown that any finitely branching non-probabilistic action-labelled transition system can be decorated with probabilities such that the resulting system is a *reactive* system with the same bisimulation equivalence classes.

[17] See Section 2.2, page 30, for the definition of weight functions.

[18] Recall that $\mu \preceq_R \mu'$ iff there exists a weight function for (μ, μ') with respect to R (see Section 2.2, page 30).

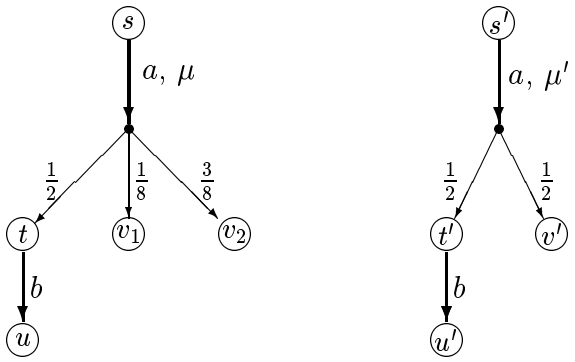


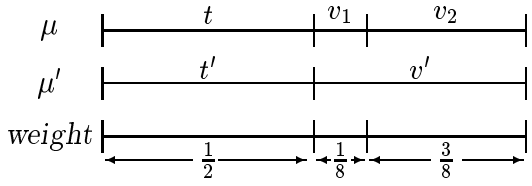
Figure 3.5: $s \sim s'$

- *If* $t \xrightarrow{a} \mu$ *then there exists* $t' \xrightarrow{a} \mu'$ *with* $\mu \preceq_R \mu'$.
- *If* $t' \xrightarrow{a} \mu'$ *then there exists* $t \xrightarrow{a} \mu$ *with* $\mu \preceq_R \mu'$.

Proof: easy verification. Uses similar arguments to those in [JoLa91, dViRu97]. ■

Proposition 3.4.4 yields that two states s, s' are bisimilar iff for each transition $s \xrightarrow{a} \mu$ of s there is a transition $s' \xrightarrow{a} \mu'$ where $\mu \preceq_\sim \mu'$. In that case, the weight function *weight* for (μ, μ') with respect to $\sim$ shows how to combine parts of bisimilar states that are reached by s and s' respectively via that transitions.

Example 3.4.5 The states s and s' in the action-labelled concurrent probabilistic system shown in Figure 3.5 on page 55 are bisimilar. A weight function for (μ, μ') with respect to $\sim$ can be obtained as follows. Clearly, $t \sim t'$ and v_1, $v_2 \sim v'$. Hence, t and t' can be combined as well as v_1, v_2 and v'.



Thus, $weight(t, t') = 1/2$, $weight(v_1, v') = 1/8$, $weight(v_2, v') = 3/8$ (and $weight(x, y) = 0$ in all other cases) yields a weight function for (μ, μ') with respect to $\sim$. ■

Remark 3.4.6 The "inference" from concurrent probabilistic systems to stratified systems sketched on page 40 can be extended for the action labelled case. For this, we associate with each action-labelled concurrent system $\mathcal{S} = (S, Act, Steps)$ the action-labelled stratified system $\mathcal{S}' = (S', Act, Steps')$ where

$$S' = S \cup \{(s, \nu) : s \in S, \nu \in Steps_a(s) \text{ for some } a \in Act\},$$
$$Steps'(s) = \{(a, \mu^1_{(s,\nu)}) : (a, \nu) \in Steps(s)\} \text{ and } Steps'(s, \nu) = \{(a_{random}, \nu)\}.$$

It is easy to see that this inference preserves bisimulation equivalence; i.e., if s, $s' \in S$ then s and s' are bisimilar as states of $\mathcal{S}$ iff s and s' are bisimilar as states of $\mathcal{S}'$. ■

The result of Milner [Miln89] that in every image-finite (non-probabilistic) labelled transition system bisimulation can be approximated by "finitary bisimulation" carries over to the probabilistic case.

Definition 3.4.7 [The relations $\sim_n$] *Let $(S, Act, Steps)$ be an action-labelled concurrent probabilistic system. We define inductively equivalence relations $\sim_n$ on S. We set $\sim_0 = S \times S$ and, for $n = 0, 1, 2, \ldots$, $s \sim_{n+1} s'$ iff:*

- *If $s \xrightarrow{a} \mu$ then there is a transition $s' \xrightarrow{a} \mu'$ with $\mu[C] = \mu'[C]$ for all $C \in S/\sim_n$.*
- *If $s' \xrightarrow{a} \mu'$ then there is a transition $s \xrightarrow{a} \mu$ with $\mu[C] = \mu'[C]$ for all $C \in S/\sim_n$.*

Lemma 3.4.8 *Let $(S, Act, Steps)$ be an image-finite action-labelled concurrent probabilistic system and s, $s' \in S$. Then, $s \sim s'$ iff $s \sim_n s'$ for all $n \geq 0$.*

Proof: see Section 3.7, Lemma 3.7.5 (page 68). ■

Lemma 3.4.8 can be adapted for the fully probabilistic case. In that case, no further assumptions (like image-finiteness) are needed. We state (without proof) that, whenever s, s' are states of an action-labelled fully probabilistic system then $s \sim s'$ iff $s \sim_n s'$ for all $n \geq 0$. Here, $s \sim_0 s'$ for all states s, s' and $s \sim_{n+1} s'$ iff $\mathbf{P}(s, a, C) = \mathbf{P}(s', a, C)$ for all $a \in Act$ and $C \in S/\sim_n$.

3.4.2 Simulation

Simulation can be viewed as "uni-directional bisimulation" in the sense that a process $\mathcal{P}'$ "simulates" another process $\mathcal{P}$ if each step of $\mathcal{P}$ can be "simulated" by a step of $\mathcal{P}'$. In that case, $\mathcal{P}$ can be viewed as an "implementation" of $\mathcal{P}'$ as each step of $\mathcal{P}$ is "allowed" by the "specification" $\mathcal{P}'$. The definition of a simulation for concurrent probabilistic systems with action labels by Segala & Lynch is based on that idea: the notion of a simulation is derived from the characterization of bisimulation in Proposition 3.4.4 (page 54) by dropping the symmetry (cf. Definition 3.4.9). At the end of this section, we show how this definition of a simulation can be modified for the fully probabilistic case. The resulting simulation preorder on action-labelled fully probabilistic systems can be viewed as an adaption of the "satisfaction relation" proposed by Jonsson & Larsen [JoLa91] that work with fully probabilistic systems and proposition labels.

Definition 3.4.9 [Simulation (concurrent case), cf. [SeLy94]] *Let $(S, Act, Steps)$ be an action-labelled concurrent probabilistic system. A* simulation *for $(S, Act, Steps)$ is a subset R of $S \times S$ such that for all $(s, s') \in R$:*

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

We say s implements *s' and s'* simulates *s (denoted by $s \sqsubseteq_{\text{sim}} s'$) iff there exists a simulation which contains (s, s'). s, s' are called* similar *(denoted by $s_1 \sim_{\text{sim}} s_2$) iff $s \sqsubseteq_{\text{sim}} s'$ and $s' \sqsubseteq_{\text{sim}} s$.*

In the non-probabilistic case, the above notion of a simulation agrees with Milner's notion of a simulation [Miln89]. Note that in the non-probabilistic case, $s \sqsubseteq_{\text{sim}} s'$ iff the function *weight* with $\mathit{weight}(u, u') = 0$ if $(u, u') \neq (s, s')$ and $\mathit{weight}(s, s') = 1$ is a weight function for $(\mu_s^1, \mu_{s'}^1)$ with respect to $\sqsubseteq_{\text{sim}}$. Hence, if $(S, Act, \longrightarrow)$ is a non-probabilistic labelled transition system (i.e. S is a set of states and $\longrightarrow$ a subset of $S \times Act \times S$) and $R \subseteq S \times S$ then R is a simulation in the sense of Definition 3.4.9 (i.e. R is a simulation for the induced probabilistic transition system $(S, Act, Steps)$ where $Steps(s) = \{(a, \mu_t^1) : s \xrightarrow{a} t\}$) if and only if R is a simulation in the sense of Milner.

Example 3.4.10 Consider the transition system of Figure 3.6 (page 57). Clearly, $u \sqsubseteq_{\mathrm{sim}}$

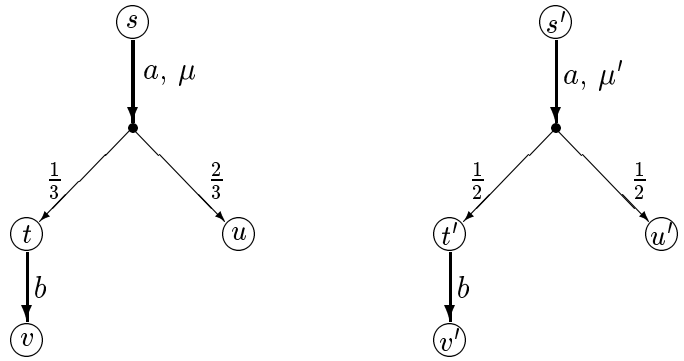
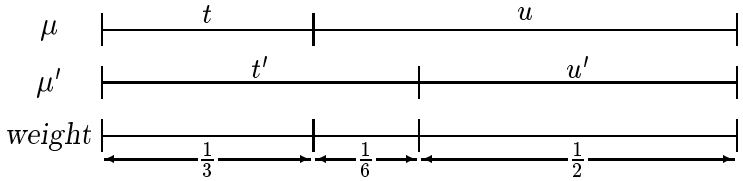


Figure 3.6: $s \sqsubseteq_{\mathrm{sim}} s'$

u' and $u, t \sqsubseteq_{\mathrm{sim}} t'$. A weight function for (μ, μ') with respect to $\sqsubseteq_{\mathrm{sim}}$ can be obtained by combining certain parts of t (of u) with certain parts of t' (of u' and t'). The weight function *weight* for (μ, μ') with respect to $\sqsubseteq_{\mathrm{sim}}$ is given by: $\mathit{weight}(t, t') = 1/3$, $\mathit{weight}(u, t') = 1/6$, $\mathit{weight}(u, u') = 1/2$.



We obtain $s \sqsubseteq_{\mathrm{sim}} s'$. ■

Remark 3.4.11 [**Alternative simulation-like preorders**] There are simpler possibilities to drop the symmetry from the definition of bisimulation equivalence thus yielding alternative definitions of a simulation preorder that do not use weight functions. One possibility is to consider the downward closure $t \downarrow_R$ of all elements $t \in S$ and to (re-)define the relation $\preceq_R$ on $\mathit{Distr}(S)$ by:

$$\mu \preceq'_R \mu' \text{ iff } \mu[t \downarrow_R] \geq \mu'[t \downarrow_R] \text{ for all } t \in S.$$

Another possibility is to deal with the upward closures $t \uparrow_R$. Both possibilities yield a preorder that is strictly coarser than the simulation preorder à la [SeLy94]. We argue that none of these relations can be viewed as a probabilistic counterpart to Milner's simulation preorder.

We define a $\downarrow$-*simulation* to be a binary relation R on S such that for all $(s, s') \in R$ and $s \xrightarrow{a} \mu$ there exists $s' \xrightarrow{a} \mu'$ with $\mu[t \downarrow_R] \geq \mu'[t \downarrow_R]$ for all $t \in S$. Here, $t \downarrow_R = \{u \in S : (u, t) \in R\}$. Similarly, a $\uparrow$-*simulation* is a binary relation R on S such that for all $(s, s') \in R$ and $s \xrightarrow{a} \mu$ there exists $s' \xrightarrow{a} \mu'$ with $\mu[t \uparrow_R] \geq \mu'[t \uparrow_R]$ for all $t \in S$ where $t \uparrow_R = \{u \in S : (t, u) \in R\}$. We define $s \sqsubseteq_{\downarrow} s'$ ($s \sqsubseteq_{\uparrow} s'$) iff $(s, s') \in R$ for some $\downarrow$-simulation ($\uparrow$-simulation). Using the results of Chapter 5 (Lemma 5.3.11 on page 113) we obtain that, for $(S, \mathit{Act}, \mathit{Steps})$ to be finite, the simulation preorder $\sqsubseteq_{\mathrm{sim}}$ is a $\downarrow$-simulation and a $\uparrow$-simulation. Thus,

$$s \sqsubseteq_{\mathrm{sim}} s' \text{ implies } s \sqsubseteq_{\downarrow} s' \text{ and } s \sqsubseteq_{\uparrow} s'.$$

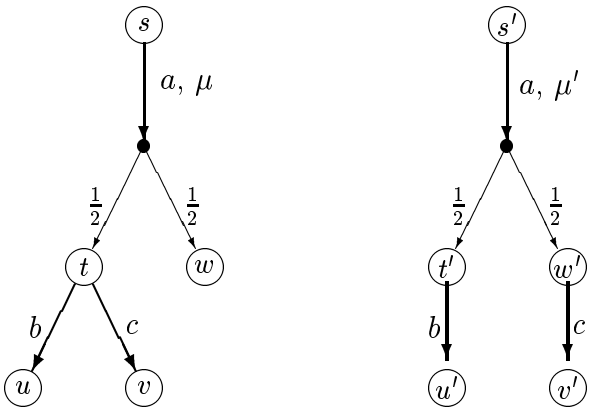


Figure 3.7: $s \sqsubseteq_{\downarrow} s'$ and $s \not\sqsubseteq_{\text{sim}} s'$

In the system of Figure 3.7 (page 58) we have $s \sqsubseteq_{\downarrow} s'$ but $s \not\sqsubseteq_{\text{sim}} s'$. From the branching time view, the process on left (i.e. the process with initial state s) should not be considered to be an implemention of the process on the right (i.e. the process with initial state s') as s can reach a state where both actions c and b can be performed while s' cannot. In the system of Figure 3.8 (page 58) we have $s \sqsubseteq_{\uparrow} s'$ but $s \not\sqsubseteq_{\text{sim}} s'$. In our opinion, the process on the right (the process with initial state s') cannot be viewed as a simulation of the process on the left (the process with initial state s) since s reachs a non-terminal state after performing a with probability 1, while s' can reach a terminal state (w') after

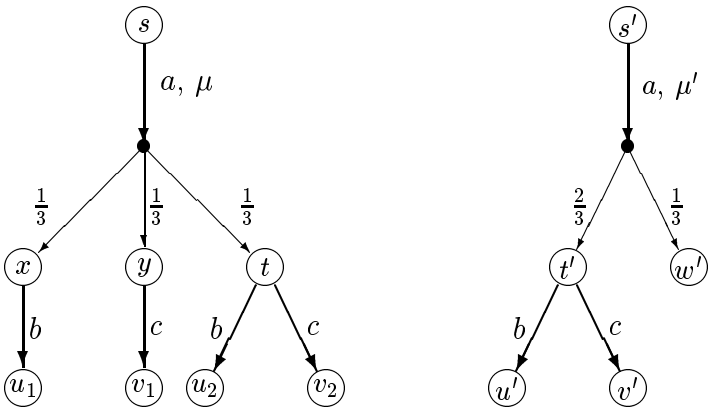


Figure 3.8: $s \sqsubseteq_{\uparrow} s'$ and $s \not\sqsubseteq_{\text{sim}} s'$

performing a with non-zero probability.[19] ■

Definition 3.4.12 [The relations $\sqsubseteq_n$] *Let $(S, Act, Steps)$ be an action-labelled concurrent probabilistic system. By induction on n we define relations $\sqsubseteq_n \subseteq S \times S$:*

- $s \sqsubseteq_0 s'$ *for all states s, $s' \in S$*
- $s \sqsubseteq_{n+1} s'$ *iff whenever $s \xrightarrow{a} \mu$ then there exists a transition $s' \xrightarrow{a} \mu'$ and a weight function weight for (μ, μ') with respect to $\sqsubseteq_n$ (i.e. $\mu \preceq_{\sqsubseteq_n} \mu'$).*

Similarly to Lemma 3.4.8 (page 56) we obtain the following.

[19] Even the relation $\sqsubseteq_{\uparrow} \cap \sqsubseteq_{\downarrow}$ is coarser than $\sqsubseteq_{\text{sim}}$. In the system of Figure 3.8, we add a transition $w' \xrightarrow{a} \mu^1_{w'}$ and obtain $s \not\sqsubseteq_{\text{sim}} s'$ while $(s \sqsubseteq_{\uparrow} s') \wedge (s \sqsubseteq_{\downarrow} s')$.

Lemma 3.4.13 *Let $(S, Act, Steps)$ be an image-finite action-labelled concurrent probabilistic system and s, $s' \in S$. Then, $s \sqsubseteq_{\mathrm{sim}} s'$ iff $s \sqsubseteq_n s'$ for all $n \geq 0$.*

Proof: see Section 3.7, Lemma 3.7.6 (page 68). ■

The following lemma shows that $\sqsubseteq_{\mathrm{sim}}$ is a preorder and its kernel $\sim_{\mathrm{sim}}$ is coarser than bisimulation equivalence.

Lemma 3.4.14 *Let $(S, Act, Steps)$ be an action-labelled concurrent probabilistic system and $s, s', s'', s_1, s_1', s_2, s_2' \in S$. Then:*

(a) $s \sim s' \implies s \sim_{\mathrm{sim}} s'$

(b) $s \sqsubseteq_{\mathrm{sim}} s',\ s' \sqsubseteq_{\mathrm{sim}} s'' \implies s \sqsubseteq_{\mathrm{sim}} s''$

(c) $s_1 \sqsubseteq_{\mathrm{sim}} s_2,\ s_1 \sim s_1',\ s_2 \sim s_2' \implies s_1' \sqsubseteq_{\mathrm{sim}} s_2'$

Proof: (a) follows immediately by the definition of a simulation and Proposition 3.4.4 (page 54). (c) follows by (a) and (b). The transitivity of $\sqsubseteq_{\mathrm{sim}}$ (item (b)) can be derived from Remark 2.2.1 (page 30). ■

By part (a) of Lemma 3.4.14, bisimulation equivalence $\sim$ is finer than simulation equivalence $\sim_{\mathrm{sim}}$. As in the non-probabilistic case, simulation equivalence $\sim_{\mathrm{sim}}$ does not coincide with bisimulation equivalence $\sim$. For instance, for the non-probabilistic system of Figure 3.9 (page 59) we have $s \sim_{\mathrm{sim}} s'$ but $s \not\sim s'$. In the case of reactive systems, simulation and bisimulation equivalence coincide. This result can be viewed as the probabilistic counter-

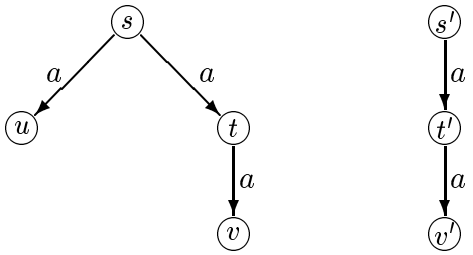


Figure 3.9: $s \sim_{\mathrm{sim}} s'$ but $s \not\sim s'$

part to the well-known result that simulation equivalence and bisimulation equivalence are the same for deterministic (non-probabilistic) transition systems.[20]

Theorem 3.4.15 *Let $(S, Act, Steps)$ be a reactive action-labelled concurrent probabilistic system and s, $s' \in S$. Then, $s \sim_{\mathrm{sim}} s'$ iff $s \sim s'$.*

Proof: see Section 5.3.1, Thoerem 5.3.6 (page 110). ■

We adapt the definition of the simulation preorder à la Segala & Lynch [SeLy94] (Definition 3.4.9, page 56) for action-labelled fully probabilistic systems. The resulting definition of the simulation preorder on fully probabilistic systems with action labels (Definition 3.4.17, page 60) can be viewed as the action-labelled counterpart to the definition of the "satisfaction relation" for fully probabilistic systems with proposition labels by Jonsson & Larsen (cf. Definition 4.3 in [JoLa91]). In the sequel, $\mathcal{S} = (S, Act, \mathbf{P})$ denotes an action-labelled fully probabilistic system.

[20] Recall that a non-probabilistic action-labelled transition system is called deterministic iff, for each state s and action a, there is at most one transition $s \xrightarrow{a} t$.

Definition 3.4.16 [Weight functions in fully probabilistic systems] *Let s, $s' \in S$ and $R \subseteq S \times S$. If s is non-terminal then a* weight function *for (s, s') with respect to R is a function $weight : S \times Act \times S \to [0, 1]$ such that for all $a \in Act$ and t, $t' \in S$:*

1. *If $weight(t, a, t') > 0$ then $(t, t') \in R$.*
2. $$\sum_{u' \in S} weight(t, a, u') \;=\; \mathbf{P}(s, a, t), \qquad \sum_{u \in S} weight(u, a, t') \;=\; \mathbf{P}(s', a, t').$$

We write $s \sqsubseteq_R s'$ iff either s is terminal or there exists a weight function for (s, s') with respect to R.

In particular, if s is non-terminal and $s \sqsubseteq_R s'$ then s' is non-terminal. Suppose s and s' to be non-terminal and let μ and μ' be the induced distributions on $Act \times S$, i.e. $\mu(a, t) = \mathbf{P}(s, a, t)$ and $\mu'(a, t') = \mathbf{P}(s', a, t')$. If $s \sqsubseteq_R s'$ then the weight function *weight* for (s, s') induces a weight function $weight' : (Act \times S) \times (Act \times S) \to [0, 1]$,

$$weight'(\langle a, t\rangle, \langle a, t'\rangle) \;=\; weight(t, a, t')$$

for (μ, μ') with $R' = \{(\langle a, t\rangle, \langle a, t'\rangle : a \in Act, (t, t') \in R\}$. Vice versa, if $\mu \preceq_{R'} \mu'$ (where R' is as before) then $s \sqsubseteq_R s'$.

Definition 3.4.17 [Simulation (fully probabilistic case)] *A* simulation *for $\mathcal{S}$ is a binary relation R on S such that $s \sqsubseteq_R s'$ for all $(s, s') \in R$. We say s* implements *s' and s'* simulates *s (denoted by $s \sqsubseteq_{\text{sim}} s'$) iff there exists a simulation that contains (s, s').*

Example 3.4.18 Consider the action-labelled fully probabilistic system of Figure 3.10 on page 60. The relation $R = \{ (s, s'), (t, t'), (u, u'), (v, u'), (w, w') \}$ is a simulation as e.g. $weight(t, a, t') = weight(v, b, u') = weight(u, b, u') = 1/3$ (and $weight(\cdot) = 0$ in all

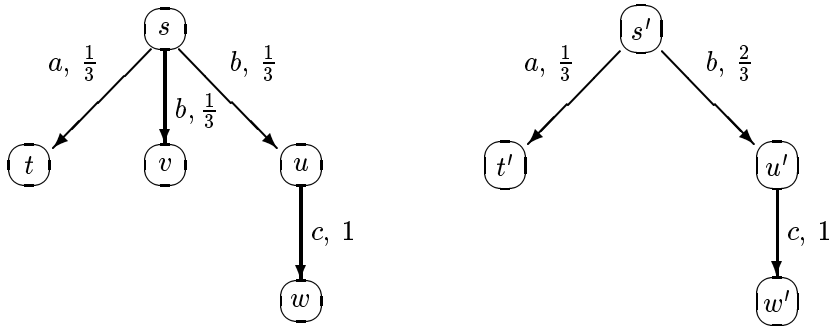


Figure 3.10: $s \sqsubseteq_{\text{sim}} s'$

other cases) is a weight function for (s, s') with respect to R. Hence, $s \sqsubseteq_{\text{sim}} s'$. ■

As in Lemma 3.4.14 (page 59) it can be shown that $\sqsubseteq_{\text{sim}}$ is transitive; as in Lemma 3.4.13 (page 59), it can be shown that $\sqsubseteq$ can be approximated by the "finitary" relation $\sqsubseteq_n$. More precisely, if s, s' are states of an action-labelled fully probabilistic system then $s \sqsubseteq_{\text{sim}} s'$ iff $s \sqsubseteq_n s'$ for all $n \geq 0$.[21] Similarly to Theorem 3.4.15 (page 59) we obtain that,

[21] Here, the relations $\sqsubseteq_n$ are defined as follows. Let $R_0 = S \times S$ and $R_{n+1} = \sqsubseteq_{R_n}$ (where $\sqsubseteq_{R_n}$ is defined as in Definition 3.4.16, page 60). Then, $s \sqsubseteq_n s'$ iff $(s, s') \in R_n$.

for action-labelled fully probabilistic systems, bisimulation equivalence and simulation equivalence are the same. This result can be viewed as the action-labelled counterpart to Theorem 4.6 of [JoLa91] which considers fully probabilistic systems with proposition labels and characterizes bisimulation equivalence for them in terms of weight functions.

Theorem 3.4.19 (cf. Theorem 4.6 in [JoLa91]) *Let $(S, Act, \mathbf{P})$ be an action-labelled fully probabilistic system and s, $s' \in S$. Then,*

$$s \sim s' \quad \textit{iff} \quad s \sim_{\mathrm{sim}} s'.$$

Proof: see Section 5.3, Theorem 5.3.7 (page 110). ■

3.5 Probabilistic processes

The behaviour of a probabilistic process can be described by "pointed probabilistic systems", i.e. a probabilistic system together with a specified state, the *initial state*. Alternatively, we could deal with a distribution for the initial states in the fully probabilistic case and a set of initial states in the concurrent case (as done e.g. in [CoYa95, JoYi95, Sega95a]).

Definition 3.5.1 [Probabilistic processes] *A* probabilistic process *is a tuple $\mathcal{P} = (\mathcal{S}, s_{init})$ consisting of a (fully or concurrent) probabilistic system $\mathcal{S}$ with state space S and an* initial state *$s_{init} \in S$.*

For instance, a fully probabilistic process is a tuple $\mathcal{P} = (S, \mathbf{P}, s_{init})$ consisting of a fully probabilistic system $(S, \mathbf{P})$ and an initial state $s_{init} \in S$. Probabilistic processes are extended by action or proposition labels in the obvious way. E.g. an action-labelled concurrent probabilistic process is a tuple $(S, Act, Steps, s_{init})$ consisting of an action-labelled concurrent probabilistic transition system $(S, Act, Steps)$ and an initial state $s_{init} \in S$. Bisimulation equivalence and the simulation preorder are adapted for probabilistic processes as follows. We consider the probabilistic system that arises by taking the disjoint union of the two underlying probabilistic systems and compare the initial states in the composed system. For instance, let $\mathcal{P} = (\mathcal{S}, s_{init})$ and $\mathcal{P}' = (\mathcal{S}', s'_{init})$ be two action-labelled concurrent probabilistic processes where $\mathcal{S} = (S, Act, Steps)$, $\mathcal{S}' = (S', Act', Steps')$ are the underlying systems. Then, $\mathcal{P}$ and $\mathcal{P}'$ are said to be bisimilar (written $\mathcal{P} \sim \mathcal{P}'$) iff s_{init} and s'_{init} are bisimilar as states of the composed system $\mathcal{S} \uplus \mathcal{S}' = (S \uplus S', Act \cup Act', \overline{Steps})$ where $\overline{Steps}(s) = Steps(s)$ if $s \in S$ and $\overline{Steps}(s) = Steps'(s)$ if $s \in S'$.[22] Similarly, we define bisimulation equivalence (also denoted $\sim$) for action-labelled fully probabilistic processes, the simulation preorder $\sqsubseteq_{\mathrm{sim}}$, simulation equivalence $\sim_{\mathrm{sim}}$ and the relations $\sim_n$ and $\sqsubseteq_n$ for action-labelled probabilistic processes. Here, in the fully probabilistic case, the composed system $\mathcal{S} \uplus \mathcal{S}'$ is defined as follows. If $\mathcal{S} = (S, Act, \mathbf{P})$ and $\mathcal{S}' = (S', Act', \mathbf{P}')$ then $\mathcal{S} \uplus \mathcal{S} = (S \uplus S', Act \cup Act', \overline{\mathbf{P}})$ where $\overline{\mathbf{P}}(s, a, t) = \mathbf{P}(s, a, t)$ if $(s, a, t) \in S \times Act \times S$, $\overline{\mathbf{P}}(s, a, t) = \mathbf{P}'(s, a, t)$ if $(s, a, t) \in S' \times Act' \times S'$ and $\overline{\mathbf{P}}(s, a, t) = 0$ in all other cases. Clearly, the results of Section 3.3.1 carry over to the pointed case. That is,

[22] Here, each distribution μ on S is identified with the distribution $\overline{\mu} : S \uplus S' \to [0, 1]$, $\overline{\mu}(t) = \mu(t)$ if $t \in S$ and $\overline{\mu}(t) = 0$ if $t' \in S'$. In the same way, each distribution μ' on S' is viewed as a distribution on $S \uplus S'$.

in the fully and concurrent case, $\sqsubseteq_{\mathrm{sim}}$ is a preorder on the collection of all action-labelled (fully or concurrent) probabilistic processes.

- In the fully probabilistic case: $\mathcal{P} \sim \mathcal{P}'$ iff $\mathcal{P} \sim_{\mathrm{sim}} \mathcal{P}'$.
- In the concurrent case: $\sim$ is strictly finer than $\sim_{\mathrm{sim}}$. And:
  (a) If $\mathcal{P}$ and $\mathcal{P}'$ are image-finite then
    (i) $\mathcal{P} \sim \mathcal{P}'$ iff $\mathcal{P} \sim_n \mathcal{P}'$ for all $n \geq 0$
    (ii) $\mathcal{P} \sqsubseteq_{\mathrm{sim}} \mathcal{P}'$ iff $\mathcal{P} \sqsubseteq_n \mathcal{P}'$ for all $n \geq 0$.
  (b) If $\mathcal{P}$ and $\mathcal{P}'$ are reactive then $\mathcal{P} \sim \mathcal{P}'$ iff $\mathcal{P} \sim_{\mathrm{sim}} \mathcal{P}'$.

3.6 Related models

We briefly explain the relation of our models to those proposed in the literature. First we observe that in contrast to some authors we require that, in the fully probabilistic case, the probabilities of the outgoing transitions of a non-terminal state s sum up to one. In those models that allow for *substochastic* states (i.e. states s where $\sum_t \mathbf{P}(s,t) \in]0,1[$ or $\sum_{a,t} \mathbf{P}(s,a,t) \in]0,1[$ for action-labelled systems) the value

$$\Delta(s) \;=\; 1 \;-\; \sum_t \mathbf{P}(s,t)$$

(or, when dealing with action labels, $\Delta(s) = 1 - \sum_{a,t} \mathbf{P}(s,a,t)$) can be viewed as the probability for the system to "halt" in s. Depending on the system under consideration, this "halting" might be interpreted as well-termination, deadlock or divergence. In our fully probabilistic model (which does not allow for substochastic states), we assume that "halting" is described by special state transitions. For instance, in absence of action labels, one can use an auxiliary terminal state $\mathbf{0}$ and the probabilistic state transitions $\mathbf{P}(s,\mathbf{0}) = \Delta(s)$ to formalize "halting". Using action labels, one might deal with such an auxiliary state $\mathbf{0}$ and a special action symbol, e.g. 0, and the transition probabilities $\mathbf{P}(s,0,\mathbf{0}) = \Delta(s)$.

We now briefly sketch how our models are related to the models considered in the literature where we ignore the differences arising from the use of sets of initial states or distributions for the initial states (rather than dealing with a single initial state as we do) and/or allowing substochastic states.

In the notations of [vGSST90], fully probabilistic processes with action labels are called *generative* processes.[23] They also agree with the *fully probabilistic automaton* of Segala [Sega95a]. In the concurrent case, our action-labelled processes coincide with the *simple probabilistic automaton* of Segala & Lynch [SeLy94, Sega95a].

The *alternating model* of Hansson & Jonsson [HaJo90, Hans91] can be obtained from action-labelled stratified processes (in the sense of Definition 3.3.12 on page 52) by removing the action symbol a_{random} from the steps of the probabilistic states. The stratified model of [vGSST90] essentially agrees with the alternating model. The main difference between stratified systems à la [vGSST90] and alternating systems à la [HaJo90, Hans91] are

[23] It should be noticed that this just holds for the formal definition of the model. The interpretation of generative systems in the approach of [vGSST90] is slightly different from ours since we assume internal probabilistic choice while [vGSST90] deals with external probabilistic choice.

that [vGSST90] assume an external probabilistic choice while [HaJo90, Hans91] deal with internal probabilistic choice and that, in the approach of [vGSST90], the non-probabilistic states cannot behave non-deterministically.[24] Ignoring the different interpretation of the probabilistic choice operator and allowing non-determinism in the non-probabilistic states of a stratified system in the sense of [vGSST90] we obtain the alternating model; and hence, by adding the action symbol a_{random}, action-labelled stratified systems in our sense.

A proposition-labelled fully probabilistic process is a *sequential Markov chain* in the sense of Vardi [Vard85] (or Courcoubetis & Yannakakis [CoYa88, CoYa95]) while proposition-labelled concurrent probabilistic processes agree with *probabilistic non-deterministic systems* à la Bianco & de Alfaro [BidAl95, dAlf97a, dAlf97b]. In essence, the *concurrent Markov chains* of [Vard85, VaWo86, CoYa88, CoYa95] are the same as stratified proposition-labelled systems.

The model considered by Pnueli & Zuck [Pnue83, PnZu86a, PnZu86b, PnZu93] (just called *probabilistic programs*) can be viewed as a generalization of concurrent probabilistic systems in our sense. In the approach of Pnueli & Zuck, each probabilistic program is assigned a set of "commands" (called "transitions" in the approach of Pnueli & Zuck), where each command *comm* is associated with an enabling predicate (represented by a subset $Enabled(comm)$ of the state space S) and a set $Modes(comm) = \{mode^1, \ldots, mode^k\}$ of "modes". Each mode $mode^i$ is associated with a non-zero probability and a *set* of possible next successor states. If we assume the sets of the modes to be singletons (that prescribe unique successor states), each command *comm* corresponds to a distribution μ_{comm} on the state space. In that case, the probabilistic programs à la Pnueli & Zuck specializes to concurrent probabilistic systems in our sense where $Steps(s)$ is given by the set of commands that are enabled in state s, i.e. $Steps(s) = \{\mu_{comm} : s \in Enabled(comm)\}$. If we assume that in addition the commands are associated with an action label then the model of Pnueli & Zuck can also be viewed as a generalization of our concurrent probabilistic systems with action labels.

Remark 3.6.1 Van Glabbeek et al [vGSST90] present a hierarchy for the several action-labelled systems (reactive, generative, stratified) together with the corresponding type of bisimulation equivalence. We briefly sketch how the inferences between the models can be reformulated for our notations.

Given a generative (i.e. action-labelled fully probabilistic) system $\mathcal{S}_G = (S, Act, \mathbf{P}_G)$, we may abstract from the probabilities for chosing a certain action and deal with the reactive transition probabilities

$$\mathbf{P}_R(s,a,t) \;=\; \frac{\mathbf{P}_G(s,a,t)}{\mathbf{P}_G(s,a)} \;\; \text{(provided that } \mathbf{P}_G(s,a) > 0).$$

In the case where $\mathbf{P}_G(s,a) = 0$ we put $\mathbf{P}_R(s,a,t) = 0$ for all states t. As shown in [vGSST90], this inference from generative systems to reactive systems preserves bisimulation equivalence.[25] We now compare the generative and stratified view. Let $\mathcal{P}_S =$

[24] Note that [vGSST90] considers a process calculus with synchronous parallelism and probabilistic choice rather than non-determinism. Thus, the systems in [vGSST90] do not behave non-deterministically.

[25] Formally, if $\mathcal{P}_G$ and $\mathcal{Q}_G$ are generative processes and $\mathcal{P}_R$, $\mathcal{Q}_R$ the associated reactive processes then $\mathcal{P}_G \sim \mathcal{Q}_G$ implies $\mathcal{P}_R \sim \mathcal{Q}_R$ while the converse does not hold.

$(S, Act, Steps, s_{init})$ be an action-labelled stratified process where none of the non-probabilistic states of $\mathcal{P}_S$ behaves non-deterministically (i.e. if s is non-probabilistic then $s \xrightarrow{a} t$ for at most one action a and state t) as it is the case for the systems in the approach of [vGSST90]. $\mathcal{P}_S$ can identified with the generative process $\mathcal{P}_G = (S, Act, \mathbf{P}_G, s_{init})$ where

$$\mathbf{P}_G(s,a,t) \;=\; \begin{cases} \mu(t) & : \text{ if } s \xrightarrow{a} \mu \text{ and } a = a_{random} \\ 1 & : \text{ if } s \xrightarrow{a} t \text{ and } a \neq a_{random} \\ 0 & : \text{ otherwise.} \end{cases}$$

Given two such action-labelled stratified processes $\mathcal{P}_S$ and $\mathcal{Q}_S$ where none of the non-probabilistic states behaves non-deterministically, we have

(*) $\mathcal{P}_S \sim \mathcal{Q}_S$ iff $\mathcal{P}_G \sim \mathcal{Q}_G$

where $\mathcal{P}_G$ and $\mathcal{Q}_G$ denote the associated generative processes.[26] This should be contrasted with the abstraction result of [vGSST90] where a different inference is used. In the approach of [vGSST90], the "if"-part of (*) does not hold. The inference from the stratified to the generative model used in [vGSST90] (adapted for our type of action-labelled stratified systems) removes the transitions labelled by a_{random} and deals with the cumulative effect of all probabilistic choices. Formally, if $\mathcal{S}_S = (S, Act, Steps)$ is an action-labelled stratified system as above (i.e. $|Steps(s)| \leq 1$ for all non-probabilistic states) then, in the approach of [vGSST90], the associated generative system is $(S, Act_G, \mathbf{P}'_G)$ where $Act_G = Act \setminus \{a_{random}\}$ and $\mathbf{P}'_G : S \times Act_G \times (S \setminus S_{prob}) \to [0,1]$ the least function such that

$$\mathbf{P}'_G(s,a,t) \;=\; \begin{cases} \sum_{u \in S} \; \mu(u) \cdot \mathbf{P}'_G(u,a,t) & : \text{ if } s \xrightarrow{a} \mu \text{ and } a = a_{random} \\ 1 & : \text{ if } s \xrightarrow{a} t \text{ and } a \neq a_{random} \\ 0 & : \text{ otherwise.} \end{cases}$$

Here, S_{prob} is the set of probabilistic states in $\mathcal{S}_S$.[27] ■

3.7 Proofs

3.7.1 Probabilistic reachability analysis

We give the proof for Theorem 3.1.6 (page 36) and Theorem 3.2.11 (page 43) that characterizes the probabilities to reach certain states as least fixed points.

In fully probabilistic systems, we use the following notation. If $\sigma \in Path_{fin}$, $|\sigma| = i$ and $\mathbf{P}(last(\sigma), t) > 0$ then $\sigma \to t$ denotes the unique path $\gamma \in Path_{fin}$ with $\gamma^{(i)} = \sigma$, $|\gamma| = i+1$

[26] The underlying notion of bisimulation equivalence for $\mathcal{P}_S$ and $\mathcal{Q}_S$ is that of Definition 3.4.2 (page 54) while for $\mathcal{P}_G$ and $\mathcal{Q}_G$ we deal with bisimulation in the sense of Definition 3.4.1 (page 54).

[27] Both transformations from the stratified to the generative model fail when non-determinism is allowed in the non-probabilistic states, because, for those states s in which non-determinism is present, the transition probabilities $\mathbf{P}_G(s,\cdot)$ or $\mathbf{P}'_G(s,\cdot)$ do not sum up to 1. Given an action-labelled stratified system $\mathcal{S}_S$ (where non-determinism is present in some non-probabilistic states), the generative (fully probabilistic) system associated by an adversary can be viewed as a refinement of $\mathcal{S}$. Here, the underlying refinement step just resolves the non-deterministic choices.

and $last(\gamma) = t$. Similarly, in concurrent probabilistic systems, we write $\sigma \xrightarrow{\mu} t$ to denote the unique path $\gamma \in Path_{fin}$ with $\gamma^{(i)} = \sigma$, $|\gamma| = i + 1$, $step(\gamma, i) = \mu$ and $last(\gamma) = t$. (Here, we assume that $\sigma \in Path_{fin}$, $|\sigma| = i$, $\mu \in Steps(last(\sigma))$ and $t \in Supp(\mu)$.)

Proposition 3.7.1 *Let $(S, \mathbf{P})$ be a fully probabilistic system, $\Sigma \subseteq Path_{fin}$. For $\sigma \in \Sigma$, let $\Sigma(\sigma) = \{\sigma' \in Path_{fin}(last(\sigma)) : \sigma \circ \sigma' \in \Sigma\}$ and $p : Path_{fin} \to [0,1]$, $p(\sigma) = Prob(\Sigma(\sigma) \uparrow)$. Then, p is the least fixed point of the operator $F : (Path_{fin} \to [0,1]) \to (Path_{fin} \to [0,1])$ which is given by $F(f)(\sigma) = 1$ if $\sigma \in \Sigma$ and*

$$F(f)(\sigma) \;=\; \sum_{t \in Next(\sigma)} \mathbf{P}(last(\sigma), t) \cdot f(\sigma \to t)$$

if $\sigma \notin \Sigma$. Here, $Next(\sigma) = \{t \in S : \mathbf{P}(last(\sigma), t) > 0\}$.

Proof: Clearly, F is monotone and preserves infinima and suprema. Let f be the least fixed point of F.[28] If $\sigma \in \Sigma$ then the path consisting of the state $last(\sigma)$ belongs to $\Sigma(\sigma)$. Thus, $\Sigma(\sigma) \uparrow = Path_{fin}(last(\sigma))$ and $p(\sigma) = f(\sigma) = 1$. Next we assume that $\sigma \notin \Sigma$. For $t \in Next(\sigma)$, let $\Sigma^t(\sigma)$ be the set of finite paths $last(\sigma) \to \sigma'$ where $\sigma' \in \Sigma(\sigma \to t)$. Then, $\Sigma(\sigma)$ can be written as disjoint union of the sets $\Sigma^t(\sigma)$, $t \in Next(\sigma)$. As $Prob(\Sigma^t(\sigma) \uparrow) = \mathbf{P}(last(\sigma), t) \cdot p(\sigma \to t)$ we obtain:

$$p(\sigma) \;=\; \sum_{t \in Next(\sigma)} Prob(\Sigma^t(\sigma) \uparrow) \;=\; \sum_{t \in Next(\sigma)} \mathbf{P}(last(\sigma), t) \cdot p(\sigma \to t) \;=\; F(p)(\sigma).$$

Thus, p is a fixed point of F. We conclude $f(\sigma) \leq p(\sigma)$ for all $\sigma \in Path_{fin}$.

For $k = 0, 1, 2, \ldots$, let $\Sigma_k(\sigma) = \{\sigma' \in \Sigma(\sigma) : |\sigma'| \leq k\}$ and $p_k(\sigma) = Prob(\Sigma_k(\sigma) \uparrow)$. Then, $\Sigma_0(\sigma) \subseteq \Sigma_1(\sigma) \subseteq \Sigma_2(\sigma) \subseteq \ldots$ and $\Sigma(\sigma) = \bigcup \Sigma_k(\sigma)$. Thus, $p(\sigma) = \lim p_k(\sigma)$. It is easy to see that $p_{k+1} = F(p_k)$. By induction on k we get $p_k(\sigma) \leq f(\sigma)$ for all $\sigma \in Path_{fin}$. Hence, $p(\sigma) \leq f(\sigma)$. We conclude that $p = f$ is the least fixed point of F. ■

Corollary 3.7.2 (cf. Theorem 3.1.6, page 36) *Let $(S, \mathbf{P})$ be a fully probabilistic system. Let S_1, S_2 be subsets of S. Let $\Sigma \subseteq Path_{fin}$ be the set of all finite paths σ such that $\sigma(i) \in S_1 \setminus S_2$, $i = 0, 1, \ldots, |\sigma| - 1$, $last(\sigma) \in S_2$. Let $\Pi = \Sigma \uparrow$. Then,*

$$p : S \to [0,1],\ p(s) = Prob\,(\Pi(s)),$$

is the least fixed point of the operator $F : (S \to [0,1]) \to (S \to [0,1])$ which is given by $F(f)(s) = 1$ if $s \in S_2$, $F(f)(s) = 0$ if $s \in S \setminus (S_1 \cup S_2)$ and, if $s \in S_1 \setminus S_2$,

$$F(f)(s) \;=\; \sum_{t \in S} \mathbf{P}(s, t) \cdot f(t).$$

Proof: follows immediately by Proposition 3.7.1 (page 65). Uses the fact that $\Sigma(\sigma) = \Sigma(last(\sigma))$ for each $\sigma \in Path_{fin}$. ■

Proposition 3.7.3 *Let $(S, Steps)$ be a concurrent probabilistic system and $\Sigma \subseteq Path_{fin}$. For $\sigma \in \Sigma$ and $A \in \mathcal{A}dv$, let $\Sigma^A(\sigma) = \{\sigma' \in Path_{fin}(last(\sigma)) : \sigma \circ \sigma' \in \Sigma^A\}$ and*

$$p^{min}(\sigma) \;=\; \inf_{A \in \mathcal{A}dv} Prob\left(\Sigma^A(\sigma) \uparrow^A\right), \quad p^{max}(\sigma) \;=\; \sup_{A \in \mathcal{A}dv} Prob\left(\Sigma^A(\sigma) \uparrow^A\right).$$

Then, p^{min} and p^{max} are the least fixed points of the operators

[28] The existence of a least fixed point can be shown using standard arguments of domain theory. See e.g. Proposition 12.1.1 (page 309).

$$F^{min},\ F^{max} : (Path_{fin} \to [0,1]) \to (Path_{fin} \to [0,1])$$

that are defined as follows. If $\sigma \in \Sigma$ *then* $F(f)(\sigma) = 1$. *If* $\sigma \notin \Sigma$ *then*

$$F^{min}(f)(\sigma) \;=\; \min \left\{ \sum_{t \in Supp(\mu)} \mu(t) \cdot f\left(\sigma \xrightarrow{\mu} t\right) \;:\; \mu \in Steps(last(\sigma)) \right\},$$

$$F^{max}(f)(\sigma) \;=\; \max \left\{ \sum_{t \in Supp(\mu)} \mu(t) \cdot f\left(\sigma \xrightarrow{\mu} t\right) \;:\; \mu \in Steps(last(\sigma)) \right\}.$$

Proof: Clearly, the operators F^* are monotone and preserve suprema. Let f^* be the least fixed point of F^*.[29] For A to be an adversary, let $p^A : Path_{fin} \to [0,1]$ be given by

$$p^A(\sigma) = Prob\left(\Sigma^A(\sigma) \uparrow^A\right).$$

Claim 1: $p^{min} = f^{min}$

Proof: We define $F^A : (Path_{fin} \to [0,1]) \to (Path_{fin} \to [0,1])$ by

$$F^A(f)(\sigma) \;=\; \sum_{t \in Supp(A(\sigma))} A(\sigma)(t) \cdot f\left(\sigma \xrightarrow{A(\sigma)} t\right).$$

By Proposition 3.7.1 (page 65), $p^A = lfp(F^A)$. For each $\sigma \in Path_{fin}$, we choose some $\mu_\sigma \in Steps(last(\sigma))$ such that

$$f^{min}(\sigma) \;=\; \sum_{t \in Supp(\mu_\sigma)} \mu_\sigma(t) \cdot f^{min}\left(\sigma \xrightarrow{\mu_\sigma} t\right).$$

Let A be the adversary with $A(\sigma) = \mu_\sigma$. Then, f^{min} is a fixed point of F^A. Thus,

(1) $p^{min} \;\leq\; P^A(\sigma) \;=\; lfp\left(F^A\right) \;= \leq f^{min}$.

Let $A \in \mathcal{A}dv$, $\sigma \in Path_{fin}$ and $\nu = A(\sigma)$. Then,

$$\begin{aligned} p^A(\sigma) \;&=\; \sum_{t \in Supp(\nu)} \nu(t) \cdot p^A\left(\sigma \xrightarrow{\nu} t\right) \;\geq\; \sum_{t \in Supp(\nu)} \nu(t) \cdot p^{min}\left(\sigma \xrightarrow{\nu} t\right) \\ &\geq\; \min \left\{ \sum_{t \in Supp(\mu)} \mu(t) \cdot p^{min}\left(\sigma \xrightarrow{\mu} t\right) \;:\; \mu \in Steps(last(\sigma)) \right\} \\ &=\; F^{min}(p^{min})(\sigma). \end{aligned}$$

Thus, $p^{min} \geq F^{min}(p^{min})$. By Proposition 12.1.1 (page 309) $p^{min} \geq lfp(F^{min}) = f^{min}$. By (1), we get $p^{min} = f^{min}$. ⌋

Claim 2: $p^{max} = lfp(F^{max})$

Proof: Let $\sigma \in Path_{fin}$. We choose some $\nu \in Steps(last(\sigma))$ such that

$$\sum_{t \in Supp(\nu)} \nu(t) \cdot p^{max}\left(\sigma \xrightarrow{\mu} t\right) \;=\; F^{max}\left(p^{max}\right)(\sigma).$$

[29] As in Proposition 3.7.1 (page 65), the existence of a least fixed point can be derived by standard arguments of domain theory. See e.g. Proposition 12.1.1 (page 309).

For each $\epsilon > 0$ and $t \in Supp(\nu)$, we choose some $A_{\epsilon,t} \in \mathcal{A}dv$ with

$$p^{max}\left(\sigma \xrightarrow{\mu} t\right) \;\leq\; p^{A_{\epsilon,t}}\left(\sigma \xrightarrow{\nu} t\right) \;+\; \epsilon.$$

Let A_ϵ be an adversary with $A_\epsilon(\sigma) = \nu$ and $A_\epsilon\left(\sigma \xrightarrow{\nu} \sigma'\right) = A_{\epsilon,t}\left(\sigma \xrightarrow{\nu} \sigma'\right)$ for each $\sigma' \in Path_{fin}$ with $first(\sigma') = t$. Then,

$$p^{A_\epsilon}\left(\sigma \xrightarrow{\nu} t\right) \;=\; p^{A_{\epsilon,t}}\left(\sigma \xrightarrow{\nu} t\right) \;\geq\; p^{max}\left(\sigma \xrightarrow{\nu} t\right) - \epsilon.$$

Thus, for all $\epsilon > 0$:

$$p^{max}(\sigma) \;\geq\; p^{A_\epsilon}(\sigma) \;=\; \sum_{t \in Supp(\nu)} \nu(t) \cdot p^{A_\epsilon}\left(\sigma \xrightarrow{\nu} t\right)$$

$$\geq \sum_{t \in Supp(\nu)} \nu(t) \cdot p^{max}\left(\sigma \xrightarrow{\nu} t\right) \;-\; \epsilon \;=\; F^{max}\left(p^{max}\right)(\sigma) - \epsilon.$$

Thus, $p^{max} \geq F^{max}\left(p^{max}\right)$. By Proposition 12.1.1 (page 309):

(2) $p^{max} \geq lfp(F^{max}) = f^{max}$.

Let $\Sigma^A_n(\sigma) = \{\sigma' \in \Sigma^A(\sigma) : |\sigma'| \leq n\}$ and $p^A_n(\sigma) = Prob(\Sigma^A_n(\sigma) \uparrow^A)$. Then,

$$p^A(\sigma) \;=\; \lim_{n \to \infty} p^A_n(\sigma).$$

Moreover, $p^A_n(\sigma) = 1$ if $\sigma \in \Sigma$. If $\sigma \notin \Sigma$ then $p^A_0(\sigma) = 0$ and

$$p^A_{n+1}(\sigma) \;=\; \sum_{t \in Supp(A(\sigma))} A(\sigma)(t) \cdot p^A_n\left(\sigma \xrightarrow{A(\sigma)} t\right).$$

By induction on n, we get $p^A_n \leq f^{max}$. Thus, $p^A \leq f^{max}$ which yields

$$p^{max} \;=\; \sup_{A \in \mathcal{A}dv} p^A \;\leq\; f^{max}.$$

From (2), we get $p^{max} = f^{max}$. ⌋■

Corollary 3.7.4 (cf. Theorem 3.2.11, page 43) *Let $(S, Steps)$ be a concurrent probabilistic system, S_1, $S_2 \subseteq S$ and let Σ be the set of finite paths σ such that $\sigma(i) \in S_1$, $i = 0, 1, \ldots, |\sigma| - 1$, and $last(\sigma) \in S_2$. For $s \in S$ and $A \in \mathcal{A}dv$, let*

$$p^{min}(s) \;=\; \inf_{A \in \mathcal{A}dv} Prob\left(\Sigma^A(s) \uparrow^A\right), \quad p^{max}(s) \;=\; \sup_{A \in \mathcal{A}dv} Prob\left(\Sigma^A(s) \uparrow^A\right).$$

Then, p^{min} and p^{max} are the least fixed points of the operators

$$F^{min},\, F^{max} : (S \to [0,1]) \to (S \to [0,1])$$

that are defined as follows. If $s \in S_2$ then $F(f)(s) = 1$. If $s \in S \setminus (S_1 \cup S_2)$ then $F(f)(s) = 0$. If $s \in S_1 \setminus S_2$ then

$$F^{min}(f)(s) \;=\; \min\left\{ \sum_{t \in S} \mu(t) \cdot f(t) \;:\; \mu \in Steps(s) \right\},$$

$$F^{max}(f)(s) \;=\; \max\left\{ \sum_{t \in S} \mu(t) \cdot f(t) \;:\; \mu \in Steps(s) \right\}.$$

Proof: follows immediately from Proposition 3.7.3 (page 65). ■

3.7.2 Bisimulation and simulation in image-finite systems

We give the proofs for Lemma 3.4.8 and Lemma 3.4.13 that show that, in image-finite systems, (bi-)simulation can be approximated by the finitary relations $\sim_n$ and $\sqsubseteq_n$. Let $(S, Act, Steps)$ be a fixed image-finite concurrent probabilistic system with action labels and s, $s' \in S$.

Lemma 3.7.5 (cf. Lemma 3.4.8, page 56) *$s \sim s'$ iff $s \sim_n s'$ for all $n \geq 0$.*

Proof: Let $\sim' = \bigcap_{n\geq 0} \sim_n$. We have to show that $\sim = \sim'$. It is easy to see that $\sim'$ is an equivalence relation. By induction on n it can be shown that $\sim_0 \supseteq \sim_1 \supseteq \sim_2 \supseteq \ldots \supseteq \sim$. Hence, $\sim' \supseteq \sim$. In order to show that $\sim' \subseteq \sim$ we prove that $\sim'$ is a bisimulation.

For each $n \geq 0$ and each $A \in S/\sim'$, there exists a unique element $A_n \in S/\sim_n$ with $A \subseteq A_n$. Then, $A_0 = S \supseteq A_1 \supseteq A_2 \supseteq \ldots$ and $A = \bigcap A_n$.

Claim 1: If $s \xrightarrow{a} \mu$ and $A \in S/\sim'$ then $\mu[A] = \inf_{n\geq 0} \mu[A_n]$.

Proof: Since $A = \bigcap A_n$ and $A_n \supseteq A_{n+1}$ we have $1 = \mu[A_0] \geq \mu[A_1] \geq \ldots \geq \mu[A]$. We put $r = \inf_{n\geq 0} \mu[A_n]$. Clearly, $r \geq \mu[A]$. We suppose $r > \mu[A]$. Let $\Delta = r - \mu[A]$. Then, $\Delta > 0$. There exists a finite subset X of $S \setminus A$ such that $\mu[Y] < \Delta$ where $Y = S \setminus (A \cup X)$.[30] For all $n \geq 0$, $A_n = A \cup (Y \cap A_n) \cup (X \cap A_n)$. The sets A, $Y \cap A_n$ and $X \cap A_n$ are pairwise disjoint. Hence,

$$\mu[A_n] = \mu[A] + \mu[Y \cap A_n] + \mu[X \cap A_n] < \mu[A] + \Delta + \mu[X \cap A_n] = r + \mu[X \cap A_n].$$

Since $r \leq \mu[A_n]$ we get $X \cap A_n \neq \emptyset$. Since X is finite and $A_0 \supseteq A_1 \supseteq \ldots$ we get $X \cap A \neq \emptyset$. Contradiction![31] ⌋

Claim 2: $\sim'$ is a bisimulation.

Proof: Let $s \sim' s'$ and $s \xrightarrow{a} \mu$. By Claim 1 it suffices to show that there is a transition $s' \xrightarrow{a} \mu'$ such that $\mu[C] = \mu'[C]$ for all $n \geq 1$ and $C \in S/\sim_n$. Since $s \sim_n s'$ there exist transitions $s' \xrightarrow{a} \mu'_n$ such that $\mu[C] = \mu'_n[C]$ for all $C \in S/\sim_n$. Since $(S, Act, Steps)$ is image-finite the set $\{\mu'_n : n \geq 0\}$ is finite. Hence, there exists a transition $s' \xrightarrow{a} \mu'$ with $\mu' = \mu'_n$ for infinitely many n. Let $n \geq 1$ and $C \in S/\sim_n$. There exists $k \geq n$ with $\mu' = \mu'_k$. C is of the form $C = \bigcup_{i\in I} B_i$ where $(B_i)_{i\in I}$ is a family of pairwise disjoint equivalence classes with respect to $\sim_k$ (as $\sim_k \subseteq \sim_n$). Thus, $\mu[B_i] = \mu'_k[B_i] = \mu'[B_i]$ for all $B \in S/\sim_k$. Then, $\mu[C] = \sum_{i\in I} \mu[B_i] = \sum_{i\in I} \mu'[B_i] = \mu'[C]$. ⌋■

Lemma 3.7.6 (cf. Lemma 3.4.13, page 59) *$s \sqsubseteq_{\mathrm{sim}} s'$ iff $s \sqsubseteq_n s'$ for all $n \geq 0$.*

Proof: Let $\sqsubseteq' = \bigcap \sqsubseteq_n$. By induction on n it can be shown that $\sqsubseteq_0 \supseteq \sqsubseteq_1 \supseteq \ldots \supseteq \sqsubseteq_{\mathrm{sim}}$. Hence, $\sqsubseteq' \supseteq \sqsubseteq_{\mathrm{sim}}$. In order to prove that $\sqsubseteq' \subseteq \sqsubseteq_{\mathrm{sim}}$ we show that $\sqsubseteq'$ is a simulation. Let $s \sqsubseteq' s'$ and $s \xrightarrow{a} \mu$. There exist transitions $s' \xrightarrow{a} \mu'_n$ and weight functions $weight_n$ for (μ, μ'_n) with respect to $\sqsubseteq_{n-1}$. Because of the image-finiteness we get that there exists a transition $s' \xrightarrow{a} \mu'$ such that $\mu' = \mu'_n$ for infinitely many n. Let A_n be the set of pairs $(u, u') \in S \times S$ with $weight_n(u, u') \neq 0$. Let B_k, $B'_k \subseteq S$, be finite sets with

$$\sum_{s\in B_k} \mu(s) > 1 - \frac{1}{2^k}, \quad \sum_{s'\in B'_k} \mu'(s') > 1 - \frac{1}{2^k}.$$

[30] This is because $\sum_{t\notin A} \mu(t)$ is convergent.
[31] Note that by definition X is a subset of $S \setminus A$.

W.l.o.g. $B_1 \subseteq B_2 \subseteq \ldots$, $B'_1 \subseteq B'_2 \subseteq \ldots$. We define by induction on n infinite sets $I_0 \supseteq I_1 \supseteq \ldots$ of natural numbers such that the sequence $(\mathit{weight}_n(u,u'))_{n\in I_i}$ is convergent for all $(u,u') \in B_i \times B'_i$, $i \geq 1$.

We put $I_0 = \{n : \mu'_n = \mu\}$. Let $i \geq 1$. We suppose that I_{i-1} is already defined. Let $(u_1,u'_1),\ldots,(u_k,u'_k)$ be the sequence of pairwise distinct pairs $(u,u') \in B_i \times B'_i$ which do not belong to $B_{i-1}\times B'_{i-1}$. For all infinite sets J of natural numbers and each $l \in \{1,\ldots,k\}$, there exists an infinite subset $J(l)$ of J such that $(\mathit{weight}_n(u_l,u'_l))_{n\in J(l)}$ is convergent.[32]

We define $J_1 = I_{i-1}(1)$, $J_l = J_{l-1}(l)$, $l = 2,\ldots,k$ and $I_i = J_k$. Then, I_i is an infinite subset of I_{i-1} and $(\mathit{weight}_n(u,u'))_{n\in I_i}$ is convergent for all $(u,u') \in B_i \times B'_i$. We put

$$\mathit{weight} : S \times S \to [0,1], \quad \mathit{weight}(u,u') \;=\; \lim_{\substack{n\to\infty \\ n\in I_i}} \mathit{weight}_n(u,u')$$

if $(u,u') \in B_i \times B'_i$ and $\mathit{weight}(u,u') = 0$ if $(u,u') \notin B_i \times B'_i$ for all $i \geq 1$. We show that *weight* is a weight function for (μ,μ') with respect to $\sqsubseteq'$.

1. If $\mathit{weight}(u,u') > 0$ then (u,u') is contained in the countable set $\bigcup\ B_i \times B'_i$.
2. Let $u \in S$. We show that $\sum_{u'}\ \mathit{weight}(u,u') \;=\; \mu(u)$. If $\mu(u) = 0$ then $\mathit{weight}_n(u,u') = 0$ for all $u' \in S$ and $n \geq 0$. Hence, $\mathit{weight}(u,u') = 0$ for all $u' \in S$. We suppose $\mu(u) > 0$. Let i be the smallest natural number i such that $1/2^i < \mu(u)$. Then, $u \in B_j$ for all $j \geq i$. Let A' be a finite subset of S. There exists some $j \geq i$ such that $\{u' \in A' : \mu'(u') > 0\} \subseteq B'_j$. Thus,

   $$\sum_{u'\in A'} \mathit{weight}(u,u') \;=\; \lim_{\substack{n\to\infty \\ n\in I_j}} \sum_{u'\in A'} \mathit{weight}_n(u,u') \;\leq\; \mu(u)$$

   Hence, $\sum_{u'}\ \mathit{weight}(u,u') \;\leq\; \mu(u)$. In order to show that $\sum_{u'}\ \mathit{weight}(u,u') \;\geq\; \mu(u)$ it is sufficient to show that for all $\varepsilon > 0$ there exists a finite subset A' of S with

   $$\sum_{u'\in A'} \mathit{weight}(u,u') \quad\geq\quad \mu(u) - \varepsilon.$$

   Let $\varepsilon > 0$ and $j \geq i$ with $1/2^j \;<\; \varepsilon$. For all $n \in I_j$,

   $$\sum_{u'\notin B'_j} \mathit{weight}_n(u,u') \;\leq\; \sum_{u'\notin B'_j} \mu'(u') \;=\; 1 \;-\; \sum_{u'\in B'_j} \mu'(u') \;<\; \frac{1}{2^j} \;<\; \varepsilon.$$

   Hence,

   $$\sum_{u'\in B'_j} \mathit{weight}_n(u,u') \;=\; \mu(u) \;-\; \sum_{u'\notin B'_j} \mathit{weight}_n(u,u') \;>\; \mu(u) - \varepsilon$$

   for all $n \in I_j$. Therefore,

   $$\sum_{u'\in B'_j} \mathit{weight}(u,u') \;=\; \lim_{\substack{n\to\infty \\ n\in I_j}} \sum_{u'\in B'_j} \mathit{weight}_n(u,u') \;\geq\; \mu(u) - \varepsilon.$$

   Similarly, it can be shown that $\sum_u\ \mathit{weight}(u,u') \;=\; \mu'(u')$.
3. If $\mathit{weight}(u,u') > 0$ then $(u,u') \in B_i \times B'_i$ for some $i \geq 1$ and $\mathit{weight}_n(u,u') > 0$ for infinitely many n (more precisely, for almost all $n \in I_i$). Hence, $u \;\sqsubseteq_n\; u'$ for infinitely many n, and therefore $u \;\sqsubseteq'\; u'$.

■

[32] This is because $\mathit{weight}_n(u_l,u'_l) \in [0,1]$, and hence, $(\mathit{weight}_n(u_l,u'_l))_{n\in J}$ is bounded. Therefore, it contains a convergent subsequence.

Chapter 4

Probabilistic process calculi

Process calculi such as Milner's *CCS* or *SCCS* [Miln80, Miln83, Miln89], Hoare's *CSP* [Hoar85] or Bergstra & Klop's *ACP* [BeKl84] can successfully serve as high-level specification languages for compositional design and analysis of parallel systems. For specifying the quantitative behaviour of probabilistic parallel systems, various authors proposed variants of such process calculi. The main goal of that chapter is to present the basic concepts of probabilistic process calculi and how to supply them with an operational semantics based on action-labelled probabilistic processes. We mainly concentrate on the issue of *parallelism*. Detailed discussions about the several types of non-deterministic and probabilistic choices and their interplay can be found e.g. in [Lowe93b, MMS$^+$94, HMS97, HarG98, HadVi98]. Following the notations of [Lowe93b, MMS$^+$94], we use an *internal* probabilistic choice operator where the process chooses randomly which set of events/actions to offer the environment and *external* non-determinism where the environment offers a set of events/actions.[1]

Synchronous parallelism: In the synchronous parallel composition of two processes $\mathcal{P}_1$ and $\mathcal{P}_2$, the components work in a time-dependent fashion; i.e., each step of the synchronous parallel composition is composed by the independent execution of the activities of $\mathcal{P}_1$ and $\mathcal{P}_2$ within one time step. The transition probabilities of the composed system are obtained by multiplying the probabilities of the individual moves of the components $\mathcal{P}_1$ and $\mathcal{P}_2$. This reflects the assumption that $\mathcal{P}_1$ and $\mathcal{P}_2$ work independently between the synchronization points.

In the literature, several types of synchronous parallel composition for probabilistic processes are proposed. [GJS90, JoSm90, vGSST90, SmSt90, LaSk92, Toft94, KwNo98b] deal with the *(synchronous) product* $\mathcal{P}_1 \times \mathcal{P}_2$ in the style of Milner's *SCCS* [Miln83] where each step of $\mathcal{P}_1 \times \mathcal{P}_2$ is composed by *exactly one action* of $\mathcal{P}_1$ and $\mathcal{P}_2$. Other authors, e.g. [FHZ93, HarG98], focuss on the concept of a *lazy (synchronous) product* $\mathcal{P}_1 \otimes \mathcal{P}_2$ where the processes $\mathcal{P}_1$ and $\mathcal{P}_2$ have to synchronize on certain "synchronization points" but perform *sequences of actions* independently between these synchronization points. One side-effect of the use of lazy synchronous parallelism (in a non-probabilistic or probabilistic setting) is that the transition system representation of $\mathcal{P}_1 \otimes \mathcal{P}_2$ is in general much

[1] It should be noticed that the different types of probabilistic choice operators lead to different restriction operators. See Remark 4.2.4, page 81.

smaller than those of $\mathcal{P}_1 \times \mathcal{P}_2$.[2] In this sense, the use of the lazy product can also be viewed as an abstraction technique that attacks the state explosion problem.

In most cases, in the probabilistic extensions of synchronous calculi, the concept of non-deterministic choice is replaced by probabilistic choice. Typically, such languages (with synchronous parallelism and probabilistic choice rather than non-determinism) are equipped with an operational semantics that uses a model based on Markov chains (such as fully probabilistic processes with action labels) [GJS90, JoSm90, vGSST90, Toft90, LaSk92, Toft94] but – as e.g. in the case of probabilistic extensions of *SCCS* – such languages can also be provided with operational semantics that are based on the reactive or stratified view [vGSST90, Toft90, Toft94, KwNo96, Norm97, KwNo98b]. Moreover, some of these languages – together with their stratified semantics – can be used to reason about priority [SmSt90, Toft90, Toft94]. Synchronous calculi with non-deterministic and probabilistic choice are considered e.g. in [FHZ93, Norm97, KwNo98b]. The semantic models of these calculi can be viewed as generalizations of the reactive model.

Asynchronous parallelism: There are several probabilistic extensions of asynchronous calculi with non-deterministic and probabilistic choice operators. The underlying (asynchronous) parallel operators allow *communication* on certain actions (e.g. communication in the *CCS*-style on "complementary" actions or *CSP*-like communication on common actions) but also *independent* evolvement of the components. The operational semantics of such calculi are usually given in terms of probabilistic transition systems with non-determinism where the independent evolvement of the components is modelled by *interleaving*. For instance, [HaJo90, Hans91, YiLa92, Yi94] extend Milner's *CCS* by a probabilistic choice operator; probabilistic variants of Hoare's *CSP* are considered e.g. in [Lowe93a, Lowe93b, MMS$^+$94, Lowe95, Seid95].

Probabilistic shuffle: Baeten, Bergstra & Smolka [BBS92] introduce a modification of *ACP* [BeKl84] which uses probabilistic choice instead of non-determinism and several types of probabilistic shuffle operators (with possible communication). The probabilistic shuffle operators are *parametrized by the probabilities* for a communication and the autonomous moves of the components. The operational semantics is based on fully probabilistic processes with action labels. Several authors, e.g. [SCV92, NúdF95, dAHK98], took up the idea of using probability parameters that associate weights to the possible steps of the composition (communication on certain actions or independent evolvement of the components).[3] [GLN$^+$97] introduce the process algebra *PTPA* for generative (and timed) processes in which probabilistic shuffle is modelled with the help of a *normalization function* (rather than probability parameters). In the approach of [GLN$^+$97], the components $\mathcal{P}_1$ and $\mathcal{P}_2$ of the probabilistic shuffle $\mathcal{P}_1 \|_A \mathcal{P}_2$ must synchronize on the actions $a \in A$ while the actions $a \notin A$ are performed autonomously. The transition probabilities of $\mathcal{P}_1 \|_A \mathcal{P}_2$ are defined with the help of the normalization function that sums up the probabilities for $\mathcal{P}_1$ and $\mathcal{P}_2$ to participate in an a-step of $\mathcal{P}_1 \|_A \mathcal{P}_2$ where a ranges over all possible actions of $\mathcal{P}_1 \|_A \mathcal{P}_2$.[4] Both types of probabilistic shuffle (the one that use probability parameters and the one that use a normalization function) can be viewed as the interleaved execution of the two components $\mathcal{P}_1$ and $\mathcal{P}_2$ (extended by certain synchroniza-

[2] This is because the lazy product abstracts from certain local states.

[3] See [dAHK98] for an overview of these parametrized shuffle operators.

[4] Similar ideas are used in the approachs of e.g. [Chri90b, CSZ92, YCDS94, NdFL95] that define a notion of parallel composition $\mathcal{P} \| \mathcal{T}$ of a generative probabilistic process $\mathcal{P}$ and some kind of "test" $\mathcal{T}$.

tion mechanisms) with respect to a fixed randomized scheduler. This scheduler decides randomly which of the possible steps is executed next: either a synchronization action or an individual move of $\mathcal{P}_1$ or an individual move of $\mathcal{P}_2$. The probabilities of the possible steps are given either by the parameters of the probabilistic shuffle operator or by the normalization function that depends on the local states of $\mathcal{P}_1$ and $\mathcal{P}_2$.

Modelling asynchronicity by synchronicity: In the non-probabilistic synchronous case (i.e. in the case of Milner's *SCCS*), a delay operator ∂ can be defined which makes it possible e.g. to force a process to wait for a possible communication partner and to embed the asynchronous calculus *CCS* into the synchronous calculus *SCCS* [Miln83]. Intuitively, $\partial\mathcal{P}$ behaves as $\mathcal{P}$ but it may idle for indefinitely many time steps before performing the first action. Formally, $\partial\mathcal{P}$ is given by the process equation $\partial\mathcal{P} \stackrel{\text{def}}{=} \mathcal{P} + 1; \partial\mathcal{P}$ which states that $\partial\mathcal{P}$ decides non-deterministically to behave as $\mathcal{P}$ or to be idle in the next step. Here, $+$ denotes non-deterministic choice, ; sequential composition and 1 the idle action. In absence of a non-deterministic choice operator, the delay operator ∂ cannot be defined; hence, if non-deterministic choice is replaced by probabilistic choice (as it is the case for several probabilistic variants of synchronous process calculi proposed in the literature), it is no longer true that asynchronicity can be reduced to the synchronous case (at least, the author does not see how).

Organization of that chapter: We study three calculi. The first two are standard extensions of Milner's *CCS* and *SCCS*; the third a variant of *SCCS* that uses a lazy synchronous parallel composition. In Section 4.1 we consider an asynchronous calculus with *CCS*-like communication and non-deterministic and probabilistic choice (similar to the calculi of [HaJo90, Hans91, YiLa92]) and give an operational semantics based on action-labelled concurrent probabilistic processes. In Sections 4.2 and 4.3, we work with synchronous calculi with probabilistic choice (but without non-determinism) which are supplied with an operational semantics based on generative (i.e. action-labelled fully probabilistic) processes. The calculus in Section 4.2, called *PSCCS*, is a probabilistic extension of Milner's *SCCS* that works with a parallel composition $\mathcal{P}_1 \times \mathcal{P}_2$ where the components $\mathcal{P}_1$, $\mathcal{P}_2$ synchronize on all actions. Basically, it agrees with the calculi studied in [GJS90, JoSm90, vGSST90, Toft94]. In Section 4.3 we propose a probabilistic calculus which uses a lazy synchronous parallel composition $\mathcal{P}_1 \otimes \mathcal{P}_2$ where $\mathcal{P}_1$ and $\mathcal{P}_2$ have to synchronize on all visible actions while they evolve independently on their internal actions.

Modelling recursion by declarations and process equations: For all three calculi, we model recursion by *declarations*. We use process variables (of some fixed set *ProcVar*) in the statements. The process variables can be interpreted as procedure names. The bodies of these procedures are given by declarations. Formally, a declaration is a function *decl* that assign to each process variable Z a statement $decl(Z)$ (that also might contain process variables, i.e. recursive procedure calls). A *program* is a pair $\mathcal{P} = \langle decl, s\rangle$ consisting of a declaration *decl* and a statement s. The intended meaning of a program $\mathcal{P} = \langle decl, s\rangle$ is that the behaviour of $\mathcal{P}$ is given by the statement s where each occurrence of a process variable Z in s is viewed as a recursive procedure call. This corresponds to the use of *process equations* that we use in our examples. Let $Z_1, \ldots, Z_k$ be pairwise distinct process variables and $s_1, \ldots, s_k$ statements. Then, we write $Z_j \stackrel{\text{def}}{=} s_j$, $j = 1, \ldots, k$, to denote that Z_j stands for the recursive procedure whose body is given by s_j. That is, we deal with the declaration *decl* where $decl(Z_j) = s_j$, $j = 1, \ldots, k$, and identify Z_j with

the program $\langle decl, Z_j \rangle$. If op is a n-ary operator symbol of the underlying process calculus (e.g. a binary parallel composition operator $\|$ or the 1-ary (action-)prefix operator $s \mapsto a.s$) and $\mathcal{P}_i = \langle decl, s_i \rangle$, $i = 1, \ldots, n$, are programs then we write $op(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ as short for the program $\langle decl, op(s_1, \ldots, s_n) \rangle$.

4.1 *PCCS*: an asynchronous probabilistic calculus

In this section we consider a probabilistic extension of Milner's *CCS* [Miln89] which is based on the calculus of [Hans91] (see also [HaJo90, YiLa92, Yi94]). The syntax of our calculus, called *PCCS*, in obtained from *CCS* by replacing the prefix operator $a.s$ by an *action-guarded probabilistic choice* operator

$$a.\left(\sum_{i \in I} [p_i]s_i\right)$$

where p_i are real numbers between 0 and 1 denoting the probability that after performing a the above process becomes s_i (provided that the statements s_i are pairwise distinct).

In what follows, *ProcVar* is a set of process variables and *Act* is a finite nonempty set of atomic actions which contains an internal action τ (representing internal computations of a process, not visible for the environment) and which is equipped with a function $Act \to Act$, $a \mapsto \overline{a}$, where $\overline{\tau} = \tau$ and $\overline{\overline{a}} = a$ for all $a \in Act$. If $L \subseteq Act$ then we put $\overline{L} = \{\overline{a} : a \in L\}$. For α to be a visible action, $\overline{\alpha}$ is called the *complementary* action of α. Synchronization of processes is only possible by performing complementary actions α and $\overline{\alpha}$. The result of a synchronization is supposed to be invisible, i.e. it is described by the internal action τ.

Syntax of *PCCS* statements: *PCCS* statements are built from the production system shown in Figure 4.1 (page 74). Here, $a \in Act$, $Z \in ProcVar$, L is a subset of $Act \setminus \{\tau\}$

$$s \quad ::= \quad nil \;\Big|\; Z \;\Big|\; a.\left(\sum_{i \in I} [p_i]s_i\right) \;\Big|\; s_1 + s_2 \;\Big|\; s_1 \parallel s_2 \;\Big|\; s \setminus L \;\Big|\; s[\ell]$$

Figure 4.1: Syntax of *PCCS* statements

with $\overline{L} = L$, $\ell : Act \to Act$ is a relabelling function (i.e. $\ell(\overline{\alpha}) = \overline{\ell(\alpha)}$ for all visible actions α and $\ell(\tau) = \tau$), I is a nonempty countable indexing set and $(p_i)_{i \in I}$ a family of real numbers $p_i \in]0, 1]$ such that $\sum_{i \in I} p_i = 1$. For finite indexing set $I = \{i_1, \ldots, i_n\}$, we also write $a.([p_{i_1}].s_{i_1} \oplus \ldots \oplus [p_{i_n}]s_{i_n})$ instead of $a.(\sum_{i \in I} [p_i]s_i)$. $a.s$ stands short for $a.([1]s)$. $Stmt_{PCCS}$ (or shortly $Stmt$) denotes the set of all *PCCS* statements. *PCCS* denotes the set of all *PCCS* programs, i.e. all pairs $\mathcal{P} = \langle decl, s \rangle$ where $decl$ is a declaration (a function $decl : ProcVar \to Stmt_{PCCS}$) and s a *PCCS* statement.

The intended meaning of the statements is as follows. *nil* stands for a process which does not perform any action. The idea behind action-guarded probabilistic choice is that

$a.\,(\sum_{i\in I}[p_i]s_i\,)$ first performs the action a and then randomly chooses to behave as t afterwards according to the distribution μ where

$$\mu(t) \;=\; \sum_{\substack{i\in I\\ s_i=t}} p_i.$$

$+$ models non-deterministic choice, that is, $s_1 + s_2$ either behaves like s_1 or like s_2. $\|$ denotes the parallel composition with *CCS*-style communication on complementary actions (i.e., in $s_1\|s_2$, s_1 and s_2 can evolve independently but may also communicate via visible actions α and $\overline{\alpha}$). The operators $s \mapsto s \setminus L$, $s \mapsto s[\ell]$ model restriction and relabelling: $s \setminus L$ behaves like s as long as s does not perform an action $\alpha \in L$. $s[\ell]$ behaves like s where each action $\alpha \in Act$ is replaced by $\ell(\alpha)$.

Example 4.1.1 [The controller system] We consider a simple controller system of a plant that tests certain products. There are n "testing machines" where each of them can test a product (via an action called *test*). We assume that the reliability of the production is known: with probability 1/100 the product fails the test in which case the product is returned to the production department (via an action called *return*); with probability 99/100 the product passes the test in which case the product is transmitted to the vending department (via an action called *release*). Each of the testing machines can be specified by the *PCCS* program

$$Test \;\stackrel{\text{def}}{=}\; test.\Big(\Big[\tfrac{1}{100}\Big]\,return.Test \;\oplus\; \Big[\tfrac{99}{100}\Big]\,release.Test\Big).$$

Thus, the controller system (with n testing machines) is given by

$$\mathcal{P}_n \;\stackrel{\text{def}}{=}\; \underbrace{Test\|\ldots\|Test}_{n}.$$

Here, we use process equations to describe the underlying declaration as explained on page 73. ■

Remark 4.1.2 [*PCCS* **à la Hansson & Jonsson**] Our language *PCCS* is closely related to the calculus (also called *PCCS*) of [HaJo90] (see also [Hans91, YiLa92] and the extended version in [Yi94]). In contrast to our approach, [HaJo90] allow for general probabilistic choice $\sum[p_i]s_i$ (while we use action-guarded probabilistic choice $a.(\sum[p_i]s_i)$). For instance, [HaJo90] allow statements like $\left[\frac{1}{3}\right]a.nil \oplus \left[\frac{2}{3}\right]b.nil$ which stands for a process that offer a with probability 1/3 and b with probability 2/3 and terminates after performing a or b respectively. Thus, syntactically, our language *PCCS* can be viewed as a subcalculus of the calculus of [HaJo90]. Vice versa, using similar ideas as for the "inference" from action-labelled stratified systems to concurrent probabilistic systems (see Remark 3.4.6, page 55), the calculus of [HaJo90] can be embedded syntactically into ours by introducing a special action symbol a_{random} (which represents any activity that resolves the probabilistic choice, e.g. one might think of a_{random} to stand for "tossing a fair coin") and

$$\text{replacing}\;\; \sum_{i\in I}[p_i]s_i \;\;\text{ by }\;\; a_{random}.\left(\sum_{i\in I}[p_i]s_i\;\right).$$

It should be noticed that the above mentioned "embeddings" are only *syntactic* transformations. Even though the intended meanings are similar the formal semantics do *not*

$Z \xrightarrow{a}_{decl} \mu$ if $decl(Z) \xrightarrow{a}_{decl} \mu$

$$a.\left(\sum_{i\in I}[p_i]s_i\right) \xrightarrow{a}_{decl} \mu \quad \text{where} \quad \mu(s) = \sum_{\substack{i\in I\\ s_i=s}} p_i$$

$s_1 + s_2 \xrightarrow{a}_{decl} \mu$ if $s_1 \xrightarrow{a}_{decl} \mu$ or $s_2 \xrightarrow{a}_{decl} \mu$

$s_1 \parallel s_2 \xrightarrow{a}_{decl} \mu$ if one of the following three conditions is satisfied:

(i) $s_1 \xrightarrow{a}_{decl} \mu_1$ and $\mu(s) = \begin{cases} \mu_1(s_1') & : \text{ if } s = s_1' \parallel s_2 \\ 0 & : \text{ otherwise} \end{cases}$

(ii) $s_2 \xrightarrow{a}_{decl} \mu_2$ and $\mu(s) = \begin{cases} \mu_2(s_2') & : \text{ if } s = s_1 \parallel s_2' \\ 0 & : \text{ otherwise} \end{cases}$

(iii) $a = \tau$ and there exists $\alpha \in Act \setminus \{\tau\}$ with

$$s_1 \xrightarrow{\alpha}_{decl} \mu_1 \quad \text{and} \quad s_2 \xrightarrow{\overline{\alpha}}_{decl} \mu_2$$

such that

$$\mu(s) = \begin{cases} \mu_1(s_1') \cdot \mu_2(s_2') & : \text{ if } s = s_1' \parallel s_2' \\ 0 & : \text{ otherwise} \end{cases}$$

$s \setminus L \xrightarrow{a}_{decl} \mu$ if $s \xrightarrow{a}_{decl} \mu'$, $a \notin L$ and $\mu(s) = \begin{cases} \mu'(s') & : \text{ if } s = s' \setminus L \\ 0 & : \text{ otherwise} \end{cases}$

$s[\ell] \xrightarrow{a}_{decl} \mu$ if $s \xrightarrow{b}_{decl} \mu'$, $\ell(b) = a$ and $\mu(s) = \begin{cases} \mu'(s') & : \text{ if } s = s'[\ell] \\ 0 & : \text{ otherwise} \end{cases}$

Figure 4.2: Operational semantics of *PCCS*

coincide since the interpretations of + and $\parallel$ are different. While we deal with an operational semantics based on concurrent probabilistic processes [HaJo90] work with an operational semantics based on the stratified (alternating) model where action-labelled and probabilistic transitions are distinguished. In contrast to our rules for non-determinism + or parallel composition $\parallel$ (which are immediate derivations of Milner's rules for + and parallel composition and in the style of Segala [Sega95a] who defines a *CSP*-like parallel composition for concurrent probabilistic systems), the rules of [HaJo90] are based on a higher priority for the probabilistic transitions than the action-labelled transitions. In the approach of [HaJo90], the summands s_1 and s_2 of the non-deterministic choice $s_1 + s_2$ first have to perform their probabilistic transitions before the non-determinism is resolved. Similarly, by the rules of [HaJo90], the parallel composition $s_1 \| s_2$ cannot perform an action-labelled transition unless both components s_1, s_2 have resolved their probabilistic choices. ■

Operational semantics for *PCCS*: Using the classical SOS-style à la Plotkin [Plot81], we give an operational semantics for *PCCS* based on concurrent probabilistic processes with action labels. Let *decl* be a declaration. We define the transition relation $\longrightarrow_{decl} \subseteq Stmt \times Act \times Distr(Stmt)$ to be the smallest relation satisfying the rules of Figure 4.2 on page 76. Here, we write $s \xrightarrow{a}_{decl} \mu$ instead of $(s, a, \mu) \in \longrightarrow_{decl}$. The operational semantics assigns to each *PCCS* program $\mathcal{P} = \langle decl, s \rangle$ the action-labelled concurrent probabilistic process $\mathcal{O}[\![\mathcal{P}]\!] = (Stmt, Act, Steps^{decl}, s)$ where $Steps^{decl}(s) = \{(a, \mu) : s \xrightarrow{a}_{decl} \mu\}$.

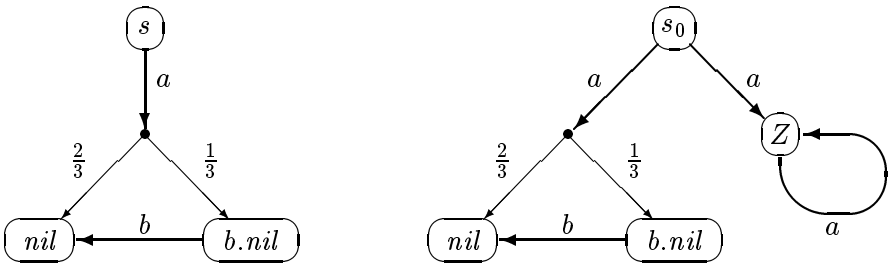


Figure 4.3:

Example 4.1.3 The operational semantics of the *PCCS* program $\langle decl, s \rangle$ where $s = a.\left(\left[\frac{1}{3}\right] b.nil \oplus \left[\frac{2}{3}\right] nil\right)$ (and where *decl* is an arbitrary declaration) is the process shown on the left of Figure 4.3 (page 77). The picture on the right shows the operational semantics $\mathcal{O}[\![\mathcal{P}_0]\!]$ of the recursive program $\mathcal{P}_0 = \langle s_0, decl_0 \rangle$ where $s_0 = s + Z$ and $decl_0(Z) = a.Z$. Figure 4.4 (page 78) shows the semantics of the program $\langle decl, (s_1 \| s_2) \setminus L \rangle$ where

$$s_1 = \alpha.\left(\left[\tfrac{1}{4}\right] \beta.nil \oplus \left[\tfrac{3}{4}\right] \overline{\alpha}.nil\right), \quad s_2 = \overline{\alpha}.\left(\left[\tfrac{1}{3}\right] \overline{\beta}.nil \oplus \left[\tfrac{2}{3}\right] \alpha.nil\right) + a.nil$$

and $L = \{\alpha, \overline{\alpha}, \beta, \overline{\beta}\}$. $t_1 \|' t_2$ stands short for $(t_1 \| t_2) \setminus L$. The a-transition of $s_1 \|' s_2$ represents the case where the a-transition of s_2 is chosen non-deterministically. Thus, $s_1 \|' s_2$ can make an a-move where s_1 does not participate, i.e. does not change its local state. The τ-transition of $s_1 \|' s_2$ stands for the synchronization of α and $\overline{\alpha}$. For instance, with probability $\frac{1}{4} \cdot \frac{1}{3} = \frac{1}{12}$, s_1 moves to the local state $\beta.nil$ and s_2 to $\overline{\beta}.nil$. In the global states $s_1 \|' nil$, $\beta.nil \|' \alpha.nil$ and $\overline{\alpha}.nil \|' \overline{\beta}.nil$ no actions are possible because of the restriction operator while in the global states $\beta.nil \|' \overline{\beta}.nil$ and $\overline{\alpha}.nil \|' \alpha.nil$ further synchronizations take place. ■

In what follows, we identify each program $\mathcal{P}$ with its operational meaning $\mathcal{O}[\![\mathcal{P}]\!]$ and lift bisimulation equivalence $\sim$ and the simulation preorder $\sqsubseteq_{\text{sim}}$ to *PCCS* programs. We define $\mathcal{P} \sim \mathcal{P}'$ iff $\mathcal{O}[\![\mathcal{P}]\!] \sim \mathcal{O}[\![\mathcal{P}']\!]$ and $\mathcal{P} \sqsubseteq_{\text{sim}} \mathcal{P}'$ iff $\mathcal{O}[\![\mathcal{P}]\!] \sqsubseteq_{\text{sim}} \mathcal{O}[\![\mathcal{P}']\!]$. It can easily be shown that all operators preserve bisimilarity and the simulation preorder. More precisely, if *decl* is a declaration then we define $\sim^{decl}$ and $\sqsubseteq^{decl}_{\text{sim}}$ as binary relation on *Stmt* by $s \sim^{decl} s'$ iff $\langle decl, s \rangle \sim \langle decl, s' \rangle$ and $s \sqsubseteq^{decl}_{\text{sim}} s'$ iff $\langle decl, s \rangle \sqsubseteq_{\text{sim}} \langle decl, s' \rangle$. Let $\theta = \sim^{decl}$ or $\theta = \sqsubseteq^{decl}_{\text{sim}}$. Then:

1. If $s_i \; \theta \; s'_i$, $i \in I$, then $a.(\sum [p_i] s_i) \;\; \theta \;\; a.(\sum [p_i] s'_i)$.
2. If $s_1 \; \theta \; s'_1$ and $s_2 \; \theta \; s'_2$ then $s_1 + s_2 \;\; \theta \;\; s'_1 + s'_2$.
3. If $s_1 \; \theta \; s'_1$ and $s_2 \; \theta \; s'_2$ then $s_1 \| s_2 \;\; \theta \;\; s'_1 \| s'_2$.
4. If $s \; \theta \; s'$ then $s[\ell] \;\; \theta \;\; s'[\ell]$.

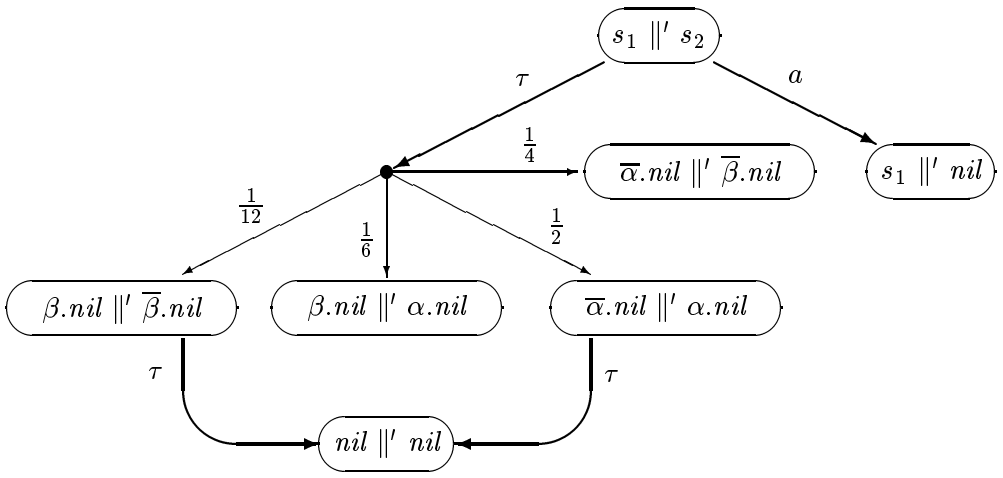


Figure 4.4: Example for the operational semantics of a parallel *PCCS* program

5. If $s \ \theta \ s'$ then $s \setminus L \ \ \theta \ \ s' \setminus L$.
6. $Z \ \theta \ decl(Z)$

Similarly, it can be shown that the weak and branching bisimulation equivalences of Segala & Lynch [SeLy94] are preserved by all operators with the exception of the non-deterministic choice operator $+$. The fact that these relations are congruences with respect to the parallel composition $\parallel$ can be derived from the results of [Sega95a].[5]

Example 4.1.4 [Simplified representation of the controller system] We consider the controller system of Example 4.1.1 on page 75. The state space of $\mathcal{O}[\![\mathcal{P}_n]\!]$ consists of 3^n states (as each of the n components *Test* is described by three states). In order to get a "simpler" description of the behaviour of $\mathcal{P}_n$ with a smaller state space (whose size is not exponential in the number of testing machines) one can use counters c_{test}, c_{return} and $c_{release}$ where the value of the counter c_a gives rise about the number of testing machines that are in the local state where the action a has to be performed next. That is, we consider the *PCCS* program $\mathcal{P}'_n \stackrel{\text{def}}{=} \mathcal{P}(n,0,0)$ where

$$\mathcal{P}(m,k,l) \stackrel{\text{def}}{=} Test(m,k,l) \ + \ Return(m,k,l) \ + \ Release(m,k,l)$$

and

$$Test(m,k,l) \stackrel{\text{def}}{=} \begin{cases} test.\left(\left[\frac{1}{100}\right]\mathcal{P}(m-1,k+1,l) \oplus \left[\frac{99}{100}\right]\mathcal{P}(m-1,k,l+1)\right) & : \text{ if } m \geq 1 \\ nil & : \text{ otherwise} \end{cases}$$

$$Return(m,k,l) \stackrel{\text{def}}{=} \begin{cases} return.\mathcal{P}(m+1,k-1,l) & : \text{ if } k \geq 1 \\ nil & : \text{ otherwise} \end{cases}$$

$$Release(m,k,l) \stackrel{\text{def}}{=} \begin{cases} release.\mathcal{P}(m+1,k,l-1) & : \text{ if } l \geq 1 \\ nil & : \text{ otherwise.} \end{cases}$$

[5] Note that our parallel composition is similar to the one introduced in [Sega95a]; the only difference is that [Sega95a] uses another communication mechanism which is based on *CSP*-style synchronization on common actions and independent evolvement on all other actions (rather than the *CCS*-style synchronization on complementary actions that we use).

Intuitively, the first component m is the value of the counter c_{test} for the testing machines that are in their initial state (i.e. that have to test a product) while the second component k and the third component l stand for the values of the counters c_{return} and $c_{release}$ respectively. It is easy to see that $\mathcal{O}[\![\mathcal{P}_n]\!] \sim \mathcal{O}[\![\mathcal{P}'_n]\!]$ and that $\mathcal{O}[\![\mathcal{P}'_n]\!]$ has $(n+1)(n+2)/2$ states. Thus, by the compositionality of bisimulation equivalence $\sim$, for investigating the behaviour of the controller system $\mathcal{P}_n$ in an environment $\ldots \| \mathcal{P}_n \| \ldots$ one can switch from the exponential-large system $\mathcal{O}[\![\mathcal{P}_n]\!]$ to the polynomial-large system $\mathcal{O}[\![\mathcal{P}'_n]\!]$. ■

4.2 *PSCCS*: a synchronous probabilistic calculus

The specification language *PSCCS* was introduced by Giacalone, Jou & Smolka [GJS90] and later considered by several authors, e.g. [JoSm90, vGSST90, LaSk92].[6] *PSCCS* uses a *SCCS*-style synchronous parallel composition $s_1 \times s_2$ where all transitions of the product $s_1 \times s_2$ are composed by individual moves of s_1 and s_2; more precisely, in each step of $s_1 \times s_2$, each of the components s_1 and s_2 performs exactly one step. We assume a commutative and associative function $Act \times Act \to Act$, $(a, b) \mapsto a * b$ where $a * b$ stands for the result of the simultaneous execution of the actions a and b.[7] In contrast to Section 4.1 (and the following Section 4.3) the special action symbol τ is not needed here. Nevertheless, it might be contained in *Act* in which case it does not play a distinguished role and is treated as any other action.

Syntax of *PSCCS*: Let *ProcVar* be a set of process variables. *PSCCS* statements are given by the grammar shown in Figure 4.5 (page 79). Here, $Z \in ProcVar$, $a \in Act$,

$$s \quad ::= \quad nil \;\Big|\; Z \;\Big|\; a.s \;\Big|\; \sum_{i \in I} [p_i] s_i \;\Big|\; s_1 \times s_2 \;\Big|\; s \setminus L \;\Big|\; s[\ell]$$

Figure 4.5: Syntax of *PSCCS* statements

$L \subseteq Act$, I is a countable indexing set, p_i are real numbers $p_i \in]0, 1]$ with $\sum p_i = 1$ and $\ell : Act \to Act$ is a relabelling function. $Stmt_{PSCCS}$ (or shortly *Stmt*) denotes the collection of all *PSCCS* statements. A *PSCCS* program is a pair $\mathcal{P} = \langle decl, s \rangle$ where s is a statement and *decl* a declaration (i.e. a function $decl : ProcVar \to Stmt_{PSCCS}$).

The intended meanings of inaction *nil*, prefixing $a.s$, restriction $s \setminus L$ and relabelling $s[\ell]$ are as in the case of *CCS* or *PCCS*. $\sum_{i \in I} [p_i] s_i$ models probabilistic choice: if s_i, $i \in I$, are pairwise distinct then, with probability p_i, $\sum [p_i] s_i$ behaves as s_i. For finite indexing set $I = \{i_1, \ldots, i_n\}$, we also write $[p_{i_1}].s_{i_1} \oplus \ldots \oplus [p_{i_n}] s_{i_n}$ instead of $\sum_{i \in I} [p_i] s_i$. If s_i may perform the action a_i and becomes t_i afterwards $i = 1, 2$, then $s_1 \times s_2$ may move to $t_1 \times t_2$ via the action $a_1 * a_2$. In the synchronous parallel composition (also called product) $s_1 \times s_2$,

[6] Note that several authors use the name *PCCS* for that calculus while we call it *PSCCS* since it is an extension of *SCCS* and to avoid confusions with the language *PCCS* considered in Section 4.1.

[7] In *SCCS* [Miln83], $(Act, *)$ is supposed to be an Abelian monoid with a unit 1. For our purposes, it suffices to assume that $*$ is commutative and associative.

$$\mathbf{P}^{decl}(Z,a,t) \;=\; \mathbf{P}^{decl}(decl(Z),a,t), \quad \mathbf{P}^{decl}(a.s,a,s) \;=\; 1$$

$$\mathbf{P}^{decl}\left(\sum_{i\in I}[p_i]s_i,a,t\right) \;=\; \sum_{i\in I} p_i\cdot\mathbf{P}^{decl}(s_i,a,t)$$

$$\mathbf{P}^{decl}(s_1\times s_2,a,t_1\times t_2) \;=\; \sum_{(b,c)\in Syn_a}\mathbf{P}^{decl}(s_1,b,t_1)\cdot\mathbf{P}^{decl}(s_2,c,t_2)$$

$$\mathbf{P}^{decl}(s\setminus L,a,t\setminus L) \;=\; \mathbf{P}^{decl}(s,a,t) \;\text{ if } a\notin L$$

$$\mathbf{P}^{decl}(s[\ell],a,t[\ell]) \;=\; \sum_{b\in\ell^{-1}(a)}\mathbf{P}^{decl}(s,b,t)$$

Figure 4.6: Equations for $\mathbf{P}^{decl}$ in the case of *PSCCS*

the probabilistic choices in s_1 and s_2 are supposed to be resolved causally independently. Thus, the probability of the transition of $s_1 \times s_2$ to the state $t_1 \times t_2$ via the action a is obtained by summing up the products of the probabilities for s_1 to move to t_1 via an action b and s_2 to move to t_2 via an action c such that $a = b * c$.

Operational semantics of *PSCCS*: With slight differences – that mainly arises from the fact that we do not allow substochastic states in fully probabilistic systems and deal with internal probabilistic choice (which leads to another interpretation of the restriction operator; cf. Remark 4.2.4, page 81) – we provide *PSCCS* with the operational *generative* semantics of [GJS90, JoSm90, vGSST90]. For this, we fix a declaration *decl* and define the transition probability function $\mathbf{P}^{decl} : Stmt \times Act \times Stmt \to [0,1]$ as the least function that satisfies the equations of Figure 4.6 on page 80.[8] Here, for $a \in Act$,

$$Syn_a = \{(b,c) \in Act \times Act : b * c = a\}.$$

Example 4.2.1 We consider the recursive *PSCCS* program $\langle decl, Z\rangle$ where

$$decl(Z) = \left[\tfrac{1}{3}\right] Z \;\oplus\; \left[\tfrac{2}{3}\right] a.nil.$$

We get the equation $\mathbf{P}^{decl}(Z,a,nil) \;=\; \frac{1}{3}\cdot\mathbf{P}^{decl}(Z,a,nil) + \frac{2}{3}$ whose unique solution is

$$\mathbf{P}^{decl}(Z,a,nil) = 1.$$

For the recursive *PSCCS* program $\langle decl, Z'\rangle$ where $decl(Z') = Z'$ we get the equation $\mathbf{P}^{decl}(Z',a,t) = \mathbf{P}^{decl}(Z',a,t)$. Clearly, the least solution is 0. Hence, $\mathbf{P}^{decl}(Z',a,t) = 0$ for all $a \in Act$ and $t \in Stmt$. ■

The so defined transition probability function $\mathbf{P}^{decl} : Stmt \times Act \times Stmt \to [0,1]$ is substochastic which means that the sum of the probabilities for the outgoing transitions

[8] The existence of a least function satisfying the equations of Figure 4.6 follows with standard domain-theoretic arguments; see e.g. Propostion 12.1.1 on page 309.

Figure 4.7: Examples for the operational semantics of *PSCCS* programs

of a certain statement might be a real number between 0 and 1 (rather than 0 or 1). For instance, for the statement $s_1 = \left[\frac{1}{3}\right] a.nil \oplus \left[\frac{2}{3}\right] nil$ we have $\mathbf{P}^{decl}(s_1, a, nil) = 1/3$ and $\mathbf{P}^{decl}(s_1, b, t) = 0$ if $(b, t) \neq (a, nil)$. For this reason, we introduce an auxiliary statement $\mathbf{0}$ that denotes inaction and a special action symbol 0 which is needed for modelling transitions from a state $s \in Stmt$ to $\mathbf{0}$. We define $Stmt_{\mathbf{0}} = Stmt \cup \{\mathbf{0}\}$, $Act_0 = Act \cup \{0\}$ and extend $\mathbf{P}^{decl}$ to a function $Stmt_{\mathbf{0}} \times Act_0 \times Stmt_{\mathbf{0}} \to [0, 1]$ (also called $\mathbf{P}^{decl}$) as follows. For $s \in Stmt$, we put

$$\mathbf{P}^{decl}(s, 0, \mathbf{0}) \;=\; 1 \;-\; \sum_{a \in Act} \sum_{t \in Stmt} \mathbf{P}^{decl}(s, a, t).$$

and $\mathbf{P}^{decl}(\cdot) = 0$ in all remaining cases (e.g. $\mathbf{P}^{decl}(s, 0, t) = 0$ if $t \in Stmt$ or $\mathbf{P}^{decl}(\mathbf{0}, a, t) = 0$ for all $a \in Act_0$ and $t \in Stmt_{\mathbf{0}}$). The operational semantics assigns to each *PSCCS* program $\mathcal{P} = \langle decl, s \rangle$ the fully probabilistic process $\mathcal{O}[\![\mathcal{P}]\!] = (Stmt_{\mathbf{0}}, Act_0, \mathbf{P}^{decl}, s)$.

Example 4.2.2 The operational semantics of $\langle decl, s_1 \rangle$ where $s_1 = \left[\frac{1}{3}\right] a.nil \oplus \left[\frac{2}{3}\right] nil$ is the process shown on the left of Figure 4.7 on page 81. The picture on the right shows the operational semantics of $\langle decl, s_1 \times s_2 \rangle$ where s_1 is as before and $s_2 = \left[\frac{1}{4}\right] b.nil \oplus \left[\frac{3}{4}\right] c.nil$. In both cases, *decl* is an arbitrary declaration. ■

Remark 4.2.3 There are several possibilities for a formal definition of the transition probability function $\mathbf{P}^{decl}$. Some authors, e.g. [vGSST90], use indices for the transitions and rules like

$$\text{if } s_j \xrightarrow{a[p]}_k t \text{ for some } j \in I \text{ then } \sum_{i \in I} [p_i] s_i \;\xrightarrow{a[p]}_{j.k}\; t$$

from which the transition probability function $\mathbf{P}^{decl}$ can be derived by

$$\mathbf{P}^{decl}(s, a, t) \;=\; \sum_j \left\{ p : s \xrightarrow{a[p]}_j t \right\}.$$

Other authors, e.g. [JoSm90, Toft94], use multisets of transitions. However, the resulting semantics $\mathcal{O}[\![\mathcal{P}]\!]$ does not depend on the chosen way for defining the transition probability function $\mathbf{P}^{decl}$. ■

Remark 4.2.4 [Internal, external probabilistic choice and restriction] Our interpretation of restriction $s \setminus L$ differs from those in [GJS90, JoSm90, vGSST90] (where the syntax $s\lceil A$ instead of $s \setminus L$ where $L = Act \setminus A$ is used). In the rule for $s \setminus L$, they use

a *normalization factor* (the probability $\mathbf{P}^{decl}(s, Act \setminus L)$ for state s to perform an action $a \in Act \setminus L$). Their rule for the restriction operator leads to the equation

$$\mathbf{P}^{decl}(s \setminus L, a, t \setminus L) \;=\; \frac{\mathbf{P}^{decl}(s, a, t)}{\mathbf{P}^{decl}(s, Act \setminus L)}$$

(provided that $\mathbf{P}^{decl}(s, Act \setminus L) > 0$). Thus, they deal with the conditional probabilities for the $(Act \setminus L)$-labelled transitions of s under the assumption that s performs an action from $Act \setminus L$. In contrast to this, in our approach the value $\mathbf{P}^{decl}(s, Act \setminus L)$ represents the probability of deadlock in $s \setminus L$ that results from the restriction (but not from a deadlock in s). For example, for the statement

$$s = t \setminus \{a\} \quad \text{where} \quad t \;=\; \left[\tfrac{1}{2}\right] a.nil \;\oplus\; \left[\tfrac{1}{2}\right] b.nil$$

we deal with the transition probabilities $\mathbf{P}^{decl}(s, b, nil \setminus \{a\}) = \mathbf{P}^{decl}(s, 0, \mathbf{0}) = 1/2$ while $\mathbf{P}^{decl}(s, b, nil \setminus \{a\}) = 1$ in the approachs of [GJS90, JoSm90, vGSST90] that focus on an *external* probabilistic choice operator where the process randomly chooses one of the *events offered by environment*. Hence, for the statement t of above, if the environment offers a and b then a and b are chosen with equal probability while, for an environment that just offers b (but not a), the action b will be performed (with probability 1). In contrast to this, we assume an *internal* probabilistic choice operator where the probabilistic choices are resolved *independently on the environment*. Thus, in our approach, if just b is available while t is willing to perform a and b with equal probability then either b will be performed (if the randomized choice selects b) or a deadlock occurs (if a is selected), both with probability 1/2. ■

Beside the use of another probabilistic choice operator, several other variants of the operational semantics are possible and might be useful in certain applications.

- We do not distinguish between well-termination and deadlock. If one wants to consider *nil* as a well-terminated process then one can use an auxiliary statement (e.g. *exit*), a new action symbol (e.g. $\surd$) and the transition probabilities $\mathbf{P}^{decl}(nil, \surd, exit) = 1$ instead of $\mathbf{P}^{decl}(nil, 0, \mathbf{0}) = 1$. Then, the 0-labelled transitions to $\mathbf{0}$ (that might arise from the restriction operator or recursion) represent deadlock while the $\surd$-labelled transitions to *exit* represent well-termination.
- Another possible variant concerns the rule for the product. In our approach, a deadlock in $s_1 \times s_2$ occurs if it occurs in one of the components s_1 or s_2. Alternatively, one could allow the non-deadlocked component to perform further actions even if a deadlock has occurred in the other component. If 0-labelled and $\surd$-labelled transitions are used, one can use rules to specify that a deadlock in $s_1 \times s_2$ occurs iff it occurs in one of the components while $s_1 \times s_2$ behaves as s_1 if s_2 has well-terminated (i.e. has performed the action $\surd$) and vice versa.

For fixed declaration *decl*, we define the relations $\sim^{decl}$ and $\sqsubseteq^{decl}_{\text{sim}}$ for *PSCCS* statements as in the case of *PCCS* (see page 77). Then, $\sim^{decl}$ and $\sqsubseteq^{decl}_{\text{sim}}$ are congruences with respect to all operators of *PSCCS*. This result for bisimulation equivalence $\sim^{decl}$ was established by Jou & Smolka (cf. Lemma 4.1 in [JoSm90]). The congruence proof for $\sqsubseteq^{decl}_{\text{sim}}$ is an easy verification and omitted here.

4.3 *PLSCCS*: a lazy synchronous calculus

In this section we propose a new calculus, called *PLSCCS*, which arises from *PSCCS* by replacing the synchronous parallel composition $\times$ by a *lazy synchronous* parallel composition $\otimes$. In the lazy product $\mathcal{P}_1 \otimes \mathcal{P}_2$, the processes $\mathcal{P}_1$ and $\mathcal{P}_2$ are forced to synchronize on all visible actions while the internal actions τ are executed independently.

As in the case of *PCCS*, we assume a special action τ that denotes any internal or invisible computation. While τ does not play a distinguished role in the product $s_1 \times s_2$ of *PSCCS* (i.e. τ is treated as any other action), in the lazy product $s_1 \otimes s_2$, the components s_1 and s_2 may perform arbitrary many internal τ-steps *independently* before they *synchronize on some visible actions*. In other words, each step of $s_1 \otimes s_2$ is composed by sequences of steps of s_1 and s_2, where each of them starts with an arbitrary number of internal actions τ and ends up with a visible action. As in the case of *PSCCS*, the probabilistic choices of the components are supposed to be independent. Hence, the probabilities for the transitions of $s_1 \otimes s_2$ are given by the product of the individual probabilities where we deal with the cumulative effect of the τ-transitions.

Syntax of *PLSCCS*: *PLSCCS* statements are given by the grammar shown in Figure 4.8 (page 83). Here, $Z \in ProcVar$, $a \in Act$, $L \subseteq Act \setminus \{\tau\}$, I is a nonempty countable

$$s \quad ::= \quad nil \;\Big|\; Z \;\Big|\; a.s \;\Big|\; \sum_{i \in I} [p_i]s_i \;\Big|\; s_1 \otimes s_2 \;\Big|\; s \setminus L \;\Big|\; s[\ell]$$

Figure 4.8: Syntax of *PLSCCS* statements

indexing set, p_i are real numbers $p_i \in]0,1]$ with $\sum p_i = 1$ and $\ell : Act \to Act$ is a relabelling function with $\ell(\tau) = \tau$. $Stmt_{PLSCCS}$ (or shortly $Stmt$) denotes the collection of all *PLSCCS* statements. *PLSCCS* denotes the set of all *PLSCCS* programs, i.e. pairs $\mathcal{P} = \langle decl, s \rangle$ consisting of a declaration *decl* and a *PLSCCS* statement. The intended meanings of *nil*, the prefix operator $a.s$, the probabilistic choice operator $\sum [p_i]s_i$, the restriction operator $s \setminus L$ and the relabelling operator $s[\ell]$ are as in the case of *PSCCS* (see page 79). For the lazy product $s_1 \otimes s_2$, we assume a function

$$(Act \setminus \{\tau\}) \times (Act \setminus \{\tau\}) \to Act, \quad (\alpha, \beta) \mapsto \alpha * \beta.$$

where, as in the case of *PSCCS*, $\alpha * \beta$ stands for the result of the synchronization on the visible actions α and β. Note that $\alpha * \beta = \tau$ is possible. Each a-labelled transition of the lazy product $s_1 \otimes s_2$ is composed by *sequences of steps* of the components s_1 and s_2 that are labelled by strings of the form $\tau^*\alpha$ and $\tau^*\beta$ respectively such that a is the result of the synchronized execution of the visible actions α and β (i.e. $a = \alpha * \beta$). Thus, the probability of $s_1 \otimes s_2$ to move via the action a to $t_1 \otimes t_2$ is given by the sum over all probabilities $Prob(s_1, \tau^*\alpha, t_1) \cdot Prob(s_2, \tau^*\beta, t_2)$ where (α, β) ranges over all pairs (α, β) of visible actions such that $\alpha * \beta = a$. [9]

[9] Recall that $Prob^{decl}(s, \tau^*\alpha, t)$ is the probability for s to perform a sequence of internal actions followed by α ending up in the state t (cf. Section 3.3.1, page 50).

Operational semantics for *PLSCCS*: We supply *PLSCCS* with an operational semantics based on action-labelled fully probabilistic processes. As in the case of *PSCCS*, we fix a declaration $decl : ProcVar \to Stmt$ and define the transition probabilities $\mathbf{P}^{decl}(s, a, t)$ (where $s, t \in Stmt$ and $a \in Act$) with the help of a higher-order operator on the function space $Stmt \times Act \times Stmt \to [0, 1]$. For the definition of the semantics of the lazy product, we have to deal with the probabilities $Prob^{decl}(s, \tau^*\alpha, t)$ for s to move to t via a sequence of steps labelled by a string of $\tau^*\alpha$. For this, we deal with an operator on function pairs and define the pair $\langle \mathbf{P}^{decl}, \mathbf{Q}^{decl} \rangle$ as the least pair of functions

$$\mathbf{P}^{decl} : Stmt \times Act \times Stmt \to [0, 1]$$

and

$$\mathbf{Q}^{decl} : Stmt \times (Act \setminus \{\tau\}) \times Stmt \to [0, 1]$$

that satisfies the equations of Figure 4.9 on page 85.[10] We proceed as in the case of *PSCCS* and extend the so defined function $\mathbf{P}^{decl} : Stmt : Act \times Stmt \to [0, 1]$ to a function $Stmt_{\mathbf{0}} \times Act_0 \times Stmt_{\mathbf{0}} \to [0, 1]$ (also called $\mathbf{P}^{decl}$). For $s \in Stmt$, we put

$$\mathbf{P}^{decl}(s, 0, \mathbf{0}) \;=\; 1 \;-\; \sum_{a \in Act} \; \sum_{t \in Stmt} \mathbf{P}^{decl}(s, a, t)$$

and define $\mathbf{P}^{decl}(\cdot) = 0$ in all remaining cases.[11] The operational semantics assigns to each *PLSCCS* program $\mathcal{P} = \langle decl, s \rangle$ the action-labelled fully probabilistic process $\mathcal{O}[\![\mathcal{P}]\!] = (Stmt_{\mathbf{0}}, Act_0, \mathbf{P}^{decl}, s)$. Let $Prob^{decl}$ denote the probability measure in the action-labelled fully probabilistic system $(Stmt_{\mathbf{0}}, Act_0, \mathbf{P}^{decl})$.

Lemma 4.3.1 *For all $s, t \in Stmt$ and $\alpha \in Act$, $\mathbf{Q}^{decl}(s, \alpha, t) = Prob^{decl}(s, \tau^*\alpha, t)$.*[12]

Proof: easy verification. Uses structural induction on the syntax of s and Proposition 3.3.4 (page 49). ■

Corollary 4.3.2 *For all $s_1, s_2, t_1, t_2 \in Stmt$ and $a \in Act$,*

$$\mathbf{P}^{decl}(s_1 \otimes s_2, a, t_1 \otimes t_2) \;=\; \sum_{(\alpha,\beta) \in Syn_a} Prob^{decl}(s_1, \tau^*\alpha, t_1) \cdot Prob^{decl}(s_2, \tau^*\beta, t_2).$$

Proof: follows immediately by Lemma 4.3.1 (page 84). ■

Recall that $Prob^{decl}(s, \tau^*\alpha) = \sum_t Prob^{decl}(s, \tau^*\alpha, t)$ is the probability for s to perform a sequence of τ's followed by α (cf. Section 3.3.1, page 50).

Corollary 4.3.3 *For all $s_1, s_2 \in Stmt$,*

$$\mathbf{P}^{decl}(s_1 \otimes s_2, 0, \mathbf{0}) \;=\; 1 \;-\; \sum_{\substack{\alpha_1 \in Act \\ \alpha_1 \neq \tau}} \; \sum_{\substack{\alpha_2 \in Act \\ \alpha_2 \neq \tau}} Prob^{decl}(s_1, \tau^*\alpha_1) \cdot Prob^{decl}(s_2, \tau^*\alpha_2).$$

[10] As in the case of *PSCCS*, the existence of a least function pair satisfying the equations of Figure 4.9 can be derived with standard methods of domain-theory; see e.g. Remark 12.1.2 on page 309.

[11] Here, $Stmt_{\mathbf{0}} = Stmt \cup \{\mathbf{0}\}$ and $Act_0 = Act \cup \{0\}$ where the new action symbol 0 and the auxiliary statement $\mathbf{0}$ are interpreted as in the case of *PSCCS*. See the explanations on page 81.

[12] In the notations of Section 3.3.1 (page 50), $Prob^{decl}(s, \tau^*\alpha, t)$ is the probability for the program $\langle decl, s \rangle$ to behave as $\langle decl, t \rangle$ after performing a sequence of steps labelled by an element of $\tau^*\alpha$.

$$\mathbf{P}^{decl}(Z,a,t) = \mathbf{P}^{decl}(decl(Z),a,t), \quad \mathbf{P}^{decl}(a.s,a,s) = 1$$

$$\mathbf{P}^{decl}\left(\sum_{i\in I}[p_i]s_i,a,t\right) \;=\; \sum_{i\in I} p_i\cdot\mathbf{P}^{decl}(s_i,a,t)$$

$$\mathbf{P}^{decl}(s_1\otimes s_2,a,t_1\otimes t_2) \;=\; \sum_{(\alpha,\beta)\in Syn_a}\mathbf{Q}^{decl}(s_1,\alpha,t_1)\cdot\mathbf{Q}^{decl}(s_2,\beta,t_2)$$

$$\mathbf{P}^{decl}(s\setminus L,a,t\setminus L) = \mathbf{P}^{decl}(s,a,t) \;\text{ if } a\notin L$$

$$\mathbf{P}^{decl}(s[\ell],a,t[\ell]) \;=\; \sum_{b\in\ell^{-1}(a)}\mathbf{P}^{decl}(s,b,t)$$

$$\mathbf{Q}^{decl}(s,\alpha,t) \;=\; \mathbf{P}^{decl}(s,\alpha,t) \;+\; \sum_{u\in Stmt}\mathbf{P}^{decl}(s,\tau,u)\cdot\mathbf{Q}^{decl}(u,\alpha,t).$$

Figure 4.9: Equations for $\mathbf{P}^{decl}$ and $\mathbf{Q}^{decl}$ in the case of *PLSCCS*

Proof: follows immediately by Corollary 4.3.2 (page 84). ■

Example 4.3.4 We consider the *PLSCCS* programs $\mathcal{P}_1 = \langle decl, s_1\rangle$, $\mathcal{P}_2 = \langle decl, s_2\rangle$ where *decl* is an arbitrary declaration,

$$s_1 \;=\; \left[\tfrac{1}{2}\right]\tau.(nil\setminus L) \;\oplus\; \left[\tfrac{1}{2}\right]\alpha.nil \;\text{ and }\; s_2 = \tau.\alpha.nil$$

for some subset L of $Act\setminus\{\tau\}$. The operational semantics of $\mathcal{P}_1$ and $\mathcal{P}_2$ are shown in Figure 4.10 (page 86). We now investigate the lazy product $s_1\otimes s_2$ where we assume that $\alpha * \alpha = \alpha$. In the lazy product $s_1\otimes s_2$, s_2 first performs its internal step and then offers the synchronization on α while s_1 chooses randomly between the internal step and the synchronization on α, both with probability 1/2. In the former case, s_1 and s_2 cannot synchronize, because, after performing the internal transition, s_2 waits forever for the synchronization on α. In the latter case, s_1 is idle until s_2 offers the synchronization on α. Figure 4.11 (page 86) shows the operational semantics of the program $\langle decl, s_1\otimes s_2\rangle$. Clearly, we have $Prob^{decl}(s_1,\tau^*\alpha,nil) = \mathbf{P}^{decl}(s_1,\alpha,nil) = 1/2$ and $Prob^{decl}(s_2,\tau^*\alpha,nil) = 1$. Thus, $\mathbf{P}^{decl}(s_1\otimes s_2,\alpha,nil\otimes nil) = 1/2$. On the other hand,

$$1 \;-\; \sum_{\substack{\alpha_1\in Act\\ \alpha_1\neq\tau}}\;\sum_{\substack{\alpha_2\in Act\\ \alpha_2\neq\tau}} Prob^{decl}(s_1,\tau^*\alpha_1)\cdot Prob^{decl}(s_2,\tau^*\alpha_2)$$

$$=\; 1 \;-\; Prob^{decl}(s_1,\tau^*\alpha)\cdot Prob^{decl}(s_2,\tau^*\alpha) \;=\; 1-\frac{1}{2}\cdot 1 = \frac{1}{2}$$

which yields $\mathbf{P}^{decl}(s_1\otimes s_2,0,\mathbf{0}) = 1/2$. ■

Example 4.3.5 We consider the programs $\mathcal{Q}_1 = \langle decl, s_1\rangle$, $\mathcal{Q}_2 = \langle decl, s_2\rangle$ where

$$s_1 = Z,\; decl(Z) \;=\; \left[\tfrac{1}{3}\right]\tau.Z \oplus \left[\tfrac{1}{3}\right]\tau.\alpha.w \oplus \left[\tfrac{1}{3}\right]\beta.v,\; s_2 \;=\; \left[\tfrac{1}{4}\right]\tau.\gamma.t \oplus \left[\tfrac{3}{4}\right]\delta.u.$$

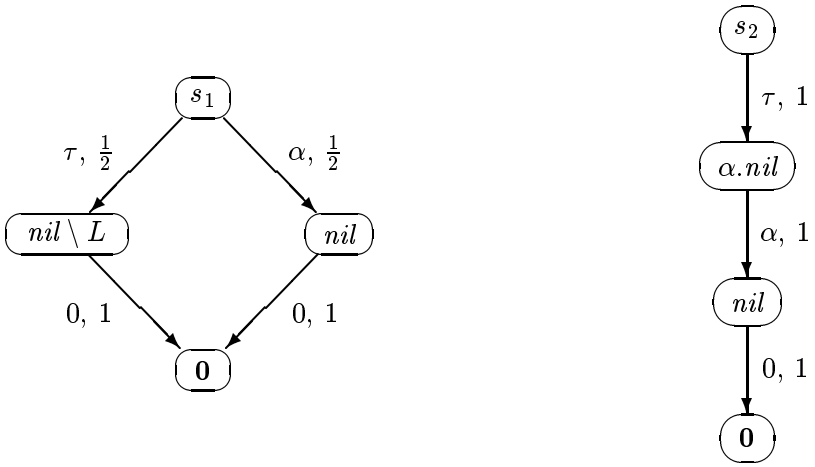


Figure 4.10: The operational semantics of *PLSCCS* programs $\mathcal{P}_1$ and $\mathcal{P}_2$

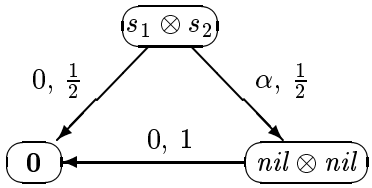


Figure 4.11:

Here, t, u, v, w are pairwise different statements. The operational semantics of $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are shown in in Figure 4.12 (page 87) where the outgoing transitions of t, u, v and w are omitted. We consider the lazy product $\mathcal{Q}_1 \otimes \mathcal{Q}_2$. We have

$$Prob^{decl}(s_1, \tau^*\alpha, w) = Prob^{decl}(s_1, \tau^*\beta, v) = 1/2$$

and $Prob^{decl}(s_2, \tau^*\gamma, t) = 1/4$, $Prob^{decl}(s_2, \tau^*\delta, u) = 3/4$. Thus,

$$\begin{array}{rcl} \mathbf{P}^{decl}(s_1 \otimes s_2, \alpha * \gamma, w \otimes t) & = & \mathbf{P}^{decl}(s_1 \otimes s_2, \beta * \gamma, v \otimes t) \;=\; \frac{1}{8}, \\ \mathbf{P}^{decl}(s_1 \otimes s_2, \alpha * \delta, w \otimes u) & = & \mathbf{P}^{decl}(s_1 \otimes s_2, \beta * \delta, v \otimes u) \;=\; \frac{3}{8}. \blacksquare \end{array}$$

In some applications, it might be helpful to work with a special visible idle action, e.g. called *wait*, by which a process can be forced to be idle in the next time step. Formally, we require that *wait* is a visible action such that $wait * \alpha = \alpha * wait = \alpha$ for all $\alpha \in Act \setminus \{\tau\}$. For example, $\alpha.s \otimes wait.t$ first performs α and then behaves as $s \otimes t$. I.e. the process $wait.t$ does not influence the first step, even though formally, it participates by performing the action *wait*.

Example 4.3.6 [The communication protocol *Sender* $\otimes$ *Receiver*] We consider a variant of the simple communication protocol of Example 1.2.2 on page 20 which we specify as the *lazy product* of a sender (who tries to send messages) and a receiver (who waits for the messages by the sender). The sender works with an uncertain medium that might lose message (with probability 0.01). If the message gets lost then the sender retries to deliver the message. In case where the message is delivered correctly, the sender waits for an acknowledgement of the receipt. For simplicity, we assume that the

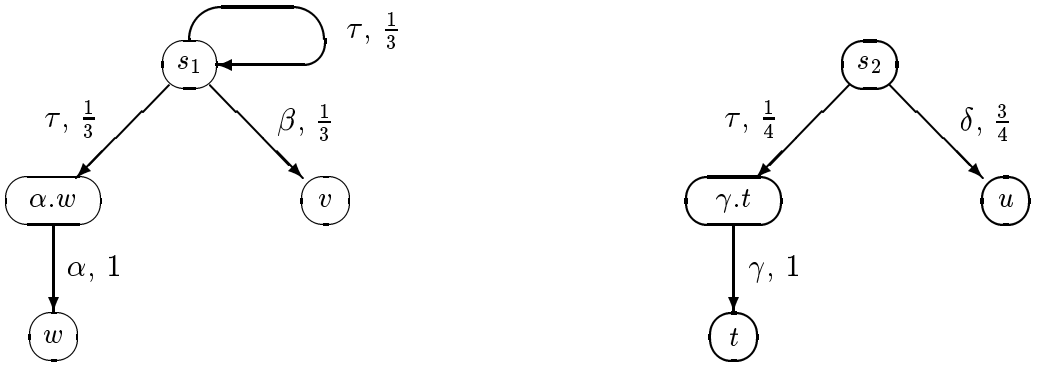


Figure 4.12: The operational semantics of *PLSCCS* programs $\mathcal{Q}_1$ and $\mathcal{Q}_2$

acknowledgement is transmitted by a safe medium that does not loose messages. The behaviour of the sender can be specified using process equations as explained on page 73.

$$
\begin{aligned}
&\mathit{Sender} \stackrel{\text{def}}{=} \mathit{produce}.\mathit{Try\_to\_send}\\
&\mathit{Try\_to\_send} \stackrel{\text{def}}{=} [0.01]\mathit{Lost} \ \oplus \ [0.99]\mathit{Deliver}\\
&\mathit{Lost} \stackrel{\text{def}}{=} \tau.\mathit{Try\_to\_send}\\
&\mathit{Deliver} \stackrel{\text{def}}{=} \mathit{deliver}!.\mathit{Wait\_for\_response}\\
&\mathit{Wait\_for\_response} \stackrel{\text{def}}{=} \mathit{wait}.\mathit{ack}?.\mathit{Sender}
\end{aligned}
$$

We use the visible actions *produce* (which means the action by which the sender generates a message), *deliver*! (the output action by which the medium transmits the message to the receiver), *ack*? (an input action that denotes that the sender sender reads the acknowledgement) and the action *wait* that is used to force the sender to be idle in the step where the receiver works up the message. The invisible action τ is used to describe the activities that are needed for preparing the next attempt to deliver the message. The operational semantics of the sender is shown in Figure 4.13 (page 87.) The receiver is

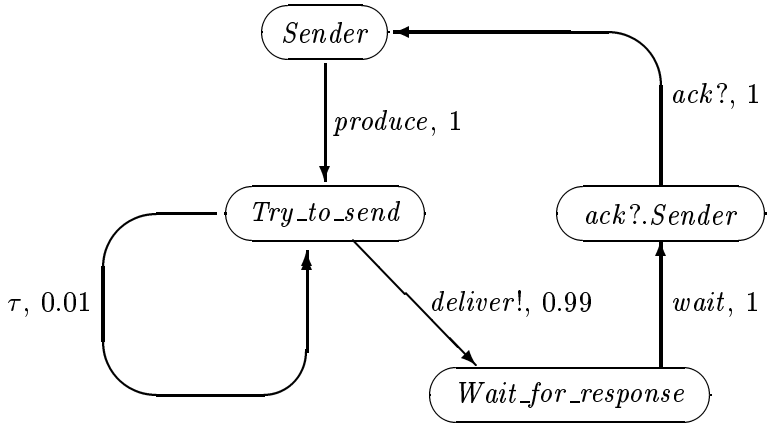


Figure 4.13: The operational semantics of the sender

specified as follows.

$$\begin{aligned} &\mathit{Receiver} \stackrel{\text{def}}{=} \mathit{wait}.\mathit{Get\_message} \\ &\mathit{Get\_message} \stackrel{\text{def}}{=} \mathit{deliver}?.\mathit{consume}.\mathit{Acknowledge} \\ &\mathit{Acknowledge} \stackrel{\text{def}}{=} \mathit{ack}!.\mathit{Receiver} \end{aligned}$$

We use the actions *deliver*? (the input action that stands for the receipt of the message), *consume* (an action by which the receiver works up the message and produces the acknowledgement), *ack*! (the output action by which the receiver acknowledges the receipt of the message) and the action *wait* (which ensures that the receiver is idle while the sender generates the next message). We suppose that $\mathit{deliver}! * \mathit{deliver}? = \mathit{ack}? * \mathit{ack}! = \tau$. The operational semantics of $\mathit{Sender} \otimes \mathit{Receiver}$ is shown in Figure 4.14 (page 88.)[13] ■

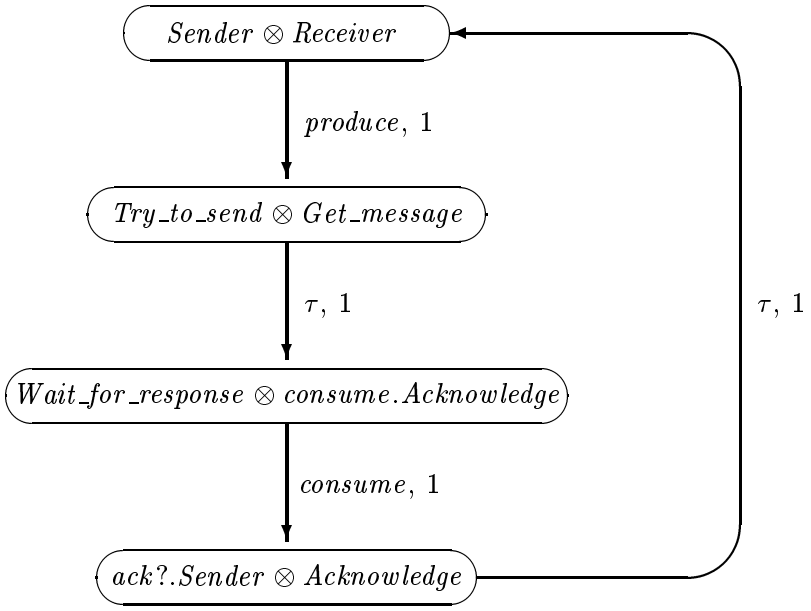


Figure 4.14: The operational semantics of $\mathit{Sender} \otimes \mathit{Receiver}$

As in the cases of *PCCS* or *PSCCS*, we adapt bisimulation equivalence $\sim$ and the simulation preorder $\sqsubseteq_{\text{sim}}$ for *PLSCCS* programs and statements where, for $\theta \in \{\sim, \sqsubseteq_{\text{sim}}\}$, we define $\mathcal{P}\ \theta\ \mathcal{P}'$ iff $\mathcal{O}[\![\mathcal{P}]\!]\ \theta\ \mathcal{O}[\![\mathcal{P}']\!]$ and $s\ \theta^{decl}\ s'$ iff $\langle decl, s\rangle\ \theta^{decl}\ \langle decl, s'\rangle$. It is easy to see that $s_1\ \theta^{decl}\ s_1'$, $s_2\ \theta^{decl}\ s_2'$ implies $s_1 \otimes s_2\ \theta^{decl}\ s_1' \otimes s_2'$. Thus, bisimulation equivalence and the simulation preorder are congruences for *PLSCCS*. In Chapter 7 we define weak bisimulation equivalence for action-labelled fully probabilistic systems for which we show that it preserves all operators of *PLSCCS* (except the probabilistic choice operator). This algebraic property is especially useful, since it allows one to replace components by equivalent ones that are minimized with respect to their internal behaviour.

[13] Note that the probability for the sender to reach the state *Wait_for_response* from *Try_to_send* via a path labelled by a trace of $\tau^* \mathit{deliver}!$ is 1.

Chapter 5

Denotational models

The recent trend in the semantics of programming languages is to provide a programming language with several (pairwise "consistent") semantics that describe different views, e.g. an operational, a denotational and a logical based semantics. While the operational semantics focusses on the stepwise behaviour, the denotational approach is based on *compositionality* (i.e. the existence of semantic operators for modelling the syntactic constructs of the language such as non-deterministic choice $+$, sequential composition ; or parallel composition $\|$). Another characteristic features of denotational semantics for programming languages with recursion or repetition is a the use of *fixed point equations* for the definition of the meaning of recursive or repetitive programs. Typically, these fixed point equations are solved with the help of Tarski's or Banach's fixed point theorems in which cases the semantic domain is supposed to be equipped with an appropriate *partial order* or *metric*. Thus, denotational semantics, being compositional, provide the theory that underpins system decomposition; and, if fully abstract, i.e., if the inherent order (in the partial order setting) or equality (in the metric setting) in the model precisely corresponds to the operational (pre)order or equivalence, the denotational semantics can provide additional insight into the nature of operational notions, and eventually serve as an intermediate link between the operational semantics and an appropriate logic.

Several authors proposed denotational semantics for probabilistic process calculi (see Section 1.2.2, page 23), but only a few of them investigate the issue of *full abstraction* with respect to an operational notion of "process equality". In the context of probabilistic process calculi with recursion, denotational models and related full abstraction results are presented for testing [Chri90a, Chri90b, Norm97, KwNo98a, KwNo98b] and failure [MMS+94] equivalence. To handle recursive processes, Morgan et al [MMS+94] use the standard partial order approach for establishing denotational least fixed point semantics. Christoff [Chri90a, Chri90b] deals with a variant of acceptance trees à la Hennessy [Henn88] and models recursion by equations as labellings for the branches in these acceptance trees. Kwiatkowska & Norman [KwNo96, Norm97, KwNo98a, KwNo98b] use a variant of the standard metric denotational approach and define the semantics as the limit of a Cauchy sequence in a complete metric space.

Based on the joint work with Marta Kwiatkowska [BaKw97], this chapter presents a method for providing denotational semantics for probabilistic calculi like *PCCS* or *PSCCS* that are *fully abstract* with respect to *bisimulation* and *simulation*. Full abstraction of

a denotational semantics $\mathcal{D}$ with respect to bisimulation means that $\mathcal{D}$ identifies exactly those programs that are bisimilar, i.e. $\mathcal{D}[\![\mathcal{P}]\!] = \mathcal{D}[\![\mathcal{P}']\!]$ iff $\mathcal{P} \sim \mathcal{P}'$ while full abstraction with respect to simulation means that the semantic domain (the range of $\mathcal{D}$) is equipped with an order $\sqsubseteq$ that reflects the simulation preorder, i.e. $\mathcal{D}[\![\mathcal{P}]\!] \sqsubseteq \mathcal{D}[\![\mathcal{P}']\!]$ iff $\mathcal{P} \sqsubseteq_{\text{sim}} \mathcal{P}'$. The partial order approach is used to obtain a fully abstract denotational semantics with respect to simulation; the metric setting to obtain full abstraction with respect to bisimulation.

As in the non-probabilistic case (e.g. [dBaZu82, GoRo83, dBaMe88, Abra91, RuTu93, Bai97]) the semantic domains $\mathbb{D}$ and $\mathbb{M}$ of the partial order and metric semantics respectively are obtained by applying standard categorical methods for solving *recursive domain equations*.[1] The main idea of the fully probabilistic case is to identify each process $\mathcal{P}$ with a *distribution* μ on pairs $(a, \mathcal{Q})$ where a is an action label, $\mathcal{Q}$ is a process and $\mu(a, \mathcal{Q})$ the probability for $\mathcal{P}$ to perform the action a and to behave as $\mathcal{Q}$ afterwards. This leads to recursive equations of the form

$$X \quad \cong \quad \{\mathbf{0}\} \ \cup \ \mathit{Distr}(\mathit{Act} \times X)$$

for the semantic domain X.[2] Here, $\mathbf{0}$ is a special symbol to denote inaction (i.e. a process like *nil* that does not perform any action). The central idea in the concurrent case is to represent a process $\mathcal{P}$ by a *set* of pairs (a, μ) consisting of an action a and a distribution μ on processes where each element (a, μ) of that set represents a non-deterministic alternative. From this, we obtain domain equations of the form

$$X \quad \cong \quad \mathit{Pow}_*(\mathit{Act} \times \mathit{Distr}(X))$$

where $\mathit{Pow}_*(\cdot)$ denotes a suitable powerdomain construction and where inaction is modelled e.g. by $\emptyset$. Unfortunately, in both cases the equation cannot be solved with the standard methods of [SmPl82, AbJu94] or [AmRu89, MaZe91, RuTu93] for solving recursive domain equations in the partial order or metric approach respectively since the distribution operator $X \mapsto \mathit{Distr}(X)$ fails the necessary condition of preserving completeness.[3] Nevertheless, the equations $X \cong \{\mathbf{0}\} \cup \mathit{Distr}(\mathit{Act} \times X)$ and $X \cong \mathit{Pow}_{\mathit{fin}}(\mathit{Act} \times \mathit{Distr}(X))$ have final solutions in *SET*, the category of sets and functions, which yields *final semantics* in the sense of Rutten & Turi [RuTu93] that are fully abstract with respect to bisimulation.[4]

In order to obtain fully abstract denotational models that can serve as semantic domains for providing denotational semantics in the metric or partial order framework we switch from $\mathit{Distr}(\cdot)$ to the probabilistic powerdomain $\mathit{Eval}(\cdot)$ of *evaluations* in the sense of Jones & Plotkin [JoPl89] (cf. Section 12.1.4, page 313). While distributions assign probabilities to *elements*, the evaluations decorate *sets* with "probabilities" (values in [0,1]). In our cases, where the underlying domain is a metric space or partial order, the set $\mathit{Eval}(\cdot)$ of all evaluations on $(\cdot)$ covers $\mathit{Distr}(\cdot)$. We solve domain equations of the form

$$X \ \cong \ \{\mathbf{0}\} \cup \mathit{Eval}(\mathit{Act} \times X) \ \text{ and } \ X \ \cong \ \mathit{Pow}_*(\mathit{Act} \times \mathit{Eval}(X))$$

[1] The recursive domain equations reflect the coinductive nature of bisimulation and simulation.
[2] Recall that $\mathit{Distr}(\cdot)$ denotes the set of distributions on $(\cdot)$. See Section 2.2 (page 30 ff).
[3] See Remark 5.1.13 (page 95) and Remark 5.1.18 (page 97).
[4] Here, $\mathit{Pow}_{\mathit{fin}}(\cdot)$ denotes the collection of finite subsets of $(\cdot)$.

where we apply the method of Abramsky & Jung [AbJu94] in the partial order approach and the method of Rutten & Turi [RuTu93] in the metric setting. The resulting domains $\mathbb{D}$ and $\mathbb{M}$ are shown to be *internally fully abstract* with respect to (bi-)simulation which means that the inherent order on $\mathbb{D}$ agrees with the simulation preorder and that bisimulation equivalence coincides with the equality on $\mathbb{M}$. Using the standard procedures to give denotational semantics in the partial order and metric approach we obtain denotational semantics on $\mathbb{D}$ and $\mathbb{M}$ and the desired full abstraction results.

Organization of that chapter: In Section 5.1 we consider the (more interesting and complicate) case of concurrent probabilistic processes in detail and the language *PCCS* (see Section 4.1, page 74 ff) where we shrink our attention to finitely branching systems.[5] The results for the fully probabilistic case and *PSCCS* (see Section 4.2, page 79 ff) are summarized in Section 5.2.

In this chapter, the reader is supposed to be familiar with the basic concepts of denotational (least fixed point or metric) semantics and categorical methods for solving recursive domain equations. The mathematical preliminaries that are needed in this chapter can be found in the appendix (Sections 12.1.1, 12.1.2, 12.1.3 and 12.1.4; see page 307 ff). Moreover, the reader should recall the notations that we use for distributions and weight functions (Section 2.2, page 30).

5.1 Denotational models: concurrent case

We take as basis finitely branching action-labelled concurrent probabilistic processes together with the bisimulation equivalence (Definition 3.4.3, page 54) and the simulation preorder (Definition 3.4.9, page 56). First, we turn our attention to the construction of semantic domains which can serve as fully abstract denotational models for languages with non-determinism, probabilistic choice and recursion (such as *PCCS*). We start with the equation $X \cong Pow_*(Act \times Distr(X))$ that we solve in *SET* and that yields "final semantics" in the sense of [RuTu93] (see Section 5.1.1). Then, taking the domain equations for non-probabilistic processes as basis, we derive domain equations involving the probabilistic powerdomain of evaluations which – when solved respectively in appropriate categories of partially ordered sets or metric spaces – give rise to semantic domains for probabilistic processes that are internally fully abstract with respect to simulation and bisimulation respectively (see Sections 5.1.2 and 5.1.3). Having obtained these internally fully abstract semantic domains, we use the standard procedures for establishing denotational semantics on dcpo's and complete metric spaces and obtain dentational semantics for *PCCS* that are shown to be fully abstract with respect to bisimulation and simulation respectively (see Section 5.1.4).

Simplified notations: In the remainder of this section we deal with *finitely branching action-labelled* concurrent probabilistic systems or processes with action labels of a fixed finite nonempty set *Act*. For simplicicity, we briefly speak about probabilistic systems or probabilistic processes rather than finitely branching action-labelled concurrent systems/processes of the form $(S, Act, Steps)$ or $(S, Act, Steps, s)$.

[5] Note that the operational semantics for *PCCS* (which does not allow for unbounded non-deterministic choice) always yields a finitely branching process.

5.1.1 The domain $I\!P$

In the non-probabilistic case, the final solution of the equation $X \cong Pow_{fin}(Act \times X)$ in SET yields a characterization of "action-labelled trees" [Barr93, RuTu93, Bai97]. These can be viewed as canonical representatives of the bisimulation equivalence classes of (non-probabilistic) finitely branching labelled transition systems with action labels in Act. Here, $Pow_{fin}(\cdot)$ denotes the collection of finite subsets of $(\cdot)$. We adapt this idea for the probabilistic case and show that the bisimulation equivalence classes of probabilistic processes form the final solution of the domain equation $X \cong Pow_{fin}(Act \times Distr(X))$.

Notation 5.1.1 [The domain $I\!P$] *Let $I\!P$ be the set of bisimulation equivalence classes of probabilistic processes.*[6]

We use symbols like $\mathcal{T}, \mathcal{T}', \mathcal{T}_1, \mathcal{T}_2, \ldots$ to range ober the elements of $I\!P$.

Notation 5.1.2 [The bisimulation equivalence classes $[\![\mathcal{P}]\!]$] *For $\mathcal{P}$ to be a probabilistic process, $[\![\mathcal{P}]\!]$ denotes the bisimulation equivalence class of $\mathcal{P}$.*

Notation 5.1.3 [The process $\mathcal{P}_t$] *Let $\mathcal{P} = (S, Act, Steps, s)$ be a probabilistic process and $t \in S$. Then, $\mathcal{P}_t$ denotes the probabilistic process $(S, Act, Steps, t)$.*

Notation 5.1.4 [The elements $\mathcal{T}_A$] *Let $\mathcal{P} = (S, Act, Steps, s)$ be a probabilistic process and $A \in S/\sim$. Let $\mathcal{T}_A$ be the unique element of $I\!P$ with $\mathcal{T}_A = [\![\mathcal{P}_t]\!]$ for all (some) $t \in A$.*

We associate with $I\!P$ the probabilistic system $\mathcal{S}(I\!P) = (I\!P, Act, Steps^{I\!P})$ where

$$Steps^{I\!P}([\![\mathcal{P}]\!]) \;=\; \{(a, Distr(f)(\mu)) : (a, \mu) \in Steps(s)\}.$$

where $f : S \to I\!P$ is given by $f(t) = [\![\mathcal{P}_t]\!]$. Hence, if $\mathcal{T}$ is the bisimulation equivalence class of $\mathcal{P} = (S, Act, Steps, s)$ (i.e. $\mathcal{T} = [\![\mathcal{P}]\!]$) then $\mathcal{T} \stackrel{a}{\longrightarrow} \nu$ iff there exists a transition $s \stackrel{a}{\longrightarrow} \mu$ with $\nu(\mathcal{T}_A) = \mu[A]$ for all bisimulation equivalence classes $A \in S/\sim$. In the sequel, each element $\mathcal{T}$ of $I\!P$ is identified with the probabilistic process $(I\!P, Act, Steps, \mathcal{T})$. The following lemma and its corollary show that for each probabilistic process $\mathcal{P}$, $[\![\mathcal{P}]\!]$ (viewed as a probabilistic process) is the unique element of $I\!P$ which is bisimilar to $\mathcal{P}$.

Lemma 5.1.5 *Let $\mathcal{P}$, $\mathcal{P}'$ be probabilistic processes.*

(a) $\mathcal{P} \;\sim\; [\![\mathcal{P}]\!]$
(b) $\mathcal{P} \sim \mathcal{P}'$ *if and only if* $[\![\mathcal{P}]\!] \;=\; [\![\mathcal{P}']\!]$
(c) $\mathcal{P} \sqsubseteq_{\rm sim} \mathcal{P}'$ *if and only if* $[\![\mathcal{P}]\!] \;\sqsubseteq_{\rm sim}\; [\![\mathcal{P}']\!]$

Proof: (b) is clear since $[\![\cdot]\!]$ is defined to be the bisimulation equivalence class of $(\cdot)$. (c) follows by (a) and part (c) of Lemma 3.4.14 (page 59). We show (a). Let

[6] In order to see that $I\!P$ is really a set consider a fixed set $States$ of cardinality ω and define $I\!P$ to be the set of bisimulation equivalence classes of probabilistic processes whose states belong to $States$. Then, each probabilistic process is bisimilar to some probabilistic process whose states belong to $States$. I.e. $I\!P$ represents all bisimulation classes of probabilistic processes. Note that the set of states s which can be reached from the initial state is always countable. Recall that we assume a fixed finite set Act of actions and that we only consider finitely branching processes.

$\mathcal{P} = (S, Act, Steps, s)$ and $R = \{(t, [\![\mathcal{P}_t]\!]) : t \in S\}$. We show that R fulfills the conditions of Proposition 3.4.4 (page 54).[7] Let $f : S \to I\!P$ be as before (i.e. $f(t) = [\![\mathcal{P}_t]\!]$). It is easy to see that

- If $t \xrightarrow{a} \mu$ then $[\![\mathcal{P}_t]\!] \xrightarrow{\alpha} Distr(f)(\mu)$.
- If $[\![\mathcal{P}_t]\!] \xrightarrow{a} \nu$ then $\nu = Distr(f)(\mu)$ for some transition $s \xrightarrow{a} \mu$.

By Remark 2.2.3 (page 31), $\mu \preceq_R Distr(f)(\mu)$. By Proposition 3.4.4 (page 54), $\mathcal{P}_t \sim f(t) = [\![\mathcal{P}_t]\!]$ for all $t \in S$. In particular, with $t = s$, we obtain $\mathcal{P} \sim [\![\mathcal{P}]\!]$. ■

Corollary 5.1.6 *$I\!P$ is internally fully abstract with respect to bisimulation, i.e. for all $\mathcal{T}, \mathcal{T}' \in I\!P$: $\mathcal{T} \sim \mathcal{T}'$ iff $\mathcal{T} = \mathcal{T}'$.*

Proof: Let $\mathcal{P}$, $\mathcal{P}'$ be probabilistic process with $\mathcal{T} = [\![\mathcal{P}]\!]$, $\mathcal{T}' = [\![\mathcal{P}']\!]$. Then, by part (a) of Lemma 5.1.5, $\mathcal{T} \sim \mathcal{P}$ and $\mathcal{T}' \sim \mathcal{P}'$. Hence, $\mathcal{T} \sim \mathcal{T}'$ implies $\mathcal{P} \sim \mathcal{P}'$, and therefore $\mathcal{T} = [\![\mathcal{P}]\!] = [\![\mathcal{P}']\!] = \mathcal{T}'$. ■

Next we show that $I\!P$ is a final solution of the equation $X \cong Pow_{fin}(Act \times Distr(X))$ in SET.

Theorem 5.1.7 *$I\!P$ is the final coalgebra (and hence the final fixed point) of the functor $Pow_{fin} \circ \mathcal{F}_{Act} \circ Distr : SET \to SET$.*[8]

Proof: Let $\mathcal{F} = Pow_{fin} \circ \mathcal{F}_{Act} \circ Distr$. We define $e : I\!P \to \mathcal{F}(I\!P)$ by $e(\mathcal{T}) = \{(a, \mu) : \mathcal{T} \xrightarrow{a} \mu\}$. Then, $(I\!P, e)$ is a coalgebra of $\mathcal{F}$. Let (Y, f) be a coalgebra of $\mathcal{K}$, i.e. $f : Y \to \mathcal{F}(Y)$ is a function. We associate with Y a probabilistic system $(Y, Act, Steps)$ where we put $Steps(y) = f(y)$. Hence, $y \xrightarrow{a} \nu$ iff $(a, \nu) \in f(y)$. It is easy to see that the function $F : Y \to I\!P$, $F(y) = [\![\mathcal{P}_y]\!]$ satisfies $\mathcal{F}(F) \circ f = e \circ F$. We show the uniqueness of F as a function $Y \to I\!P$ satisfying $\mathcal{F}(F) \circ f = e \circ F$. Whenever $F' : Y \to I\!P$ is a function with $\mathcal{F}(F') \circ f = e \circ F'$ then we show that $R = \{(y, F'(y)) : y \in Y\}$ satisfies the conditions of Proposition 3.4.4 (page 54).[9] It is easy to see that

- If $y \xrightarrow{a} \nu$ then $F'(y) \xrightarrow{a} Distr(F')(\nu)$.
- If $F'(y) \xrightarrow{a} \mu$ then $y \xrightarrow{a} \nu$ for some $\nu \in Distr(Y)$ where $\mu = Distr(F')(\nu)$.

By Remark 2.2.3 (page 31), $\nu \preceq_R Distr(F')(\nu)$. Thus, $\mathcal{P}_y \sim F'(y)$ for all $y \in Y$. By Lemma 5.1.5 and Corollary 5.1.6, $F'(y) = [\![\mathcal{P}_y]\!]$ for all $y \in Y$. Hence, $F = F'$. ■

Since $I\!P$ is the final coalgebra we get a *final semantics* in the sense of Rutten & Turi [RuTu93]. Let $(S, Act, Steps)$ be a probabilistic system. Then, (S, f) is a coalgebra of $\mathcal{F} = Pow_{fin} \circ \mathcal{F}_{Act} \circ Distr$ where $f : S \to \mathcal{F}(S)$ is given by $f(s) = \{(a, \nu) : s \xrightarrow{a} \nu\}$. The final semantics $F : S \to I\!P$ is defined as the unique function with $\mathcal{F}(F) \circ f = e \circ F$. As we saw in the proof of Theorem 5.1.7, $F(s) = [\![\mathcal{P}_s]\!]$ where $\mathcal{P}_s = (S, Act, Steps, s)$. By Lemma 5.1.5 (page 92), the final semantics is fully abstract in the sense that it identifies two states iff they are bisimilar and that it preserves the simulation preorder $\sqsubseteq_{\text{sim}}$.

[7] Here, R is viewed as a binary relation on the state space of the composed system (which is defined as described on page 61).

[8] Recall the definitions of the distribution functor $Distr : SET \to SET$ (page 312), the functor $\mathcal{F}_{Act} : SET \to SET$ which assigns to each set X the set $Act \times X$ (see page 312) and the functor $Pow_{fin} : SET \to SET$ which assigns to each set X the set $Pow_{fin}(X)$ of finite subsets of X (see page 312).

[9] Here, R is viewed as a binary relation on the state space of the composed system (which is defined as described on page 61).

Example 5.1.8 We consider the processes $\mathcal{P}$ and $\mathcal{P}'$ of Figure 3.6 on page 57. (I.e. $\mathcal{P}$ and $\mathcal{P}'$ are the processes with initial state s and s' respectively.) The final semantics $[\![\mathcal{P}]\!]$ and $[\![\mathcal{P}']\!]$ of $\mathcal{P}$ and $\mathcal{P}'$ (as elements of $I\!P = Pow_{fin}(Act \times Distr(I\!P))$) are

$$[\![\mathcal{P}]\!] = \{(a, \nu)\} \text{ and } [\![\mathcal{P}']\!] = \{(a, \nu')\}$$

where $\nu(\mathcal{T}) = 1/3$, $\nu(\emptyset) = 2/3$, $\nu'(\mathcal{T}) = \nu'(\emptyset) = 1/2$ and $\mathcal{T} = \{(b, \mu^1_\emptyset)\}$. ■

5.1.2 The semantic domain $I\!D$

We aim at solving a recursive domain equation of the form $D \cong Pow_*(Act \times Eval(D))$ in an appropriate category of dcpo's. The reason not to deal the equation $D \cong Pow_*(Act \times Distr(D))$ is that the distribution functor *Distr* does *not preserve completeness* of partially ordered sets (cf. Remark 5.1.13, page 95), and hence, fails for the standard methods for solving recursive domain equations for dcpo's. First we turn to the question which powerdomain $Pow_*(\cdot)$ should be used. We follow the ideas of the non-probabilistic case where the initial solution of the domain equation

$$D \cong Pow_{Hoare}(\{\bot\} \cup Act \times D)$$

yields a semantic domain that is internally fully abstract with respect to the simulation preorder [Bai97].[10] Note that the auxiliary element $\bot$ is needed as $Act \times D$ fails to be a dcpo (because it does not have a bottom element). Inaction (*nil*) is then modelled by the set $\{\bot\}$, the bottom element in $Pow_{Hoare}(\{\bot\} \cup Act \times D)$. In the probabilistic case, we adapt this idea and deal with the equation

$$D \cong Pow_{Hoare}(\{\bot\} \cup Act \times Eval(D)).$$

Recall the definitions of the category $CONT_\bot$ of continuous domains and strict and d-continuous functions (see pages 308 and 311) and of the locally d-continous functors $Eval : CONT_\bot \to CONT_\bot$ which assigns to each continuous domain D the powerdomain $Eval(D)$ of evaluations on D (see page 314), $Pow_{Hoare} : CONT_\bot \to CONT_\bot$ (see page 308) and $\mathcal{F}^{cont}_{Act} : CONT_\bot \to CONT_\bot$ which assigns to each continuous domain D the domain $\{\bot\} \cup Act \times D$ (see page 312). We solve the above equation in the category $CONT_\bot$ of continuous domains (alternatively, we could work with the larger category $DCPO_\bot$ of dcpo's and strict and d-continuous functions). For this, we have to show the local d-continuity of the associated functor $Pow_{Hoare} \circ \mathcal{F}^{cont}_{Act} \circ Eval$.

Lemma 5.1.9 *The functor* $\mathcal{F}_{\text{cont}} = Pow_{Hoare} \circ \mathcal{F}^{cont}_{Act} \circ Eval : CONT_\bot \to CONT_\bot$ *is locally d-continuous.*

Proof: follows from the local d-continuity of $Eval$, $\mathcal{F}^{cont}_{Act}$ and Pow_{Hoare}. ■

Notation 5.1.10 [The domain $I\!D$] *$I\!D$ denotes the initial fixed point of* $\mathcal{F}_{\text{cont}}$.[11]

In what follows, we deal with the isomorphism as an equality, i.e. if $(I\!D, j)$ is the initial fixed point of $\mathcal{F}_{\text{cont}}$ then we suppose $I\!D = \mathcal{F}_{\text{cont}}(I\!D)$ and $j = id_{I\!D}$. Note that the partial

[10] Here, $Pow_{Hoare}(\cdot)$ denotes the Hoare powerdomain (cf. Section 12.1.1, page 308).

[11] By the results of [AbJu94] (see page 311), $\mathcal{F}_{\text{cont}}$ has an initial fixed point.

order on $\mathbb{D}$ is the inclusion. The bottom element $\bot_{\mathbb{D}}$ in $\mathbb{D}$ is $\{\bot\}$ where $\bot$ denotes the bottom element in $\{\bot\} \cup Act \times Eval(\mathbb{D})$. If $(x_i)_{i\in I}$ is a directed family of elements in $\mathbb{D}$ then the least upper bound $\bigsqcup x_i$ is $(\bigcup x_i)^{cl}$, the Scott-closure of $\bigcup x_i$ in $\{\bot\} \cup Act \times Eval(\mathbb{D})$. If A, B are finite subsets of $\{\bot\} \cup Act \times Eval(\mathbb{D})$ then the Scott-closure $A^{cl} = A\downarrow$ and $A^{cl} \subseteq B^{cl}$ if and only if $A \sqsubseteq_L B$ where $\sqsubseteq_L$ denotes the lower preorder.[12]

The desired internal full abstraction result for $\mathbb{D}$ states that (in some sense) the inherent order on $\mathbb{D}$ (i.e. the inclusion) "reflects" the simulation preorder. For this, we show that $\mathbb{P}$ (together with the preorder $\sqsubseteq_{\rm sim}$) can be *embedded* into $\mathbb{D}$ via a function $\imath_{\mathbb{D}} : \mathbb{P} \to \mathbb{D}$ such that $\mathcal{T} \sqsubseteq_{\rm sim} \mathcal{T}'$ iff $\imath_{\mathbb{D}}(\mathcal{T}) \subseteq \imath_{\mathbb{D}}(\mathcal{T}')$. Thus, the subspace $\imath_{\mathbb{D}}(\mathbb{P})$ of $\mathbb{D}$ represents the *simulation equivalence classes* of probabilistic processes. The basic lemma for the proof of this full abstraction result is Theorem 5.1.12 which asserts a general connection between $Distr(D)$ and $Eval(D)$ that hold for any dcpo D: $Distr(D)$ equipped with the weight-function-based preorder $\preceq_{\rm sim}$ (as defined in Notation 5.1.11) is a subspace of $Eval(D)$; in particular, Theorem 5.1.12 yields that $\preceq_{\rm sim}$ is a partial order on $Distr(D)$.

Notation 5.1.11 [The partial order $\preceq_{\rm sim}$] *Let D be a partially ordered set (with partial order $\sqsubseteq$) and μ, $\mu' \in Distr(D)$. Then, $\mu \preceq_{\rm sim} \mu'$ iff there exists a weight function for (μ, μ') with respect to $\sqsubseteq$.*[13]

The following theorem shows that, for each dcpo D, the set $Distr(D)$ equipped with the order $\preceq_{\rm sim}$ can be viewed as a subspace of $Eval(D)$. More precisely, we show that the function $Distr(D) \to Eval(D)$, $\mu \mapsto E_\mu$, is order-preserving. Thus, each distribution μ can be identified with the evaluation E_μ.[14]

Theorem 5.1.12 *If D is a dcpo then $\preceq_{\rm sim}$ is a partial order on $Distr(D)$. Moreover, for all μ, $\mu' \in Distr(D)$, $\mu \preceq_{\rm sim} \mu'$ iff $E_\mu \sqsubseteq E_{\mu'}$.*

Proof: see Section 5.3, Theorem 5.3.2 (page 105) and Corollary 5.3.4 (page 109). ■

Remark 5.1.13 [Incompleteness of $Distr(D)$] In general, $Distr(D)$ is not complete. Consider the dcpo $D = \{0,1\}^\infty$ of all (finite or infinite) words built from 0 and 1 equipped with the prefix ordering. Let μ_k be the distribution with

$$\mu_k(x) = \begin{cases} 1/2^k & : \text{ if } x \text{ is a word of length } k \\ 0 & : \text{ otherwise.} \end{cases}$$

It is easy to see that $(\mu_k)_{k\geq 1}$ is a monotone sequence in $Distr(D)$ which does not have an upper bound in $Distr(D)$. In order to see that $\mu_k \sqsubseteq_{\rm sim} \mu_{k+1}$ consider the distribution *weight* on $D \times D$ with $weight(x, x0) = weight(x, x1) = 1/2^{k+1}$ if x is a word of length k and $weight(x, y) = 0$ in all other cases. ■

Using Theorem 5.1.12 we can show the following connection between $\mathbb{P}$ and $\mathbb{D}$.

Theorem 5.1.14 *There exists a unique function $\imath_{\mathbb{D}} : \mathbb{P} \to \mathbb{D}$ such that*

$$\imath_{\mathbb{D}}(\mathcal{T}) = \left\{(a, E_{Distr(\imath_{\mathbb{D}})(\mu)}) : \mathcal{T} \stackrel{a}{\longrightarrow} \mu\right\}^{cl}.$$

[12] Recall that that the lower preorder is given by $A \sqsubseteq_L B$ iff for all $x \in A$ there exists $y \in B$ such that $x \sqsubseteq y$. Here, $\sqsubseteq$ denotes the order on $\{\bot\} \cup Act \times Eval(\mathbb{D})$.

[13] Note that $\preceq_{\rm sim} = \preceq_{\sqsubseteq}$ (with the notations of Section 2.2, page 30).

[14] Recall that E_μ is given by $E_\mu(U) = \mu[U]$, see page 313.

Moreover, for all $\mathcal{T}$, $\mathcal{T}' \in I\!P$, $\mathcal{T} \sqsubseteq_{\mathrm{sim}} \mathcal{T}'$ *iff* $\imath_{I\!D}(\mathcal{T}) \subseteq \imath_{I\!D}(\mathcal{T}')$.

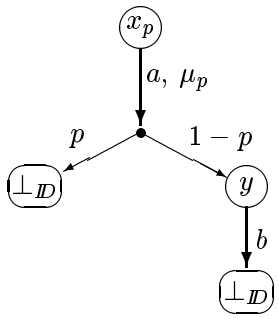
Proof: see Section 5.3, Theorem 5.3.10 (page 111). ■

The final semantics of Section 5.1.1 (page 93) yields a semantics on $I\!D$ which is fully abstract with respect to the simulation preorder in the following sense. If $\mathcal{P}$, $\mathcal{P}'$ are probabilistic processes then $\mathcal{P} \sqsubseteq_{\mathrm{sim}} \mathcal{P}'$ iff $\imath_{I\!D}([\![\mathcal{P}]\!]) \subseteq \imath_{I\!D}([\![\mathcal{P}']\!])$ (Lemma 5.1.5, page 92, and Theorem 5.1.14). Thus, the element $\imath_{I\!D}([\![\mathcal{P}]\!])$ can be considered as the simulation equivalence class of $\mathcal{P}$.

Example 5.1.15 We consider the processes $\mathcal{P}$ and $\mathcal{P}'$ of Example 5.1.8 (cf. Figure 3.6, page 57). $\mathcal{P}$ and $\mathcal{P}'$ are represented in $I\!D$ by

$$\begin{array}{rcccl} \imath_{I\!D}([\![\mathcal{P}]\!]) & = & \{(a, E_\nu)\}^{cl} & = & \{\bot\} \cup \{(a,E) : E \in \mathbf{E}_{\frac{2}{3}}\}, \\ \imath_{I\!D}([\![\mathcal{P}']\!]) & = & \{(a, E_{\nu'})\}^{cl} & = & \{\bot\} \cup \{(a,E) : E \in \mathbf{E}_{\frac{1}{2}}\}. \end{array}$$

Here, $\nu = \mu_{\frac{2}{3}}$, $\nu' = \mu_{\frac{1}{2}}$, $\mathbf{E}_p = \{E_{\mu_q} : p \leq q \leq 1\}$ where μ_p is is the unique distribution on $I\!D$ with $\mu_p(\bot_{I\!D}) = p$ and $\mu_p(\{(b, E^1_{\mu_{\bot_{I\!D}}})\}^{cl}) = 1-p$. The picture on the right shows the "maximal transitions" of $x_p = \{(a, E_{\mu_p})\}^{cl}$. For each $x \in I\!D$, each element $(a, E_\mu) \in x$ can be viewed as a "transition" $x \xrightarrow{a} \mu$. Maximality of a transition $x \xrightarrow{a} \mu$ means that, whenever $x \xrightarrow{a} \nu$ then $\nu \preceq_{\mathrm{sim}} \mu$. ■



We refer the interested reader to Section 5.3.2 (page 114 ff) where we investigate some domain-theoretic properties of the domain $I\!D$ and show the domain-theoretic differences between $I\!D$ and the corresponding domain for non-probabilistic processes.

5.1.3 The semantic domain $I\!M$

In the non-probabilistic case, a complete ultrametric space M that is internally fully abstract with respect to bisimulation is obtained by solving the recursive domain equation

$$M \cong Pow_{comp}(Act \times M_{\frac{1}{2}})$$

(see [dBaZu82, GoRo83, dBaMe88, RuTu93, Bai97]). The subscript $1/2$ denotes that the distance on M is multiplied with the factor $1/2$ and $Pow_{comp}(\cdot)$ denotes the collection of compact subsets of $(\cdot)$ (see page 311). We adapt this idea to the probabilistic case and solve the equation

$$M \cong Pow_{comp}(Act \times Eval(M)_{\frac{1}{2}}).$$

Recall the notations for ultrametric spaces (Section 12.1.2, page 310 ff) and the method by Rutten & Turi [RuTu93] for solving recursive domain equations in the category CUM of complete ultrametric spaces and non-expansive functions (Section 12.1.3, page 312). First, we show that the probabilistic powerdomain $Eval(\cdot)$ of evaluations can be considered as an endofunctor on CUM which is locally non-expansive in the sense of [RuTu93]. It is easy to see that for each distribution μ on M:

$$\mu(x) = \inf\{\ E_\mu(U)\ : x \in U \in Opens(M)\ \} = \inf\{\ E_\mu(B)\ : x \in B \in Balls(M)\ \}$$

where E_μ is given by $E_\mu(U) = \mu[U]$ (see page 313).[15] By the above, whenever μ, $\mu' \in Distr(M)$ with $E_\mu = E_{\mu'}$ then $\mu = \mu'$. Hence, the set $Distr(M)$ of distributions on M can be considered as a subspace of $Eval(M)$. We suppose $Eval(M)$ to be endowed with the distance

$$d(E_1, E_2) \;=\; \inf \{\; \rho > 0 \;:\; E_1(B) = E_2(B) \;\; \forall\, B \in Balls_\rho(M) \;\}\,.$$

Theorem 5.1.16 *If M is a complete ultrametric space then $Eval(M)$ is a complete ultrametric space. In this case, $Eval(M)$ is the completion of $Distr(M)$ (considered as a subspace of $Eval(M)$).*

Proof: see Section 5.3 Theorem 5.3.22 (page 120) and Theorem 5.3.23 (page 122). ■

The ultrametric d on $Eval(M)$ can be characterized as follows: $d(E_1, E_2) \leq r$ iff $E_1(B) = E_2(B)$ for all $B \in \bigcup_{\rho > r} Balls_\rho(M)$ iff $E_1(U) = E_2(U)$ for all ρ-sets U where $\rho > r$. Whenever $f : M \to M'$ is a non-expansive function between ultrametric spaces M and M' and B is an open ball in M' with radius ρ then $f^{-1}(B)$ is a ρ-set. From this, whenever E_1, E_2 are evaluations on $Eval(M)$ then

$$d(Eval(f)(E_1), Eval(f)(E_2)) \;\leq\; d(E_1, E_2),$$

i.e. $Eval(f)$ is non-expansive. Hence, $Eval$ can be considered as an endofunctor of CUM.

Lemma 5.1.17 *The functor $Eval : CUM \to CUM$ is locally non-expansive.*

Proof: easy verification. ■

Remark 5.1.18 [**Incompleteness of** $Distr(M)$] Similarly to Remark 5.1.13 (page 95) the set $\{0,1\}^\infty$ equipped with the natural distance

$$d(x, y) \;=\; \inf \left\{ \frac{1}{2^n} : x[n] \neq y[n] \right\}$$

(where $z[n]$ denotes the n-th prefix of z) yields an example for a complete metric space M where $Distr(M)$ is not complete (i.e. $Distr(M)$ as a subspace of $Eval(M)$ is not closed). Consider the sequence (μ_k) which is defined as in Remark 5.1.13 (page 95). Then, $d(\mu_k, \mu_i) \leq 1/2^i$ for all $k \geq i$. Thus, (μ_k) is a Cauchy sequence in $Distr(M)$ which does not have a limit in $Distr(M)$.[16] ■

Recall the definitions of the functors $Pow_{comp} : CUM \to CUM$ which assigns to each complete ultrametric space M the set $Pow_{comp}(M)$ of compact subsets of M (see page 312) and $\mathcal{F}^{cum}_{Act}$ with $\mathcal{F}^{cum}_{Act}(M) = Act \times M_{\frac{1}{2}}$ where the subscript $\frac{1}{2}$ means that the distance on M is multiplied with the factor $1/2$ (see page 312).

[15] Recall that for each ultrametric space M, we deal with the topology of open balls, i.e. $Opens(M)$ is the set of all subsets U of M such that, for each $x \in M$, there is some open ball B with $x \in B \subseteq U$ (see page 310).

[16] The limit of (μ_k) in $Eval(M)$ is the unique evaluation E on M such that for all open balls B, $E(B) = 1/2^n$ if $B = B(x, 1/2^n)$ for some finite word x of the length n, and $E(B) = 0$ in all other cases (i.e. those cases where $B = \{x\}$ for some finite word x). This evaluation E is not of the form E_μ for some $\mu \in Distr(M)$.

Lemma 5.1.19 *The functor* $\mathcal{F}_{\text{cum}} = Pow_{comp} \circ \mathcal{F}^{cum}_{Act} \circ Eval : CUM \to CUM$ *is locally contracting.*

Notation 5.1.20 [The domain $\mathbb{M}$] *Let $\mathbb{M}$ denote the unique fixed point of $\mathcal{F}_{\text{cum}}$.*[17]

As in the case of $\mathbb{D}$, we deal with the isomorphism as an equality, i.e. if $(\mathbb{M}, j)$ is the unique fixed point of $\mathcal{F}_{\text{cum}}$ then we suppose $\mathbb{M} = \mathcal{F}_{\text{cum}}(\mathbb{M}))$ and $j = id_{\mathbb{M}}$.

Theorem 5.1.21 *$\mathbb{P}$ is a dense subspace of $\mathbb{M}$. More precisely, there exists a unique function $\imath_{\mathbb{M}} : \mathbb{P} \to \mathbb{M}$ such that for all $\mathcal{T} \in \mathbb{P}$,*

$$\imath_{\mathbb{M}}(\mathcal{T}) \;=\; \{(a, E_{Distr(\imath_{\mathbb{M}})(\mu)}) : \mathcal{T} \xrightarrow{a} \mu\}.$$

This function $\imath_{\mathbb{M}}$ is injective and $\imath_{\mathbb{M}}(\mathbb{P})$ is a dense subspace of $\mathbb{M}$.

Proof: see Section 5.3, Theorem 5.3.27 (page 124). ■

By Lemma 5.1.5 (page 92) and Theorem 5.1.21, $\mathbb{M}$ is fully abstract with respect to bisimulation in the following sense. If $\mathcal{P}$, $\mathcal{P}'$ are probabilistic processes then $\mathcal{P} \sim \mathcal{P}'$ iff $\imath_{\mathbb{M}}([\![\mathcal{P}]\!]) = \imath_{\mathbb{M}}([\![\mathcal{P}']\!])$.

Example 5.1.22 We consider the processes $\mathcal{P}$ and $\mathcal{P}'$ of Example 5.1.8 (cf. Figure 3.6, page 57); see also Example 5.1.15, page 96. $\mathcal{P}$ and $\mathcal{P}'$ are represented in $\mathbb{M}$ by

$$\imath_{\mathbb{M}}([\![\mathcal{P}]\!]) = \{(a, E_\nu)\} \quad \text{and} \quad \imath_{\mathbb{M}}([\![\mathcal{P}']\!]) = \{(a, E_{\nu'})\}$$

where $\nu(\{(b, E_{\mu^1_\emptyset})\}) = 1/3$, $\nu(\emptyset) = 2/3$ and $\nu'(\{(b, E_{\mu^1_\emptyset})\}) = \nu'(\emptyset) = 1/2$. ■

Remark 5.1.23 [The "distance" between processes] Theorem 5.1.21 (page 98) yields a "distance" for probabilistic processes which generalizes the one that is obtained from the approach of deBakker & Zucker [dBaZu82] for non-probabilistic processes: For $\mathcal{P}$, $\mathcal{P}'$ to be probabilistic processes, the "distance" between $\mathcal{P}$ and $\mathcal{P}'$ is given by

$$d(\mathcal{P}, \mathcal{P}') \;=\; d_{\mathbb{M}}\left(\imath_{\mathbb{M}}([\![\mathcal{P}]\!]), \imath_{\mathbb{M}}([\![\mathcal{P}']\!])\right).$$

Roughly speaking, the distance between two processes $\mathcal{P}$ and $\mathcal{P}'$ is $1/2^n$ if n is the maximal number such that the n-cuts of the unwindings of $\mathcal{P}$ and $\mathcal{P}'$ are bisimilar. For instance, the processes $\mathcal{P}$ and $\mathcal{P}'$ on Figure 5.1 (page 99) have the distance $1/2$ as their 1-cuts are bisimilar.[18] Kwiatkowska & Norman [KwNo96, Norm97] deal with a different metric which is not based on n-cuts. For instance, in the approach of [KwNo96, Norm97] the processes $\mathcal{P}_n$ shown on the left of Figure 5.2 (page 99) "converge" to the process $\mathcal{P}$ (shown on the right of Figure 5.2) while in our setting the sequence $(\mathcal{P}_n)$ (more precisely, the induced sequence $(\imath_{\mathbb{M}}([\![\mathcal{P}_n]\!]))$ in $\mathbb{M}$) is not a Cauchy sequence. ■

[17] The existence of a unique fixed point of $\mathcal{F}_{\text{cum}}$ is ensured by the results of [RuTu93] (see page 312).

[18] Note that in the 1-cut, the states t, t', u and u' are viewed to be bisimilar as we ignore the b-labelled transitions.

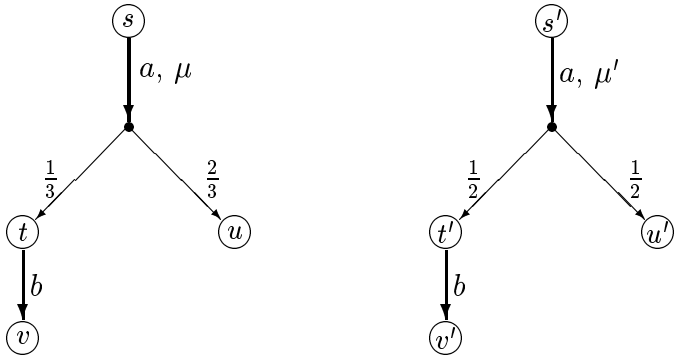


Figure 5.1: Two processes with distance 1/2

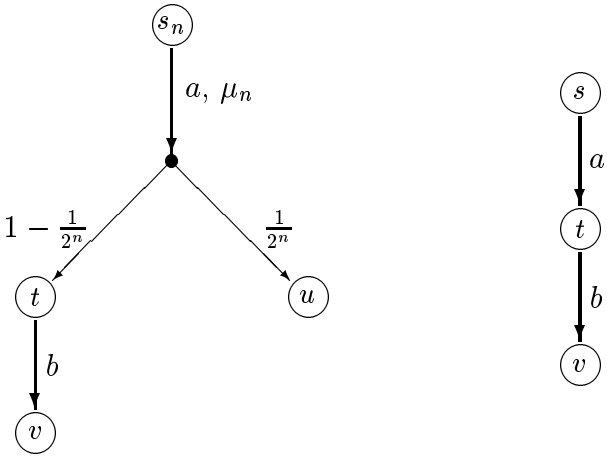


Figure 5.2: Two processes with distance 1

5.1.4 Denotational semantics on $\mathbb{M}$ and $\mathbb{D}$

This section shows how to establish denotational semantics for the process algebra *PCCS* (see Section 4.1, page 74 ff) on $\mathbb{M}$ and $\mathbb{D}$ and the desired full abstraction results.

Denotational semantics in the metric and partial order approach: We assume the reader to be familiar with the Scott-Strachey approach to establish denotational semantics in the partial order approach and the standard procedure to give denotational semantics in the metric approach [Stoy77, Niva79, dBaZu82]. Here, we only give a brief summary. We refer to [BMC94, dBdV96] for a full treatment. The domain X of a denotational semantics $\mathcal{D}$ for a process algebra *PA* (like *CCS* or a probabilistic calculus like *PCCS*) is equipped with a set *SemOp* of *semantic operators* that reflect the operators of *PA* in the following sense. If *op* is an n-ary operator symbol (like the binary operator symbol $+$ for modelling non-deterministic choice or the 1-ary (action-)prefix operator $s \mapsto a.s$) and op_X the corresponding semantic operator on X then

$$\mathcal{D}[\![\ op(\mathcal{P}_1, \ldots, \mathcal{P}_n)\ ]\!] \ = \ op_X\,(\mathcal{D}[\![\mathcal{P}_1]\!], \ldots, \mathcal{D}[\![\mathcal{P}_n]\!])$$

for all *PA* programs $\mathcal{P}_1, \ldots, \mathcal{P}_n$. Moreover, for any declaration *decl*, the meanings of a procedure name (process variable) Z and the body of the procedure $decl(Z)$ are the same. That is, for any fixed declaration *decl*, the function $s \mapsto \mathcal{D}[\![\langle decl, s\rangle]\!]$ is a *homomorphism* from the word algebra $(Stmt, Op)$ to $(X, SemOp)$ such that the meaning of process variable

Z is given by $decl(Z)$, i.e. $\mathcal{D}[\![\langle decl, Z\rangle]\!] = \mathcal{D}[\![\langle decl, decl(Z)\rangle]\!]$. It is known (see, for instance, [BMC94]) that function $\mathcal{D}$ satisfies these conditions iff, for any fixed declaration $decl$, the function $Stmt \to X$, $s \mapsto \mathcal{D}[\![\langle decl, s\rangle]\!]$, is a *fixed point* of the operator $F_{decl} : (Stmt \to X) \to (Stmt \to X)$, defined by

$$F_{decl}(f)(op(s_1, \ldots, s_n)) \;=\; op_X\,(F_{decl}(f)(s_1), \ldots, F_{decl}(f)(s_n))$$

for each operator op of PA and, for each process variable Z, $F_{decl}(f)(Z) = f(decl(Z))$. Given this function F_{decl}, the denotational semantics of PA – regardless of whether we follow the partial order or metric approach – is obtained by $\mathcal{D}[\![\langle decl, P\rangle]\!] = f_{decl}(P)$ where $f_{decl} : Stmt \to X$ is a certain fixed point of F_{decl}. In the partial order approach it is guaranteed – by Tarski's fixpoint theorem – that F_{decl} has a *least* fixed point, provided that F_{decl} is *d-continuous* on the function space $Stmt \to X$ (which is the case if all semantic operators are d-continuous). In the metric approach it is guaranteed – by Banach's fixpoint theorem – that F_{decl} has a *unique* fixed point, provided that F_{decl} is contracting. In order to guarantee that F_{decl} is contracting it is sufficient that the semantic operators are *non-expansive* and *contracting* in certain arguments. For the latter, one has to shrink the domain of $\mathcal{D}$ to *guarded programs*, i.e. those programs $\langle decl, s\rangle$ such that, for all procedures (process variables) Z, Z', each recursive procedure call of Z' in $decl(Z)$ is preceeded by at least one action.

Guarded $PCCS$: The formal definition of guardedness for the process algebra $PCCS$ is as follows. Guarded $PCCS$ statements are built from the following production system.

$$g \;::=\; nil \;\Big|\; a.\Big(\sum_{i\in I}[p_i]s_i\Big) \;\Big|\; g_1 + g_2 \;\Big|\; g_1 \parallel g_2 \;\Big|\; g \setminus L \;\Big|\; g[\ell]$$

where s_i are arbitrary $PCCS$ statements. A declaration $decl$ is called *guarded* iff $decl(Z)$ is guarded for all $Z \in ProcVar$. $GPCCS$ the subset of *guarded programs*, i.e. all programs $\langle decl, s\rangle$ where $decl$ is a guarded declaration.

Semantic operators on $\mathbb{D}$ and $\mathbb{M}$: We give a denotational semantics for $GPCCS$ on $\mathbb{M}$, which is fully abstract with respect to bisimulation equivalence, and a denotational semantics for $PCCS$ on $\mathbb{D}$, which is fully abstract with respect to the simulation preorder. For this, we need *non-expansive/contracting* semantic operators on $\mathbb{M}$ and *d-continuous* semantic operators on $\mathbb{D}$. In the sequel, $X = \mathbb{M}$ or $X = \mathbb{D}$. We use the closue notations for subsets of $Act \times Eval(\mathbb{M})$ and subsets of $\{\bot\} \cup Act \times Eval(\mathbb{D})$ where we put $A^{cl} = A$ if $A \subseteq Act \times Eval(\mathbb{M})$ and where, for $\emptyset \neq A \subseteq \{\bot\} \cup Act \times Eval(\mathbb{D})$, A^{cl} is the Scott-closure of A and $\emptyset^{cl} = \{\bot\}$.

- The process nil is modelled by $\emptyset$ in $\mathbb{M}$ and by $\bot_{\mathbb{D}} = \{\bot\}$ in $\mathbb{D}$.
- Nondeterministic choice on $\mathbb{M}$ and $\mathbb{D}$ is modelled by set-theoretic union.
- Action-guarded probabilistic choice: Let $a \in Act$ and $(p_i)_{i\in I}$ be a countable family of real numbers $p_i > 0$ with $\sum_{i\in I}\ p_i = 1$. Let $(x_i)_{i\in I}$ be a family in X. We put

$$a.\Big(\sum_{i\in I}[p_i]x_i\Big) = \{(a, E_\mu)\}^{cl} \;\text{ where } \mu \in Distr(X) \text{ is given by } \;\mu(x) \;=\; \sum_{\substack{i\in I\\ x_i = x}} p_i.$$

We define semantic operators for modelling restriction, relabelling and parallelism as fixed points of suitable operators. This reflects the recursive nature of restriction, relabelling and parallelism (cf. Milner's expansion law [Miln89] for parallelism).

- Restriction: Let $L \subseteq Act$ with $L = \overline{L}$. We define $F_L^X : (X \to X) \to (X \to X)$ by:

$$F_L^X(f)(x) \;=\; \{\ (a, Eval(f)(E))\ :\ (a,E) \in x,\ a \notin L\ \}^{cl}.$$

- Relabelling: Let ℓ be a relabelling function. $F_\ell^X : (X \to X) \to (X \to X)$ is given by

$$F_\ell^X(f)(x) \;=\; \{\ (\ell(a), Eval(f)(E))\ :\ (a,E) \in x\ \}^{cl}.$$

- Parallel composition: We use the following notations. If $f : X \times X \to X$ is a function and $x_0,\ y_0 \in X$ then we define $f(x_0,\cdot), f(\cdot,y_0) : X \to X$ by $f(x_0,\cdot)(y) = f(x_0,y)$ and $f(\cdot,y_0)(x) = f(x,y_0)$. We define $F_\parallel^X : (X \times X \to X) \to (X \times X \to X)$ by

$$F_\parallel^X(f)(x,y) \;=\; \Big(F_1^X(f)(x,y) \cup F_2^X(f)(x,y) \cup F_{\rm syn}^X(f)(x,y)\Big)^{cl}$$

  where
  $F_1^X(f)(x,y) \;=\; \{\ (a, Eval(f(\cdot,y))(E))\ :\ (a,E) \in x\ \},$
  $F_2^X(f)(x,y) \;=\; \{\ (a, Eval(f(x,\cdot))(E))\ :\ (a,E) \in y\ \},$
  $F_{\rm syn}^X(f)(x,y) \;=\; \{(\tau, Eval(f)(E_1 * E_2)) : (\alpha,E_1) \in x, (\overline{\alpha},E_2) \in y \text{ for some } \alpha \neq \tau\}.$

In Section 5.3, Lemma 5.3.12 (page 113) and Lemma 5.3.29 (page 125) we show: The operators $F_\ell^{\mathbb{ID}}$, $F_L^{\mathbb{ID}}$ and $F_\parallel^{\mathbb{ID}}$ are d-continuous and have unique fixed points. These are d-continuous. The operators $F_\ell^{\mathbb{IM}}$, $F_L^{\mathbb{IM}}$ and $F_\parallel^{\mathbb{IM}}$ are contracting and their unique fixed points are non-expansive. Thus, the unique fixed points of the operators F_ℓ^X, F_L^X and $F_\parallel^X$ yield non-expansive, resp. d-continuous, semantic operators $x \mapsto x[\ell]$, $x \mapsto x \setminus L$, $(x,y) \mapsto x \| y$ on $\mathbb{ID}$ and $\mathbb{IM}$ for modelling relabelling, restriction and parallelism. Clearly, the union is d-continuous as an operator on $\mathbb{ID}$, and non-expansive when considered as an operator on $\mathbb{IM}$. The operator $\sum$ is contracting on $\mathbb{IM}$ and d-continuous on $\mathbb{ID}$. More precisely, if $(x_i)_{i\in I}$, $(x'_i)_{i\in I}$ are countable families in $\mathbb{IM}$ then

$$d\left(\ a.\left(\sum_{i\in I}[p_i]x_i\right),\ a.\left(\sum_{i\in I}[p_i]x'_i\right)\right) \;\leq\; \frac{1}{2}\cdot\max\{d(x_i,x'_i) : i \in I\}$$

If $(x_i)_{i\in I}$ is a family in $\mathbb{ID}$ such that $x_i = \bigsqcup X_i$ where X_i is a directed subset of $\mathbb{ID}$ then

$$a.\left(\sum_{i\in I}[p_i]x_i\right) \;=\; \bigsqcup\left\{\ a.\left(\sum_{i\in I}[p_i]y_i\right)\ :\ y_i \in X_i, i \in I\ \right\}.$$

Denotational semantics on $\mathbb{ID}$ and $\mathbb{IM}$: As before, we assume that $X = \mathbb{ID}$ or $X = \mathbb{IM}$. Let *decl* be a declaration where we suppose that *decl* is guarded when dealing with $X = \mathbb{IM}$. We define the operator $F_{decl}^X : (Stmt \to X) \to (Stmt \to X)$ as follows:

$$F_{decl}^X(f)(nil) \;=\; \emptyset, \qquad F_{decl}^X(f)(Z) \;=\; f(decl(Z))$$

$$F_{decl}^X(f)\left(a.\left(\sum_{i\in I}[p_i]s_i\right)\right) \;=\; a.\left(\sum_{i\in I}[p_i]F_{decl}^X(f)(s_i)\right)$$

$$F^X_{decl}(f)(s_1 + s_2) \;=\; F^X_{decl}(f)(s_1) \;\cup\; F^X_{decl}(f)(s_2)$$

$$F^X_{decl}(f)(s_1 \parallel s_2) \;=\; F^X_{decl}(f)(s_1) \parallel F^X_{decl}(f)(s_2)$$

$$F^X_{decl}(f)(s \setminus L) \;=\; F^X_{decl}(f)(s) \setminus L, \quad F^X_{decl}(f)(s[\ell]) \;=\; F^X_{decl}(f)(s)[\ell]$$

By the results of [BMC94], $F^{\mathbb{ID}}_{decl}$ is d-continuous, and hence has a least fixed point $f^{\mathbb{ID}}_{decl}$; $F^{\mathbb{IM}}_{decl}$ is contracting and hence has a unique fixed point $f^{\mathbb{IM}}_{decl}$. We obtain a denotational semantics for *PCCS* on $\mathbb{ID}$ and for *GPCCS* on $\mathbb{IM}$:

$$\mathcal{D}^{\mathbb{ID}} : PCCS \to \mathbb{ID}, \quad \mathcal{D}^{\mathbb{IM}} : GPCCS \to \mathbb{IM}$$

are given by $\mathcal{D}^X[\![\langle decl, s\rangle]\!] = f^X_{decl}(s)$.

Theorem 5.1.24 *The denotational semantics $\mathcal{D}^{\mathbb{ID}}$ and $\mathcal{D}^{\mathbb{IM}}$ are fully abstract with respect to simulation and bisimulation respectively. More precisely:*

(a) If $\mathcal{P}$, $\mathcal{P}' \in PCCS$ then $\mathcal{D}^{\mathbb{ID}}[\![\mathcal{P}]\!] = \imath_{\mathbb{ID}}([\![\mathcal{P}]\!])$ and $\mathcal{P} \sqsubseteq_{\rm sim} \mathcal{P}'$ iff $\mathcal{D}^{\mathbb{ID}}[\![\mathcal{P}]\!] \subseteq \mathcal{D}^{\mathbb{ID}}[\![\mathcal{P}']\!]$.
(b) If $\mathcal{P}$, $\mathcal{P}' \in GPCCS$ then $\mathcal{D}^{\mathbb{IM}}[\![\mathcal{P}]\!] = [\![\mathcal{P}]\!]$ and $\mathcal{P} \sim \mathcal{P}'$ iff $\mathcal{D}^{\mathbb{IM}}[\![\mathcal{P}]\!] = \mathcal{D}^{\mathbb{IM}}[\![\mathcal{P}']\!]$.

Here, $\mathbb{IP}$ is considered as a subspace of $\mathbb{IM}$ (Theorem 5.1.21, page 98) and $\imath_{\mathbb{ID}} : \mathbb{IP} \to \mathbb{ID}$ is as in Theorem 5.1.14, page 95.

Proof: see Section 5.3, Theorem 5.3.34 (page 127) and Theorem 5.3.34 (page 127). ■

Example 5.1.25 Let $\mathcal{P}_0 = \langle decl_0, s_0\rangle$ be as in Example 4.1.3 on page 77, i.e.

$$s_0 = s + Z \text{ and } decl_0(Z) = a.Z \text{ where } s = a.\Big(\Big[\tfrac{1}{3}\Big]\, b.nil \oplus \Big[\tfrac{2}{3}\Big]\, nil\Big).$$

The denotational semantics $\mathcal{D}^X[\![\mathcal{P}_0]\!]$ is $x \cup y$ where x and y are as follows.

- Case $X = \mathbb{ID}$: $y = \{(a, E_\mu)\}^{cl}$ and μ the unique distribution with
$$\mu(\bot_{\mathbb{ID}}) = 2/3,\ \mu(y_b) = 1/3,\ y_b = \{(b, E_{\mu^1_{\bot_{\mathbb{ID}}}})\}^{cl}$$
while x is the unique element in $\mathbb{ID}$ such that $x = \{(a, E_{\mu^1_x})\}^{cl}$.[19]
- Case $X = \mathbb{IM}$: $y = \{(a, E_\mu)\}$ and μ the unique distribution with $\mu(\emptyset) = 2/3$, $\mu(y_b) = 1/3$, $y_b = \{(b, E_{\mu^1_\emptyset})\}$ and x the unique element in $\mathbb{IM}$ with $x = \{(a, E_{\mu^1_x})\}$.[20] ■

Clearly, $\mathbb{ID}$ is more "abstract" than $\mathbb{IM}$ since simulation equivalence is coarser than bisimulation equivalence. We obtain the following *consistency* result for $\mathcal{D}^{\mathbb{ID}}$ and $\mathcal{D}^{\mathbb{IM}}$.[21]

Theorem 5.1.26 *There exists a unique function $f : \mathbb{IM} \to \mathbb{ID}$ such that*

$$f(x) \;=\; \{(a, Eval(f)(x)) : (a, E) \in x\}^{cl}$$

for all $x \in \mathbb{IM}$. This function f satisfies $f\Big(\ \mathcal{D}^{\mathbb{IM}}[\![\mathcal{P}]\!]\ \Big) \;=\; \mathcal{D}^{\mathbb{ID}}[\![\mathcal{P}]\!]$ for all $\mathcal{P} \in GPCCS$.

[19] Note that the notation $x = \{(a, E_{\mu^1_x})\}^{cl}$ only makes sense as we treat the isomorphism between $\mathbb{ID}$ and $\mathcal{F}_{\rm cont}(\mathbb{ID})$ as an equality. The precise definition of x is as follows. We define $x = \bigsqcup x_n$ where $x_0 = \bot_{\mathbb{ID}}$, $x_{n+1} = \{(a, E_{\mu^1_{x_n}})\}^{cl}$.

[20] Formally, $x = \lim x_n$ where $x_0 = \emptyset$, $x_{n+1} = \{(a, E_{\mu^1_{x_n}})\}$.

[21] For the notion "consistency" see [BMC97].

5.1.5 A few remarks about probabilistic powerdomains

To construct the denotational models we have generalized to the probabilistic setting the established catgorical methods for solving domain equations for non-probabilistic processes. These generalized domain equations involved appropriately adjusted probabilistic powerdomains of evaluations. The probabilistic powerdomain $Eval(D)$ of evaluation for a dcpo D is *smooth* in the sense that, for example, the probabilistic powerdomain $Eval(D)$ of a two-point space D is the real interval $[0,1]$. Thus, limits can be approximated by approaching them arbitrarily close. On the other hand, in the ultrametric case we obtain a *discrete* construction, in the sense that the two-point space lifted to the probabilistic case gives the real interval $[0,1]$ with the discrete topology. In particular, it is not possible to get arbitrarily close to a limit.[22] Another difference between the metric and partial order approach is the *density* of $\imath_{\mathbb{M}}(\mathbb{P})$ in $\mathbb{M}$ that stands in contrast to Lemma 5.3.15 (page 116) which shows that $\imath_{\mathbb{D}}(\mathbb{P})$ is *not a basis* of $\mathbb{D}$.

De Vink & Rutten [dViRu97] consider "continuous" reactive systems where each state s and possible action a in s is associated with a probability measure for the possible next states (rather than a distribution) and generalize the definition of bisimulation equivalence à la Larsen & Skou [LaSk89] to continuous reactive systems. More precisely, [dViRu97] solve the domain equation

$$M \quad \cong \quad Act \to \Big(\{\mathbf{0}\} \cup ProbMeas_{cs}(M)_{\frac{1}{2}} \Big)$$

in *CUM* (also with the methods of [RuTu93]) and show that the resulting domain is internally fully abstract with respect to the proposed notion of bisimulation. Here, $ProbMeas_{cs}(M)$ denotes the collection of probability measures on the Borel σ-field induced by the opens on M with *compact support* which means probability measures that vanish outside a compact set. The special symbol $\mathbf{0}$ is needed to model inaction. Clearly, each evaluation E on an ultrametric space M can be extended to a probability measure on the Borel σ-field of M in a unique way; but, in general, the induced probability measure does not have a compact support. Vice versa, a probability measure with compact support might fail the axiom of continuity. More precisely, if E is a probability measure on the Borel σ-field of an ultrametric space M then

$$E\left(\bigcup_{i \in I} U_i \right) \;=\; \sup_{i \in I} \; E(U_i)$$

for any directed *countable* family $(U_i)_{i \in I}$ of opens in M while

$$E\left(\bigcup_{j \in J} V_j \right) \;\neq\; \sup_{j \in J} \; E(V_j)$$

for a directed *uncountable* family of opens V_j in M is possible. Thus, $Eval(M)$ and $ProbMeas_{cs}(M)$ are non-comparable subsets of the space $ProbMeas(M)$ of *all* probability

[22] We should emphasise though that the methodology we used to derive the ultrametric model $\mathbb{M}$ is consistent with the established methodology (in particular, the metric satisfies the intuitive property $d(x,y) \leq \frac{1}{2^n}$ iff x and y agree up to the n-th step, and we obtain full abstraction for bisimulation), and that an attempt to obtain a "smooth" metric construction might mean having to go beyond the known techniques, see [KwNo96, Norm97].

measures on the Borel σ-field of M. Nevertheless, to obtain a denotational model that is fully abstract with respect to bisimulation we could follow the approach of [dViRu97] and work with the powerdomain $ProbMeas_{cs}(\cdot)$ of probability measures with compact support (rather than the probabilistic powerdomain $Eval(\cdot)$ of evaluations). That is, alternatively we could deal with the equation

$$M \quad \cong \quad Pow_{comp}\left(Act \times ProbMeas_{cs}(M)_{\frac{1}{2}}\right)$$

that can also be solved in CUM with the method of [RuTu93]. In that case, we would have to shrink the semantics for $PCCS$ on those programs that only use *finite branching* action-guarded probabilistic choice $a.([p_1]s_1 \oplus \ldots \oplus [p_n]s_n)$ rather than *countable branching* action-guarded probabilistic choice $a.(\sum_{i\in I}\ [p_i]s_i\ )$.[23]

5.2 Denotational models: fully probabilistic case

We briefly summarize how the results of the previous section can be modified for the fully probabilistic case, i.e. to obtain fully abstract denotational semantics for the process calculus $PSCCS$ (see Section 4.2, page 79 ff). As before, we assume Act to be a fixed finite nonempty set of actions and use the symbol $\mathbf{0}$ to denote inaction. Let $\mathbb{P}_f$ denote the collection of all bisimulation equivalence classes of fully probabilistic processes with action labels in Act. Then, $\mathbb{P}_f$ is the final solution of the equation

$$X \quad \cong \quad \{\mathbf{0}\} \ \cup \ Distr(Act \times X)$$

in SET. From this, we obtain *final semantics* à la [RuTu93]. Using the evaluation functor $Eval$ on the categories $CONT_\perp$ and CUM, *internally fully abstract* semantic domains $\mathbb{D}_f$ and $\mathbb{M}_f$ can be derived as follows. Applying the methods of [AbJu94, RuTu93] we define

- $\mathbb{D}_f$ as the initial solution of the equation $D \ \cong \ Eval\left(\{\perp\} \cup Act \times D\right)$ in $CONT_\perp$,[24]
- $\mathbb{M}_f$ as the unique solution of the equation $M \ \cong \ \{\mathbf{0}\} \ \cup \ Eval\left(Act \times M_{\frac{1}{2}}\right)$ in CUM.

The domain $\mathbb{P}_f$ can be "embedded" into $\mathbb{D}_f$ and $\mathbb{M}_f$ in a similar way as $\mathbb{P}$ is "embedded" into $\mathbb{D}$ and $\mathbb{M}$.[25] Using appropriate semantic operators on $\mathbb{D}_f$ and $\mathbb{M}_f$ – that can be obtained in a similar way as we defined the semantic operators on $\mathbb{D}$ and $\mathbb{M}$ – and the standard procedures to give denotational semantics in the partial order and metric approach we obtain *denotational semantics*

$$\mathcal{D}^{\mathbb{D}_f} : PSCCS \to \mathbb{D}_f \text{ and } \mathcal{D}^{\mathbb{M}_f} : GPSCCS \to \mathbb{M}_f$$

that are *fully abstract* with respect to *simulation* and *bisimulation* respectively.[26] I.e. $\mathcal{P} \sqsubseteq_{\text{sim}} \mathcal{P}'$ iff $\mathcal{D}^{\mathbb{D}_f}[\![\mathcal{P}]\!] \sqsubseteq_{\mathbb{D}_f} \mathcal{D}^{\mathbb{D}_f}[\![\mathcal{P}']\!]$ and $\mathcal{P} \sim \mathcal{P}'$ iff $\mathcal{D}^{\mathbb{M}_f}[\![\mathcal{P}]\!] = \mathcal{D}^{\mathbb{M}_f}[\![\mathcal{P}']\!]$.

[23] This is because the associated probability measure of a distribution with infinite support might have a non-compact support.

[24] Note that in $\mathbb{D}_f$, inaction is modelled by the evaluation $E_{\mu^1_\perp}$.

[25] $\mathbb{P}_f$ can be viewed as a *dense* subspace of $\mathbb{M}_f$ (which yields a *distance* for fully probabilistic processes). There is a function $\imath : \mathbb{P}_f \to \mathbb{D}_f$ such that $\mathcal{T} \sqsubseteq_{\text{sim}} \mathcal{T}'$ iff $\imath(\mathcal{T}) \sqsubseteq_{\mathbb{D}_f} \imath(\mathcal{T}')$. In particular, $\imath([\![\mathcal{P}]\!])$ can be viewed as a canonical representative for the simulation equivalence class of $\mathcal{P}$. Here, $[\![\mathcal{P}]\!]$ denotes the bisimulaton equivalence class of the fully probabilistic process $\mathcal{P}$.

[26] Here, $GPSCCS$ denotes the set of guarded $PSCCS$ programs which are defined in the obvious way.

5.3 Proofs

5.3.1 The partial order $\preceq_{\text{sim}}$ on $Distr(D)$

We give the proof for Theorem 5.1.12 (page 95) that states that $\preceq_{\text{sim}}$ (in the sense of Notation 5.1.11, page 95) is a partial order on distributions of a dcpo D and that the function $Distr(D) \to Eval(D)$, $\mu \mapsto E_\mu$ is an order-preserving embedding on the partially ordered set $Distr(D)$ into the dcpo $Eval(D)$. As a corollary we obtain Theorem 3.4.15 (page 59) and Theorem 3.4.19 (page 61) stating that simulation and bisimulation equivalence coincide for reactive or fully probabilistic systems.

Lemma 5.3.1 *Let D be a dcpo and μ, $\mu' \in Distr(D)$ such that $E_\mu \sqsubseteq E_{\mu'}$. Then, $\mu[U] \leq \mu'[U]$ for all G_δ-sets U.*[27]

Proof: Let $U = \bigcap_{i \geq 0} U_i$ where $U_i \in Opens(D)$. W.l.o.g. $U_1 \supseteq U_2 \supseteq \ldots$ (otherwise we deal with $U_i' = U_1 \cap \ldots \cap U_i$). Then, $A_i = D \setminus U_i$ are closed sets with $A_1 \subseteq A_2 \subseteq \ldots$ Since $E_\mu \sqsubseteq E_{\mu'}$ we have $\mu[A_i] \geq \mu'[A_i]$ for all $i \geq 1$. Let $A = \bigcup A_i = D \setminus U$. It is sufficient to show that $\mu[A] \geq \mu'[A]$. We suppose $\mu[A] < \mu'[A]$. We define $\Delta = \frac{1}{2}(\mu'[A] - \mu[A])$. Then, $\Delta > 0$. There exists a finite subset X of A with $\mu'[X] > \mu'[A] - \Delta$. Since X is finite there exists $i \geq 1$ with $X \subseteq A_i$. Thus,

$$\mu[A_i] \leq \mu[A] \ < \mu[A] + \Delta \ = \ \mu'[A] - \Delta \ < \mu'[X] \ \leq \ \mu'[A_i].$$

Contradiction. ■

Theorem 5.3.2 (cf. Theorem 5.1.12, page 95) *Let D be a dcpo and μ, $\mu' \in Distr(D)$. Then, $\mu \preceq_{\text{sim}} \mu'$ iff $E_\mu \sqsubseteq E_{\mu'}$.*

Proof: For simplicity $E = E_\mu$ and $E' = E_{\mu'}$.

$\Longrightarrow$: Let $\mu \preceq_{\text{sim}} \mu'$ and $weight : D \times D \to [0,1]$ be a weight function for (μ, μ') with respect to the partial order $\sqsubseteq$ on D. For all $U \in Opens(D)$:

$$\text{If } x \in U \text{ and } y \in D \setminus U \text{ then } weight(x,y) = 0.$$

(This is because U is upward-closed and because of the third condition of weight functions.) By the second condition of weight functions:

$$E(U) \quad = \sum_{x \in U} \mu(x) \ = \sum_{x \in U} \sum_{y \in U} weight(x,y) \ = \sum_{y \in U} \sum_{x \in U} weight(x,y)$$

$$\leq \quad \sum_{y \in U} \sum_{x \in D} weight(x,y) \ = \sum_{y \in U} \mu'(y) \ = \ E'(U).$$

Hence, $E \sqsubseteq E'$.

$\Longleftarrow$: We assume $E \sqsubseteq E'$. For $f : D \times D \to [0,1]$ be a function, we define μ_f, $\mu_f' : D \to [0,1]$ by:

$$\mu_f(x) \ = \sum_{y \in D} f(x,y), \quad \mu_f'(y) \ = \sum_{x \in D} f(x,y).$$

Let $\mathbf{F}$ be the set of functions $f : D \times D \to [0,1]$ such that

[27] Recall that G_δ-sets in a topological space are countable intersections of opens.

(1) $f(x,y) \neq 0$ for at most countably many $(x,y) \in D$.
(2) For all x, $y \in D$, $\mu_f(x) \leq \mu(x)$ and $\mu'_f(y) \leq \mu'(y)$.
(3) If $f(x,y) \neq 0$ then $x \sqsubseteq y$.

We show that there is a function $f \in \mathbf{F}$ such that $\mu_f = \mu$ and $\mu'_f = \mu'$. (Then, this function f is a weight function for (μ,μ') with respect to $\sqsubseteq$. Hence, $\mu \preceq_{\text{sim}} \mu'$.) For $f \in \mathbf{F}$, we put

$$\Theta(f) \;=\; \sum_{x,y \in D} f(x,y)$$

and

$$X_f \;=\; \{x \in D : \mu(x) > \mu_f(x)\}, \quad Y_f \;=\; \{y \in D : \mu'(y) > \mu'_f(y)\}.$$

It is easy to see that $X_f = \emptyset$ iff $\mu_f = \mu$ iff $Y_f = \emptyset$ iff $\mu'_f = \mu'$ iff $\Theta(f) = 1$. Thus, our aim is to show the existence of a function $f \in \mathbf{F}$ with $\Theta(f) = 1$. (Then, we may conclude that f is a weight function for (μ,μ') and therefore $\mu \preceq_{\text{sim}} \mu'$.)

If $f \in \mathbf{F}$ then we define a *f-path* to be a finite sequence $\vec{p} = (x_0, y_0, \ldots, x_n, y_n)$ in D such that $n \geq 0$ and

(i) $f(x_{i+1}, y_i) > 0$, $i = 0, 1, \ldots, n-1$
(ii) $x_i \sqsubseteq y_i$, $i = 0, 1, \ldots, n$
(iii) $x_0, \ldots, x_n$ and $y_0, \ldots, y_n$ are pairwise distinct (but $x_i = y_j$ is possible).

We define $\mathit{first}(\vec{p}) = x_0$, $\mathit{last}(\vec{p}) = y_n$ and

$$R_f(x) = \{\mathit{last}(\vec{p}) : \vec{p} \text{ is a } f\text{-}\mathit{path} \text{ with } \mathit{first}(\vec{p}) = x\}.$$

(Intuitively, $R_f(x)$ is the set of all $y \in D$ that can be reached from x via a f-path.)

Claim 1: Let $f \in \mathbf{F}$ and $x \in X_f$. Then, $R_f(x) \cap Y_f \neq \emptyset$.

Proof: Let $A = \{z \in D : \mu(z) > 0 \vee \mu'(z) > 0\}$, $Z = A \setminus R_f(x)$ and

$$U \;=\; \bigcap_{z \in Z} D \setminus z{\downarrow}\,.$$

As Z is countable and $z{\downarrow}$ Scott-closed, U is a G_δ-set. Thus, by Lemma 5.3.1 (page 105):

(*) $\mu'[U] \;\geq\; \mu[U]$

It is easy to see that $A \cap U \;=\; A \cap R_f(x)$.[28] Thus, $\mu[U] = \mu[R_f(x)]$ and $\mu'[U] = \mu'[R_f(x)]$. Since $x \in R_f(x)$ (as (x,x) is a f-path) and $x \in X_f$ we have

$$(**) \quad \mu[U] \;=\; \sum_{z \in R_f(x)} \mu(z) \;>\; \sum_{z \in R_f(x)} \mu_f(z) \;=\; \sum_{z \in R_f(x)} \sum_{y \in D} f(z,y).$$

We suppose that $R_f(x) \cap Y_f = \emptyset$. Then, $\mu'(y) = \mu'_f(y)$ for all $y \in R_f(x)$. Hence,

$$(***) \quad \mu'[U] \;=\; \sum_{y \in R_f(x)} \mu'(y) \;=\; \sum_{y \in R_f(x)} \mu'_f(y) \;=\; \sum_{y \in R_f(x)} \sum_{z \in D} f(z,y).$$

If $f(z,y) > 0$ and $(x_0, y_0, \ldots, x_n, y_n)$ is a f-path with $z \notin \{x_0, y_0, \ldots, x_n, y_n\}$ $x_0 = x$ and $y_n = y$ then $(x_0, y_0, \ldots, x_n, y_n, z, z)$ is a f-path. If $z = x_i$ for some i then $\vec{p} = (x_0, y_0, \ldots, x_i, x_i)$ is a f-path with $\mathit{last}(\vec{p}) = z$. If $z = y_i$ for some i then $\vec{p} = (x_0, y_0, \ldots, x_i, y_i)$ is a f-path with $\mathit{last}(\vec{p}) = z$. Thus,

[28] For the inclusion $A \cap R_f(x) \subseteq A \cap U$ we use the fact that $R_f(x)$ is upward-closed.

$$\text{if } y \in R_f(x) \text{ and } f(z,y) > 0 \text{ then } z \in R_f(x).$$

Hence,

$$\{(z,y) \in D \times D : f(z,y) > 0, y \in R_f(x)\} \ \subseteq \ \{(z,y) \in D \times D : f(z,y) > 0, z \in R_f(x)\}.$$

We obtain by (***) and (**):

$$\mu'[U] \ = \ \sum_{y \in R_f(x)} \ \sum_{z \in D} \ f(z,y) \ \leq \ \sum_{z \in R_f(x)} \ \sum_{y \in D} \ f(z,y) \ < \ \mu[U].$$

This contradicts (*). Thus, $R_f(x) \cap Y_f \neq \emptyset$. ⌋

By Claim 1 we obtain: If $f \in \mathbf{F}$, $\Theta(f) < 1$ then $X_f \neq \emptyset$. Thus (by Claim 1), there exists a f-path $\vec{p}$ with $\mathit{first}(p) \in X_f$ and $\mathit{last}(\vec{p}) \in Y_f$.

Let $\vec{p} = (x_0, y_0, \ldots, x_n, y_n)$ be a f-path with $x_0 = x \in X_f$ and $y_n = y \in Y_f$. We define

$$\Delta(f, \vec{p}) \ = \ \min \left\{ \mu(x) - \mu_f(x), \ \mu'(y) - \mu'_f(y), \ f(x_{i+1}, y_i) \ : \ i = 0, \ldots, n-1 \right\}$$

and $f[\vec{p}] : D \times D \to [0,1]$ by

$$f[\vec{p}](x,y) \ = \ \begin{cases} f(x_i, y_i) + \Delta(f, \vec{p}) & : \text{ if } (x,y) \in \{(x_i, y_i) : i = 0, 1, \ldots, n\} \\ f(x_{i+1}, y_i) - \Delta(f, \vec{p}) & : \text{ if } (x,y) \in \{(x_{i+1}, y_i) : i = 0, 1, \ldots, n-1\} \\ f(x,y) & : \text{ otherwise.} \end{cases}$$

Then, $\Delta(f, \vec{p}) > 0$, $f[\vec{p}] \in \mathbf{F}$ and

(I) $\Theta(f[\vec{p}]) \ = \ \Theta(f) + \Delta(f, \vec{p})$.

(II) For each $x \in D$ there is at most one $y \in D$ with $f(x,y) \neq f[\vec{p}](x,y)$.

(III) For each $y \in D$ there is at most one $x \in D$ with $f(x,y) \neq f[\vec{p}](x,y)$.

(IV) $|f(x,y) - f[\vec{p}](x,y)| \ \leq \ \Delta(f, \vec{p})$.

(II) and (III) follow by condition (iii) of f-paths. By induction on i we define a sequence $(f_i)_{i \geq 0}$ of functions $f_i \in \mathbf{F}$ as follows.

- Let f_0 be given by $f_0(x,y) = 0$ for all x, $y \in D$.
- Now we suppose $i \geq 1$ and that $f_0, \ldots, f_{i-1}$ are defined. If $\Theta(f_{i-1}) = 1$ then we set $f_i = f_{i-1}$. Otherwise, we define

$$\Delta_i \ = \ \sup \left\{ \ \Delta(f_{i-1}, \vec{p}) : \vec{p} \text{ is a } f\text{-path with } \mathit{first}(\vec{p}) \in X_{f_{i-1}} \text{ and } \mathit{last}(\vec{p}) \in Y_{f_{i-1}} \right\}.$$

  We choose some f_{i-1}-path $\vec{p}_i$ with

$$\mathit{first}(\vec{p}_i) \in X_{f_{i-1}}, \ \mathit{last}(\vec{p}_i) \in Y_{f_{i-1}}, \ \Delta(f_{i-1}, \vec{p}_i) > \Delta_i - 1/2^i.$$

  We define $f_i = f_{i-1}[\vec{p}_i]$.

Claim 2: $\lim f_i$ exists and $\lim f_i \in \mathbf{F}$, $\Theta(\lim f_i) = 1$.

Proof: By (I) we get $1 \ \geq \ \Theta(f_i) \ = \ \sum_{1 \leq j \leq i} \ \Delta(f_{j-1}, \vec{p}_j)$. Thus, $\sum_j \ \Delta(f_{j-1}, \vec{p}_j)$ is convergent. Let $\epsilon > 0$ and $i_\epsilon \geq 1$ such that

$$\sum_{j=i}^{k} \ \Delta(f_{j-1}, \vec{p}_j) \ < \ \epsilon \quad \text{for all } k \geq i \geq i_\epsilon.$$

By (IV), for all x, $y \in D$ and $k \geq i \geq i_\epsilon$:

$$|f_k(x,y) - f_i(x,y)| \;\leq\; \sum_{j=i+1}^{k} |f_j(x,y) - f_{j-1}(x,y)| \;\leq\; \sum_{j=i+1}^{k} \Delta(f_{j-1}, \vec{p}_j) \;<\; \epsilon.$$

Hence, $(f_i(x,y))_{i\geq 0}$ is a Cauchy sequence. Let $f(x,y) = \lim f_i(x,y)$. Then:

$$\text{(V)} \quad |f(x,y) - f_i(x,y)| \;=\; \lim_{k\to\infty} |f_k(x,y) - f_i(x,y)| \;\leq\; \epsilon \text{for all } x,y \in D,\ \epsilon > 0,\ i \geq i_\epsilon.$$

We get by (II), (III) and (IV):

$$\text{(VI)} \qquad |\mu_f(x) - \mu_{f_i}(x)| \;\leq\; \sum_{j\geq i} \Delta(f_{j-1}, \vec{p}_j) \;\leq\; \epsilon \;\text{ for all } \epsilon > 0,\ i \geq i_\epsilon,\ x \in D.$$

$$\text{(VII)} \qquad |\mu'_f(y) - \mu'_{f_i}(y)| \;\leq\; \sum_{j\geq i} \Delta(f_{j-1}, \vec{p}_j) \;\leq\; \epsilon \;\text{ for all } \epsilon > 0,\ i \geq i_\epsilon,\ y \in D.$$

Thus, $\mu_f(x) = \lim \mu_{f_i}(x) \leq \mu(x)$ and $\mu'_f(y) = \lim \mu'_{f_i}(y) \leq \mu'(y)$. Hence, $f \in \mathbf{F}$. We show that $\Theta(f) = 1$. We assume that $\Theta(f) < 1$. By Claim 1, there is some f-path $\vec{p} = (x_0, y_0, \ldots, y_n, x_n)$ with $x_0 \in X_f$ and $y_n \in Y_f$. Let $\epsilon < 1/2 \cdot \Delta(f, \vec{p})$. By (V) (and using the fact that $f(x_{j+1}, y_j) \geq \Delta(f, \vec{p})$),

$$f_i(x_{j+1}, y_j) \;\geq\; f(x_{j+1}, y_j) - \epsilon \;\geq\; \frac{1}{2} \cdot \Delta(f, \vec{p}) \;>\; 0, \quad j = 0, \ldots, n-1$$

for all $i \geq i_\epsilon$. By (VI) and (VII) (and using the fact that $\mu(x_0) - \mu_f(x_0)$, $\mu'(y_n) - \mu'_f(y_n) \geq \Delta(f, \vec{p})$) we obtain:

$$\begin{aligned} \mu(x_0) - \mu_{f_i}(x_0) \;&\geq\; \mu(x_0) - \mu_f(x_0) - \epsilon \;\geq\; \tfrac{1}{2} \cdot \Delta(f, \vec{p}), \\ \mu'(y_n) - \mu'_{f_i}(y_n) \;&\geq\; \mu'(y_n) - \mu'_f(y_n) - \epsilon \;\geq\; \tfrac{1}{2} \cdot \Delta(f, \vec{p}). \end{aligned}$$

Thus, $\vec{p}$ is a f_i-path for all $i \geq i_\epsilon$ and $\Delta(f_i, \vec{p}) \;\geq\; 1/2 \cdot \Delta(f, \vec{p})$. Let $j \geq 1$ such that $\Delta(f, \vec{p}) > 1/2^{j-1}$. Then, for all $i \geq i_\epsilon$,

$$\Delta(f_i, \vec{p}) \;\geq\; \frac{1}{2} \cdot \Delta(f, \vec{p}) \;>\; \frac{1}{2^j}.$$

Hence, $\Delta_i \;\geq\; \Delta(f_{i-1}, \vec{p}) > 1/2^j$ for all $i \geq i_\epsilon$. By definition of $\vec{p}_i$:

$$\Delta(f_{i-1}, \vec{p}_i) \;>\; \Delta_i - \frac{1}{2^i} \;>\; \frac{1}{2^j} - \frac{1}{2^i} \;\geq\; \frac{1}{2^{j+1}}$$

for all $i \geq \max\{i_\epsilon, j+1\}$. Contradiction (as $\sum_i \Delta(f_{i-1}, \vec{p}_i)$ is convergent). ⌋■

Lemma 5.3.3 *Let D be a dcpo and μ, $\mu' \in Distr(D)$ such that $E_\mu = E_{\mu'}$. Then, $\mu = \mu'$.*

Proof: Suppose $\mu(x) \neq \mu'(x)$ for some $x \in D$. W.l.o.g. $\mu(x) < \mu'(x)$. Let $\Delta = \mu'(x) - \mu(x)$. Then, $\Delta > 0$. Let $A = x\!\downarrow$. Then, A is Scott-closed. Since $\sum_{y\in A}\ \mu(y)$ and $\sum_{y\in A}\ \mu'(y)$ are convergent there exists a finite subset C_0 of A with $x \in C_0$ and

$$\sum_{y \in A\setminus C_0} \mu(y) \;<\; \Delta, \qquad \sum_{y \in A\setminus C_0} \mu'(y) \;<\; \Delta.$$

Let $K = \mu[A \setminus C_0]$, $K' = \mu'[A \setminus C_0]$, $C = C_0 \setminus \{x\}$ and $B = \bigcup \{ z\downarrow : z \in C \}$. Then, $K < \delta$, $K' < \Delta$ and A and B are Scott-closed (as C is finite). Since $E_\mu = E_{\mu'}$ we get $\mu[A] = \mu'[A]$ and $\mu[B] = \mu'[B]$. Since $C \subseteq B$ we have $\mu[C] = \mu[B] - \mu[B \setminus C]$ and $\mu'[C] = \mu'[B] - \mu'[B \setminus C]$. Since $A = (A \setminus C_0) \cup \{x\} \cup C$ and $\mu[B \setminus C] \geq 0$ we get:

$$\mu[A] = K + \mu(x) + \mu[C] = K + \mu(x) + \mu[B] - \mu[B \setminus C]$$

$$\leq K + \mu[B] + \mu(x) < \Delta + \mu[B] + \mu(x) = \mu[B] + \mu'(x).$$

Since $B \setminus C \subseteq A \setminus C_0$ we get $\mu'[B \setminus C] \leq \mu'[A \setminus C_0] = K'$. Hence,

$$\mu'[A] = K' + \mu'(x) + \mu'[C] = K' + \mu'(x) + \mu'[B] - \mu'[B \setminus C] \geq \mu'(x) + \mu'[B].$$

Since $\mu[B] = \mu'[B]$ we obtain $\mu[A] < \mu[B] + \mu'(x) = \mu'[B] + \mu'(x) \leq \mu'[A]$. Contradiction (as $\mu[A] = \mu'[A]$). ■

Corollary 5.3.4 *For every partially ordered set D, $\preceq_{\mathrm{sim}}$ is a partial order on $Distr(D)$.*

Proof: By Remark 2.2.1 (page 30), $\preceq_{\mathrm{sim}}$ is a preorder on $Distr(D)$. We show the antisymmetry of $\preceq_{\mathrm{sim}}$. Let $\overline{D}$ be the ideal completion[29] of D. We consider D as a subspace of $\overline{D}$. For μ to be a distribution on D, we define $\overline{\mu} \in Distr(\overline{D})$ as follows.

$$\overline{\mu}(x) = \begin{cases} \mu(x) & : \text{ if } x \in D \\ 0 & : \text{ otherwise.} \end{cases}$$

Clearly, if μ, $\mu' \in Distr(D)$ and $\mu \preceq_{\mathrm{sim}} \mu'$ then $\overline{\mu} \preceq_{\mathrm{sim}} \overline{\mu}'$.

Let μ, μ' be distributions on D with $\mu \preceq_{\mathrm{sim}} \mu'$ and $\mu' \preceq_{\mathrm{sim}} \mu$. Then, $\overline{\mu} \preceq_{\mathrm{sim}} \overline{\mu}'$ and $\overline{\mu}' \preceq_{\mathrm{sim}} \overline{\mu}$. As $\overline{D}$ is a dcpo we obtain by Theorem 5.3.2 (page 105): $E_{\overline{\mu}} \sqsubseteq E_{\overline{\mu}'}$ and $E_{\overline{\mu}'} \sqsubseteq E_{\overline{\mu}}$. Thus, $E_{\overline{\mu}} = E_{\overline{\mu}'}$. By Lemma 5.3.3 (page 108): $\overline{\mu} = \overline{\mu}'$. Hence, $\mu = \mu'$. ■

Lemma 5.3.5 *Let R be a preorder on a set X and μ, $\mu' \in Distr(X)$. If $\mu \preceq_{\mathrm{R}} \mu'$ and $\mu' \preceq_{\mathrm{R}} \mu$ then $\mu[A] = \mu'[A]$ for all equivalence classes A with respect to the kernel $R \cap R^{-1}$ of R.*

Proof: Let $\equiv \; = R \cap R^{-1}$ and $D = X/\equiv$ the quotient space endowed with the partial order $[x]_\equiv \sqsubseteq [x']_\equiv$ iff $(x, x') \in R$.[30] Let $f : X \to D$ be the canonical projection, i.e. $f(x) = [x]_\equiv$ for all $x \in X$. For $\mu \in Distr(X)$, we define $\mu_D : D \to [0, 1]$ by

$$\mu_D([x]_\equiv) = \sum_{x' \in [x]_\equiv} \mu(x').$$

Then, $\mu_D([x]_\equiv) = \mu[\,[y]_\equiv\,]$. It is easy to see that, if *weight* is a weight function for (μ, μ') with respect to $\preceq_{\mathrm{sim}}$ (where μ, $\mu' \in Distr(X)$) then $weight_D : D \times D \to [0, 1]$,

$$weight_D([x]_\equiv, [y]_\equiv) = \sum_{x' \in [x]_\equiv} \sum_{y' \in [y]_\equiv} weight(x', y')$$

is a weight function for (μ_D, μ'_D) with respect to $\sqsubseteq$. Hence, if μ, $\mu' \in Distr(X)$, $\mu \preceq_R \mu'$ and $\mu' \preceq_R \mu$ then $\mu_D \preceq_{\mathrm{sim}} \mu'_D$ and $\mu'_D \preceq_{\mathrm{sim}} \mu_D$. By Corollary 5.3.4 (page 109), $\mu_D = \mu'_D$. Thus, $\mu[A] = \mu_D(A) = \mu'_D(A) = \mu'[A]$ for all $A \in D = X/\equiv$. ■

[29] For the definition of an ideal completion see e.g. [AbJu94] or any other standard book about domain theory.

[30] $[y]_\equiv$ denotes the equivalence class of y with respect to $\equiv$.

Theorem 5.3.6 (cf. Theorem 3.4.15, page 59) *If $(S, Act, Steps)$ is a reactive action-labelled concurrent probabilistic system and s, $s' \in S$ then $s \sim_{\text{sim}} s'$ implies $s \sim s'$.*

Proof: We show that $\sim_{\text{sim}}$ is a bisimulation. Clearly, $\sim_{\text{sim}}$ is an equivalence relation on S. We assume S to be equipped with the preorder $R = \sqsubseteq_{\text{sim}}$. Let s, $s' \in S$, $s \sim_{\text{sim}} s'$ and $s \xrightarrow{a} \mu$. There exist transitions $s' \xrightarrow{a} \mu'$ and $s \xrightarrow{a} \mu''$ with $\mu \preceq_R \mu'$ and $\mu' \preceq_R \mu''$. As $(S, Act, Steps)$ is reactive we have $\mu = \mu''$. By Lemma 5.3.5 (page 109), $\mu[A] = \mu'[A]$ for all $A \in S/\sim_{\text{sim}}$. ■

Theorem 5.3.7 (cf. Theorem 3.4.19, page 61) *Let $(S, Act, \mathbf{P})$ be an action-labelled fully probabilistic system and s, $s' \in S$. Then, $s \sim_{\text{sim}} s'$ implies $s \sim s'$*

Proof: Clearly, if $s \sim_{\text{sim}} s'$ and s is terminal then s' is terminal and $s \sim s'$. Let $X = Act \times S$ and let s, s' be non-terminal states in S such that $s \sim_{\text{sim}} s'$. Let μ, $\mu' \in Distr(X)$ be given by $\mu(\langle a, t\rangle) = \mathbf{P}(s, a, t)$ and $\mu'(\langle a, t\rangle) = \mathbf{P}(s', a, t)$. We consider the relation $R = \{\langle a, t\rangle, \langle a, t'\rangle : t \sqsubseteq_{\text{sim}} t', a \in Act\}$. Then, $\mu \preceq_R \mu'$ and $\mu' \preceq_R \mu$. By Lemma 5.3.5 (page 109): If $a \in Act$ and $C \in S/\sim_{\text{sim}}$ then

$$\mathbf{P}(s, a, C) \;=\; \mu[C'] \;=\; \mu'[C'] \;=\; \mathbf{P}(s', a, C)$$

where $C' = \{(a, t) : t \in C\}$. (Note that $C' \in X/(R \cap R^{-1})$.) We conclude that $\sim_{\text{sim}}$ is a bisimulation. ■

5.3.2 The domain $\mathbb{D}$

This section shows the connection between $\mathbb{P}$ and $\mathbb{D}$ (Theorem 5.1.14, page 95) and the d-continuity of the semantic operators for modelling restriction, relabelling and parallelism and presents some domain-theoretic properties of $\mathbb{D}$.

Recall that $\mathbb{D}$ denotes the initial fixed point of the functor $\mathcal{F}_{\text{cont}} = Pow_{Hoare} \circ \mathcal{F}_{Act}^{cont} \circ Eval : CONT_\perp \to CONT_\perp$ where we deal with the isomorphism as an equality (Notation 5.1.10, page 94). The partial order on $\mathbb{D}$ is the inclusion, the bottom element is $\perp_{\mathbb{D}} = \{\perp\}$. The following is standard.

Notation 5.3.8 [The n-th projection $proj_n^{\mathbb{D}}$] *The functions $proj_n^{\mathbb{D}} : \mathbb{D} \to \mathbb{D}$ are defined as follows. Let $proj_0^{\mathbb{D}}(x) = \perp_{\mathbb{D}}$ for all $x \in \mathbb{D}$ and*

$$proj_{n+1}^{\mathbb{D}} = \mathcal{F}_{\text{cont}}\left(proj_n^{\mathbb{D}}\right).$$

Then, $proj_{n+1}^{\mathbb{D}}(x) \;=\; \{(a, Eval(proj_n^{\mathbb{D}})(E)) : (a, E) \in x\}^{cl}$. $(proj_n^{\mathbb{D}})_{n \geq 0}$ is a monotone sequence of strict and continuous functions on $\mathbb{D}$ satisfying

- $proj_n^{\mathbb{D}} \circ proj_k^{\mathbb{D}} = proj_k^{\mathbb{D}} \circ proj_n^{\mathbb{D}} = proj_n^{\mathbb{D}}$ for all $0 \leq n \leq k$
- $\bigsqcup_{n \geq 0} proj_n^{\mathbb{D}} \;=\; id_{\mathbb{D}}$
- $x \;\subseteq\; y$ iff $proj_n^{\mathbb{D}}(x) \;\subseteq\; proj_n^{\mathbb{D}}(y)$ for all $n \geq 0$.

Recall that, for X to be a set, we suppose the set of functions $X \to \mathbb{D}$ to be endowed with the partial order $f_1 \sqsubseteq f_2$ iff $f_1(x) \sqsubseteq f_2(x)$ for all $x \in X$. Then, $X \to \mathbb{D}$ is a dcpo (see Section 12.1.1, page 308). We often use the following fact.

Lemma 5.3.9 *Let X be a set. Each d-continuous operator $F : (X \to \mathbb{D}) \to (X \to \mathbb{D})$ with $F(proj_n^{\mathbb{D}} \circ f) = proj_{n+1}^{\mathbb{D}} \circ F(f)$ has a unique fixed point.*

Proof: The existence of a fixed point f follows by Tarski's fixed point theorem. If f and f' are fixed points of F then it can be shown by induction on n that $proj_n^{\mathbb{D}} \circ f = proj_n^{\mathbb{D}} \circ f'$. Hence,

$$f(x) \;=\; \bigsqcup_{n\geq 0} proj_n^{\mathbb{D}}(f(x)) \;=\; \bigsqcup_{n\geq 0} proj_n^{\mathbb{D}}(f'(x)) \;=\; f'(x)$$

for all $x \in \mathbb{D}$. ■

We denote the partial orders on $Eval(\mathbb{D})$ and on $\{\bot\} \cup Act \times Eval(\mathbb{D})$ by $\sqsubseteq$. Recall that $\sqsubseteq_L$ denotes the lower preorder on $\{\bot\} \cup Act \times Eval(\mathbb{D})$, i.e. if A, B are finite nonempty subsets of $\{\bot\} \cup Act \times Eval(\mathbb{D})$ then $A \sqsubseteq_L B$ iff for all $x \in A$ there exists $y \in B$ such that $x \sqsubseteq y$. Moreover,

- $A^{cl} \subseteq B^{cl}$ if and only if $A \sqsubseteq_L B$.
- $A^{cl} = A{\downarrow} = \{x : x \subseteq y \text{ for some } y \in A\}$.

Theorem 5.3.10 (cf. Theorem 5.1.14, page 95) *There exists a unique function $\imath_{\mathbb{D}} : \mathbb{P} \to \mathbb{D}$ such that*

$$\imath_{\mathbb{D}}(\mathcal{T}) \;=\; \left\{(a, E_{Distr(\imath_{\mathbb{D}})(\mu)}) : \mathcal{T} \stackrel{a}{\longrightarrow} \mu\right\}^{cl}.$$

Moreover, for all $\mathcal{T}$, $\mathcal{T}' \in \mathbb{P}$: $\mathcal{T} \sqsubseteq_{\text{sim}} \mathcal{T}'$ iff $\imath_{\mathbb{D}}(\mathcal{T}) \subseteq \imath_{\mathbb{D}}(\mathcal{T}')$.

Proof: For simplicity, we write $proj_n$ and $\imath$ rather than $proj_n^{\mathbb{D}}$ and $\imath_{\mathbb{D}}$. Let $F : (\mathbb{P} \to \mathbb{D}) \to (\mathbb{P} \to \mathbb{D})$ be given by $F(f)(\mathcal{T}) = A_f(\mathcal{T})^{cl}$ where

$$A_f(\mathcal{T}) \;=\; \{(a, E_{Distr(f)(\mu)}) : \mathcal{T} \stackrel{a}{\longrightarrow} \mu\}.$$

Note that – since $\mathcal{T}$ is finitely branching – $A_f(\mathcal{T})$ is finite and hence $F(f)(\mathcal{T}) = A_f(\mathcal{T}){\downarrow}$. We have to show that F has a unique fixed point $\imath$ and that this function $\imath$ satisfies: $\imath(\mathcal{T}) \subseteq \imath(\mathcal{T}')$ iff $\mathcal{T} \sqsubseteq_{\text{sim}} \mathcal{T}'$.

Claim 1: The operator F is d-continuous.

Proof: If $f = \bigsqcup_{i\in I} f_i$ then we show that for each $\mathcal{T} \in \mathbb{P}$:

(1) $A_{f_i}(\mathcal{T}) \sqsubseteq_L A_f(\mathcal{T})$ (which implies $F(f_i)(\mathcal{T}) \subseteq F(f)(\mathcal{T})$)

(2) Whenever $y \in \mathbb{D}$ with $A_{f_i}(\mathcal{T}) \subseteq y$ for all $i \in I$ then $A_f(\mathcal{T}) \subseteq y$. This implies that $F(f)(\mathcal{T}) \subseteq y$ for each upper bound y of $(F(f_i)(\mathcal{T}))_{i\in I}$.

Then, from (1) and (2),

$$\bigsqcup_{i\in I} F(f_i)(\mathcal{T}) \;=\; F(f)(\mathcal{T}).$$

We have $E_{Distr(f)(\mu)} = Eval(f)(\mu) = \bigsqcup Eval(f_i)(\mu) = \bigsqcup E_{Distr(f_i)(\mu)}$ by Remark 12.1.3 (page 313) and Lemma 12.1.4 (page 314). Hence:

ad (1) If $(a, E) \in A_{f_i}(\mathcal{T})$ then $\mathcal{T} \stackrel{a}{\longrightarrow} \mu$ for some distribution μ with $E = E_{Distr(f_i)(\mu)}$. Then, $E \sqsubseteq E_{Distr(f)(\mu)}$. Hence, $(a, E) \sqsubseteq (a, E_{Distr(f)(\mu)}) \in A_f(\mathcal{T})$.

ad (2) If $y \in I\!D$, $A_{f_i}(\mathcal{T}) \subseteq y$ for all $i \in I$ and $(a, E) \in A_f(\mathcal{T})$ then $E = E_{Distr(f)(\mu)}$ for some distribution μ where $\mathcal{T} \xrightarrow{a} \mu$. Then, $(a, E_{Distr(f_i)(\mu)}) \in A_{f_i}(\mathcal{T}) \subseteq y$ for all $i \in I$. Since

$$(a, E) \;=\; \bigsqcup_{i \in I} (a, E_{Distr(f_i)(\mu)})$$

in $\{\bot\} \cup Act \times Eval(I\!D)$ and y is lub-closed we get $(a, E) \in y$. $\rfloor$

Definition: Let $\imath : I\!P \to I\!D$ be the least fixed point of F (Tarski's fixed point theorem). Then, $\imath(\mathcal{T}) \;=\; F(\imath)(\mathcal{T}) \;=\; A_\imath(\mathcal{T})\downarrow$. If $\mathcal{T} \in I\!P$ then we put

$$A_n(\mathcal{T}) \;=\; \{\; (a, E_{Distr(proj_{n-1} \circ \imath)(\mu)}) \;:\; \mathcal{T} \xrightarrow{a} \mu \}.$$

Then, $proj_n(\imath(\mathcal{T})) \;=\; A_n(\mathcal{T})\downarrow$. Thus, $proj_n(\imath(\mathcal{T})) \subseteq proj_n(\imath(\mathcal{T}'))$ iff $A_n(\mathcal{T}) \;\sqsubseteq_L\; A_n(\mathcal{T}')$.

Claim 2: For all $\mathcal{T}, \mathcal{T}' \in I\!P$, $\imath(\mathcal{T}) \subseteq \imath(\mathcal{T}')$ iff $\mathcal{T} \sqsubseteq_{\rm sim} \mathcal{T}'$.

Proof: Since all elements of $I\!P$ (viewed as action-labelled concurrent probabilistic processes) are finitely branching (and hence image-finite) it is sufficient to show that

$$proj_n(\imath(\mathcal{T})) \subseteq proj_n(\imath(\mathcal{T}')) \text{ iff } \mathcal{T} \sqsubseteq_n \mathcal{T}'$$

(Lemma 3.4.13, page 59). We prove this by induction on n. In the case $n = 0$ there is nothing to show. In the induction step $n \Longrightarrow n+1$ we suppose $proj_n(\imath(\mathcal{T})) \subseteq proj_n(\imath(\mathcal{T}'))$ iff $\mathcal{T} \sqsubseteq_n \mathcal{T}'$ for all $\mathcal{T}, \mathcal{T}' \in I\!P$.

- Let $\mathcal{T}, \mathcal{T}' \in I\!P$, $proj_{n+1}(\imath(\mathcal{T})) \subseteq proj_{n+1}(\imath(\mathcal{T}'))$ and let $\mathcal{T} \xrightarrow{a} \mu$ be a transition. Let $\nu = Distr(proj_n \circ \imath)(\mu)$. Then, $(a, E_\nu) \;\in\; A_{n+1}(\mathcal{T}) \;\sqsubseteq_L\; A_{n+1}(\mathcal{T}')$. Hence, there exists $(a, E') \in A_{n+1}(\mathcal{T}')$ with $E_\nu \;\sqsubseteq\; E'$. By definition of $A_{n+1}(\mathcal{T}')$ there exists a transition $\mathcal{T}' \xrightarrow{a} \mu'$ with $E' = E_{\nu'}$ and $\nu' = Distr(proj_n \circ \imath)(\mu')$. By Theorem 5.3.2 (page 105), $\nu \preceq_{\rm sim} \nu'$. By Remark 2.2.3, $\mu \preceq_R \nu$, $\mu' \preceq_R \nu'$ where

  $$R \;=\; \{(\mathcal{T}_1, proj_n(\imath(\mathcal{T}_1)) : \mathcal{T}_1 \in I\!P\}.$$

  Using Remark 2.2.1 (page 30) and Remark 2.2.2 (page 31) we obtain $\mu \preceq_{R'} \mu'$ where

  $$R' \;=\; R\circ \subseteq \circ R^{-1} \;=\; \{\; (\mathcal{T}_1, \mathcal{T}_1') \;:\; proj_n(\imath(\mathcal{T}_1)) \;\subseteq\; proj_n(\imath(\mathcal{T}_1'))\;\}.$$

  By induction hypothesis, $R' \;\subseteq \sqsubseteq_n$. Hence, $\mu \preceq_{\sqsubseteq_n} \mu'$. Thus, $\mathcal{T} \sqsubseteq_n \mathcal{T}'$.
- Let $\mathcal{T}, \mathcal{T}' \in I\!P$, $\mathcal{T} \sqsubseteq_{n+1} \mathcal{T}'$. It suffices to show that $A_{n+1}(\mathcal{T}) \;\sqsubseteq_L\; A_{n+1}(\mathcal{T}')$. Let $(a, E) \in \; A_{n+1}(\mathcal{T})$. There exists a transition $\mathcal{T} \xrightarrow{a} \mu$ such that $E = E_\nu$ where $\nu = Distr(proj_n \circ \imath)(\mu)$. Since $\mathcal{T} \sqsubseteq_{n+1} \mathcal{T}'$ there exists a transition $\mathcal{T}' \xrightarrow{a} \mu'$ with $\mu \preceq_{\sqsubseteq_n} \mu'$. Then, $(a, E_{\nu'}) \in A_{n+1}(\mathcal{T}')$ where $\nu' = Distr(proj_n \circ \imath)(\mu')$. By Remark 2.2.3 (page 31), $\mu \preceq_R \nu$, $\mu' \preceq_R \nu'$ where R is as above. Using Remark 2.2.1 (page 30) and Remark 2.2.2 (page 31) we obtain $\nu \preceq_{R''} \nu'$ where

  $$R'' \;=\; R^{-1}\circ \sqsubseteq_n \circ R \;=\; \{(proj_n(\imath(\mathcal{T}_1)), proj_n(\imath(\mathcal{T}_1'))) : \mathcal{T}_1 \sqsubseteq_n \mathcal{T}_1'\}.$$

  By induction hypothesis we obtain $\nu \preceq_{\rm sim} \nu'$. By Theorem 5.3.2 (page 105), $E_\nu \sqsubseteq E_{\nu'}$. Thus, $(a, E) = (a, E_\nu) \sqsubseteq (a, E_{\nu'}) \in A_{n+1}(\mathcal{T}')$. $\rfloor$

Claim 3: If $\imath' : I\!P \to I\!D$ is also a fixed point of F then $\imath' = \imath$.

Proof: It is easy to see that $F(proj_n \circ f) = proj_{n+1} \circ F(f)$ for all functions $f : I\!P \to I\!D$. Hence, by Lemma 5.3.9 (page 111), F has a unique fixed point. Therefore, $\imath' = \imath$. $\rfloor\blacksquare$

Lemma 5.3.11 (cf. Remark 3.4.11, page 57) *Let $(S, Act, Steps)$ be an action-labelled concurrent probabilistic system and μ, $\mu' \in Distr(S)$ such that $\mu \preceq_R \mu'$ where $R = \{(s,s') \in S \times S : s \sqsubseteq_{\rm sim} s'\}$. Then, for each $t \in S$:*

(a) $\mu[\ t \downarrow_{\rm sim}\ ] \ \geq\ \mu'[\ t \downarrow_{\rm sim}\ ]$

(b) If $Supp(\mu)$ and $Supp(\mu')$ are finite then $\mu[\ t \uparrow_{\rm sim}\ ] \ \leq\ \mu'[\ t \uparrow_{\rm sim}\ ]$.

Here, $t \uparrow_{\rm sim} = \{u \in S : t \sqsubseteq_{\rm sim} u\}$ *and* $t \downarrow_{\rm sim} = \{u \in S : u \sqsubseteq_{\rm sim} t\}$.

Proof: For $s \in S$, we define $\mathcal{P}_s = (S, Act, Steps, s)$ and $x_s = \imath_{I\!D}([\![\mathcal{P}_s]\!])$. Then, by Theorem 5.3.10 (page 111) and Lemma 5.1.5 (page 92):

(*) $s \sqsubseteq_{\rm sim} s'$ iff $x_s \subseteq x_{s'}$.

For ν to be a distribution on S, we define $\nu_{I\!D} \in Distr(I\!D)$ by $\nu_{I\!D}(x) = \nu[U_x]$ where $U_x = \{u \in S : x_u = x\}$. By (*):

(**) If $\nu \in Distr(S)$ then $\nu[\ t \downarrow_{\rm sim}\ ] \ =\ \nu_{I\!D}[\ x_t \downarrow\ ], \quad \nu[\ t \uparrow_{\rm sim}\ ] \ =\ \nu_{I\!D}[\ x_t \uparrow\ ]$.

We fix some $t \in S$ and μ, $\mu' \in Distr(S)$ with $\mu \preceq_R \mu'$. Clearly, $\mu_{I\!D} \preceq_{\rm sim} \mu'$. Let $E = E_{\mu_{I\!D}}$ and $E' = E_{\mu'_{I\!D}}$. Then, $E \sqsubseteq E'$ (by Theorem 5.3.2, page 105). As $x_t \downarrow$ is Scott-closed we get

$$\mu[\ t \downarrow_{\rm sim}\ ] \ =\ \mu_{I\!D}[\ x_t \downarrow\ ] \ =\ E(x_t \downarrow) \ \geq\ E'(x_t \downarrow) \ =\ \mu'_{I\!D}[\ x_t \downarrow\ ] \ =\ \mu'[\ t \downarrow_{\rm sim}\ ].$$

Now we assume that $Supp(\mu)$ and $Supp(\mu')$ are finite. Let $A = Supp(\mu) \cup Supp(\mu')$, $B = \{v \in A : t \not\sqsubseteq_{\rm sim} v\}$ and

$$U \ =\ \bigcap_{v \in B} (I\!D \setminus x_v \downarrow).$$

Then, A and B are finite. Thus, U is the finite intersection of Scott-opens. Hence, U is Scott-open. Clearly, $x_t \uparrow \subseteq\ U$ and $U \cap \{x_v : v \in A\} \ \subseteq\ x_t \uparrow$. Hence, $Supp(\mu_{I\!D}) \cap U \subseteq x_t \uparrow$. Thus, $\mu_{I\!D}[\ x_t \uparrow\ ] \ =\ E(U)$. By (**), $\mu[\ t \uparrow_{\rm sim}\ ] \ =\ E(U)$. Similarly, $Supp(\mu') \cap U \subseteq x_t \uparrow$ and $\mu'[\ t \uparrow_{\rm sim}\ ] \ =\ E'(U)$. As U is Scott-open and $E \sqsubseteq E'$ we obtain

$$\mu[\ t \uparrow_{\rm sim}\ ] \ \ =\ E(U) \ \leq\ E'(U) \ =\ \mu'[\ t \uparrow_{\rm sim}\ ].$$

■

Recall the definitions of the operators $F_L^{I\!D}$, $F_\ell^{I\!D} : (I\!D \to I\!D) \to (I\!D \to I\!D)$ and $F_\parallel^{I\!D} : (I\!D \times I\!D \to I\!D) \to (I\!D \times I\!D \to I\!D)$ that were given on page 101.

Lemma 5.3.12 *The operators $F_\ell^{I\!D}$, $F_L^{I\!D}$ and $F_\parallel^{I\!D}$ are d-continuous and have unique fixed points. These are d-continuous.*

Proof: Let $F \in \{F_\ell^{I\!D}, F_L^{I\!D}, F_\parallel^{I\!D}\}$. Using the local d-continuity of *Eval* (Lemma 12.1.4, page 314) it is easy to see that F is locally d-continuous. Moreover,

$$F(proj_n^{I\!D} \circ f) = proj_{n+1}^{I\!D} \circ F(f).$$

Then, by Lemma 5.3.9 (page 111), F has a unique fixed point f. f is d-continuous since F maps d-continuous functions to d-continuous functions and since the set of d-continuous

function is lub-closed. Note that – by Tarski's fixed point theorem – the unique (least) fixed point of F can be written as least upper bound of the sequence $(F^n(f_0))_{n\geq 0}$ where $f_0(x) = \bot_{\mathbb{D}}$ for all $x \in \mathbb{D}$. Thus, by induction on n, all functions $F^n(f_0)$ are d-continuous. Hence, $f = \bigsqcup F^n(f_0)$ is d-continuous. ■

In the remainder of this section we investigate some domain-theoretic properties of the domain $\mathbb{D}$. We use some basic notions of domain theory like *SFP* domains or compactness of elements that are not explained in that thesis but can be found e.g. in [AbJu94]. The reader not familiar with (or not interested in) domain theory might skip the rest of that section.

In the non-probabilistic case where a domain-theoretic model $\mathbb{D}_{\text{nonprob}}$ for the simulation preorder can be obtained from the equation $D \cong Pow_{Hoare}(\{\bot\} \cup Act \times D))$ which can be solved in the category of *SFP* domains. The set *Tree* of finitely branching trees (the final solution of $X \cong Pow_{fin}(Act \times X)$ in *SET*) can be "embedded" into $\mathbb{D}_{\text{nonprob}}$ via a function $\imath : Tree \to \mathbb{D}_{\text{nonprob}}$ similar to the way where $\mathbb{P}$ is "embedded" into $\mathbb{D}$ via $\imath_{\mathbb{D}}$. The images of finitely branching trees of finite height with respect to $\imath$ are the compact elements of $\mathbb{D}_{\text{nonprob}}$. In particular, the set $\imath(Tree)$ is a basis of $\mathbb{D}_{\text{nonprob}}$. Moreover, if the underlying alphabet Act is finite then, for the n-th projection $proj_n : \mathbb{D}_{\text{nonprob}} \to \mathbb{D}_{\text{nonprob}}$, the elements of $proj_n(\imath(Tree))$ are compact and $proj_n(\mathbb{D}_{\text{nonprob}}) \subseteq \imath(Tree)$. (For further details about the non-probabilistic case see [Bai97]). The situation is different in the probabilistic case:

- The elements of $proj_n^{\mathbb{D}}(\imath_{\mathbb{D}}(\mathbb{P}))$ are not compact (cf. Lemma 5.3.13, page 114).
- $\imath_{\mathbb{D}}(\mathbb{P})$ is not a basis of $\mathbb{D}$ (cf. Lemma 5.3.15, page 116).
- If $n \geq 2$ then $proj_n^{\mathbb{D}}(\mathbb{D}) \not\sqsubseteq \imath_{\mathbb{D}}(\mathbb{P})$ (cf. Lemma 5.3.16, page 116).

Lemma 5.3.13 *The elements of $proj_n^{\mathbb{D}}(\imath_{\mathbb{D}}(\mathbb{P})) \setminus proj_1^{\mathbb{D}}(\imath_{\mathbb{D}}(\mathbb{P}))$, $n \geq 2$, are not compact.*

Proof: Let $n \geq 2$ and $x \in proj_n^{\mathbb{D}}(\imath_{\mathbb{D}}(\mathbb{P})) \setminus proj_1^{\mathbb{D}}(\imath_{\mathbb{D}}(\mathbb{P}))$. Then, there exists an element $(a, E_\mu) \in x$ which is maximal in x and with $\mu(y) > 0$ for some $y \in \mathbb{D}$, $y \neq \{\bot\}$. We choose $N \geq 0$ with $1/2^N < \mu(y)$ and for $n \geq N$ we put

$$\mu_n(z) \;=\; \begin{cases} \mu(z) & : \text{ if } z \neq \{\bot\},\, z \neq y \\ \mu(\{\bot\}) + 1/2^n & : \text{ if } z = \{\bot\} \\ \mu(y) - 1/2^n & : \text{ if } z = y. \end{cases}$$

Then, $\bigsqcup E_{\mu_n} = E_\mu$. Hence, $x = \bigsqcup x_n$ where $x_n = A_n^{cl}$, $A_n = (x \setminus \{(a, E_\mu)\}) \cup \{(a, E_{\mu_n})\}$ but $x \not\sqsubseteq x_n$ for all $n \geq N$. Thus, x is not compact. ■

Lemma 5.3.14 *$Distr(\mathbb{D})$ (as a subspace of $Eval(\mathbb{D})$) is not a basis of $Eval(\mathbb{D})$.*

Proof: We give an example for an evaluation $E \in Eval(\mathbb{D})$ that cannot be written as $E = \bigsqcup E_\mu$. Let $y = \{(b, E_\mu)\}^{cl}$ where $\mu = \mu^1_{\bot_{\mathbb{D}}}$. For $p \in [0,1]$, let μ_p be as in Example 5.1.15 (page 96), i.e. μ_p is the unique distribution on $\mathbb{D}$ with $\mu_p(y) = p$ and $\mu_p(\bot_{\mathbb{D}}) = 1 - p$. Let

$$x_p = \{(a, E_{\mu_p})\}^{cl}, \quad U_p = \mathbb{D} \setminus x_{1-p}{\downarrow}.$$

We define $E \in Eval(\mathbb{D})$ by $E(U) = \sup\{q : x_{1-q} \in U\}$ where $\sup\emptyset = 0$. As $x_{1-q} \in U_p$ iff $q < p$ we have $E(U_p) = p$.

Claim 1: If $\mu \in \mathit{Distr}(\mathbb{D})$, $E_\mu \sqsubseteq E$ then $\mu(x) = 0$ for all $x \in \mathbb{D} \setminus (\{\bot_{\mathbb{D}}\} \cup \{x_p : p \in [0,1]\})$.

Proof: First we observe that $\mathbb{D} \setminus \{x_p : p \in [0,1]\} \subseteq \mathbb{D} \setminus x_1{\downarrow} = U_0$. As is Scott-open we have $\mu[U_0] \leq E(U_0) = 0$. Hence, $\mu(x) = 0$ for all $x \in U_0$. ⌋

We suppose that $E = \bigsqcup_{\mu \in \mathcal{M}} E_\mu$ where $\mathcal{M} \subseteq \mathit{Distr}(\mathbb{D})$ such that $\{E_\mu : \mu \in \mathcal{M}\}$ is directed. (I.e. $\mathcal{M}$ is directed with respect to $\preceq_{\mathrm{sim}}$.) Then, for all $\mu \in \mathcal{M}$ and $p \in [0,1]$,

$$p \;=\; \sup\{\mu[U_p] : \mu \in \mathcal{M}\}.$$

Claim 2: For all $\epsilon > 0$ there is some $\mu \in \mathcal{M}$ such that $\mu[U_p] \geq p - \epsilon$ for all $p \in [0,1]$.

Proof: For each $p \in [0,1]$ we choose some μ_p with $\mu_p[U_p] \geq p - \epsilon$ (axiom of choice). Let X_p be a finite subset of $\mathbb{D}$ such that $\mu_p[\mathbb{D} \setminus X_p] < \epsilon/2$. There exists some $\delta_p > 0$ with $\delta_p < \epsilon/2$ and $\{x_q : q \in]p - \delta_p, p + \delta_p[\,\} \cap X_p \subseteq \{p\}$. As $[0,1]$ is compact there exists $p_1, \ldots, p_k \in [0,1]$ such that

$$[0,1] \;\subseteq\; \bigcup_{i=1}^{k}]p_i - \delta_i, p_i + \delta_i[.$$

where $\delta_i = \delta_{p_i}$. For all $q \in]p_i - \delta_i, p_i + \delta_i[$ and $i = 1, \ldots, k$,

$$\mu_{p_i}[U_q] \;=\; \mu_{p_i}[U_{p_i}] \;\geq\; p_i - \frac{1}{2} \cdot \epsilon \;\geq\; q - \epsilon.$$

We choose some $\mu \in \mathcal{M}$ with $\mu_{p_i} \preceq_{\mathrm{sim}} \mu$. Then, $\mu[U_q] \geq q - \epsilon$ for all $q \in [0,1]$. ⌋

Let (μ_n) be a sequence in $\mathcal{M}$ such that $\mu_1 \preceq_{\mathrm{sim}} \mu_2 \preceq_{\mathrm{sim}} \ldots$ and $\mu_n[U_p] \geq p - 1/2^n$ for all $n \geq 1$ and $p \in [0,1]$. (The existence of such a sequence follows by Claim 2 and the fact that $\mathcal{M}$ is directed with respect to $\preceq_{\mathrm{sim}}$.)

Claim 3: $\lim_{n\to\infty} \mu_n(x_{1-p}) = 0$ for all $p \in [0,1]$.

Proof: Let $\epsilon > 0$ and $n \geq 1$ such that $1/2^{n-1} < \epsilon$. For all $q \in [0,1]$, $q > p$, we have $x_{1-p} \in U_q \setminus U_p$ and $U_p \subseteq U_q$. Hence, for all q with $p < q < p + 1/2^n$,

$$q \;\geq\; \mu_n[U_q] \;\geq\; \mu_n[U_p] + \mu_n(x_{1-p}) \;\geq\; p - \frac{1}{2^n} + \mu_n(x_{1-p}) \;\geq\; q - \frac{1}{2^{n-1}} + \mu_n(x_{1-p}).$$

Thus, $\mu_n(x_{1-p}) \leq 1/2^{n-1} < \epsilon$. ⌋

Claim 4: $\lim_{n\to\infty} \mu_n(\bot_{\mathbb{D}}) = 0$.

Proof: We suppose that there is some $\epsilon > 0$ with $\mu_n(\bot_{\mathbb{D}}) \geq \epsilon$ for infinitely many n. Let (μ_{n_k}) be a subsequence of (μ_n) with $\mu_{n_k}(\bot_{\mathbb{D}}) \geq 0$ for all $k \geq 1$. We choose some $k \geq 1$ where $1/2^{n_k} < \epsilon$. Since $\bot_{\mathbb{D}} \notin U_1$ we get

$$1 - \frac{1}{2^{n_k}} \;\leq\; \mu_{n_k}[U_1] \;\leq\; 1 - \mu_{n_k}(\bot_{\mathbb{D}}) \;\leq 1 - \epsilon.$$

Contradiction. ⌋

Let $X = \bigcup_{n\geq 1}\{x \in D : \mu_n(x) > 0\}$. By Claim 1, $X \subseteq \{\bot_{\mathbb{D}}\} \cup \{x_p : p \in [0,1]\}$. Thus,

$$1 \;=\; \lim_{n\to\infty} \sum_{x \in X} \mu_n(x) \;=\; \sum_{x \in X} \lim_{n\to\infty} \mu_n(x) \;=\; 0$$

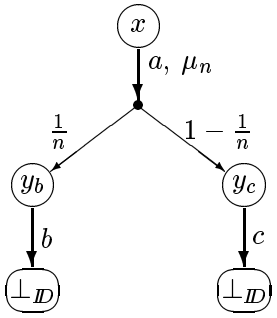
by Claim 3 and 4. Contradiction. ■

Lemma 5.3.15 *$\imath_{\mathbb{D}}(\mathbb{P})$ is not a basis of $\mathbb{D}$.*

Proof: We consider the element $y = \{(a, E)\}^{cl}$ of $\mathbb{D}$ where E is as in Lemma 5.3.14 (page 114). We suppose that y can be written in the form $y = \bigsqcup X$ for some directed subset X of $\imath_{\mathbb{D}}(\mathbb{P})$. Then, y is the Scott closure of $\bigcup_{x \in X} x$ in $\{\bot\} \cup Act \times Eval(\mathbb{D})$. Since $X \subseteq \imath_{\mathbb{D}}(\mathbb{P})$ each element x of X is of the form $\{(a, E_{\nu_1^x}), \ldots, (a, E_{\nu_{r_x}^x})\}^{cl}$ where ν_i^x are distributions with $E_{\nu_i^x} \sqsubseteq E$. Let $\mathcal{M} = \{\nu_i^x : i = 1, \ldots, r_x, x \in X\}$. It is easy to see that $\mathcal{M}$ is directed and $E = \bigsqcup\{E_\mu : \mu \in \mathcal{M}\}$ which is impossible as shown in Lemma 5.3.14 (page 114). ■

Lemma 5.3.16 *If $n \geq 2$ then $proj_n^{\mathbb{D}}(\mathbb{D}) \not\subseteq \imath_{\mathbb{D}}(\mathbb{P})$.*

Proof: Consider $x = A^{cl}$ where $A = \{(a, E_{\mu_n}) : n \geq 1\}$ and μ_n is the unique distribution on $\mathbb{D}$ with $\mu_n(y_b) = 1/n$ and $\mu_n(y_c) = 1 - 1/n$.



Here, $y_b = \{(b, E_{\mu^1_{\bot_D}})\}^{cl}$, $y_c = \{(c, E_{\mu^1_{\bot_D}})\}^{cl}$. The elements (a, E_{μ_n}), $n \geq 1$ are pairwise incomparable. Hence, x cannot be written in the form $x = X^{cl} (= X\!\downarrow)$ where X is finite. Therefore, $x \notin \imath_{\mathbb{D}}(\mathbb{P})$, but $x = proj_2^{\mathbb{D}}(x) \in proj_2^{\mathbb{D}}(\mathbb{D})$. ■

Note that Lemma 5.3.16 does not hold for $n = 1$. We have:

$$proj_1^{\mathbb{D}}(\mathbb{D}) = \{\bot_{\mathbb{D}}\} \cup \{(\alpha, E_\mu) : \alpha \in Act\} \subseteq \imath_{\mathbb{D}}(\mathbb{P})$$

where $\mu = \mu^1_{\bot_{\mathbb{D}}}$.

5.3.3 The metric probabilistic powerdomains of evaluations

This section presents the proof of Theorem 5.1.16 (page 97) stating that, for any complete ultrametric space M, the probabilistic powerdomain $Eval(M)$ of evaluations on M is the completion of $Distr(M)$. Recall that the distance on $Eval(M)$ is given by

$$d(E_1, E_2) = \inf \{ \rho > 0 : E_1(B) = E_2(B) \ \forall B \in Balls_\rho(M) \}.$$

Lemma 5.3.17 *Let M be an ultrametric space. Every nonempty open subset U of M can be written as disjoint union of open balls. If U is a ρ-set then U can be written as disjoint union of open balls with radius ρ.*

Proof: Let U be a nonempty open subset of M. If $x \in U$ then we put

$$\rho(x) = \sup \{r > 0 : B(x, r) \subseteq U\}.$$

Let $\equiv$ be the following equivalence relation on U: $x \equiv y$ iff $\rho(x) = \rho(y)$. Let V be a subset of U such that $V \cap [x]_{\equiv}$ consists exactly of one element (axiom of choice). Then, U can be written as disjoint union of the open balls $B(x, \rho(x))$, $x \in V$. In the case where U is a ρ-set we deal with the equivalence relation $x \equiv_\rho y$ iff $d(x,y) < \rho$ instead of $\equiv$. ■

Lemma 5.3.18 *Let M be an ultrametric space and E an evaluation on M. Then, for each open subset U of M, whenever $U = \bigcup_{i\in I} B_i$ where $(B_i)_{i\in I}$ is a family of pairwise disjoint open balls then for each $\epsilon > 0$ there exists a finite subset J of I with*

$$E(U) - \epsilon \leq \sum_{j\in J} E(B_j)$$

Proof: If $J \subseteq I$ is finite then we put $B_J = \bigcup_{j\in J} B_j$. Then, $E(B_J) = \sum_{j\in J} E(B_j)$. Let $\epsilon > 0$. We show that there exists a finite subset J of I with $E(B_J) \geq E(U) - \epsilon$. Let K be the set of finite subsets of I. Then, the family $(B_J)_{J\in K}$ is directed and $\bigcup_{J\in K} B_J = U$. Hence, $E(U) = \sup_{J\in K} E(B_J)$. In particular, there exists a finite subset J of I with $E(B_J) \geq E(U) - \epsilon$. ■

Immediately by Lemma 5.3.18 we get:

Corollary 5.3.19 *Each evaluation on an ultrametric space is uniquely determined by its value on the open balls.*

Lemma 5.3.20 *Let M be an ultrametric space, E an evaluation on M and $0 < r \leq 1$. Then, the set of open balls $B = B(x,r)$ for some $x \in M$ with $E(B) \neq 0$ is countable. I.e. there exists a countable subset N of M such that $E(B(x,r)) \neq 0$ implies $d(x,y) < r$ for some $y \in N$.*

Proof: Since $M = \bigcup_{B\in I} B$ where I is the set of open balls $B = B(x,r)$ for some $x \in M$ and since $B \cap B' = \emptyset$ for all B, $B' \in I$, $B \neq B'$ we get by Lemma 5.3.18 (page 117): There exists finite subsets I_n of I with

$$1 - \frac{1}{2^n} \leq \sum_{B\in I_n} E(B).$$

W.l.o.g. $I_0 \subseteq I_1 \subseteq I_2 \subseteq \ldots$ (otherwise we deal with $I'_n = I_0 \cup I_1 \cup \ldots \cup I_n$ instead of I_n). Let $J = \bigcup I_n$. Then, J is countable. Let $B \in I \setminus J$. We suppose $E(B) > 0$. Let n be a natural number with $1/2^n < E(B)$. Then,

$$1 = E(M) \geq E(B) + \sum_{B'\in I_n} E(B') \geq E(B) + 1 - \frac{1}{2^n} > 1.$$

Contradiction. Hence, $E(B) = 0$ for all $B \in I \setminus J$. ■

In the next lemma we give sufficient – and by Lemma 5.3.18 on page 117 – necessary conditions for the extension of a given function $F : \mathit{Balls}(M) \to [0,1]$ to an evaluation.

Lemma 5.3.21 *Let M be an ultrametric space and $F : \mathit{Balls}(M) \to [0,1]$ a function which satisfies:*

1. $F(M) = 1$

2. *If $B_1, \ldots, B_n$ are pairwise disjoint open balls which are contained in some open ball B then*
$$\sum_{i=1}^{n} F(B_i) \leq F(B).$$

3. *Whenever B is an open ball and $B = \bigcup_{i \in I} B_i$ where $(B_i)_{i \in I}$ is a family of pairwise disjoint open balls then for each $\epsilon > 0$ there exists a finite subset J of I with*
$$F(B) - \epsilon \leq \sum_{j \in J} F(B_j).$$

Then, there exists a unique evaluation E on M with $E(B) = F(B)$ for all $B \in Balls(M)$.

Proof: By Corollary 5.3.19 (page 117) there exists at most one evaluation E on M which extends F. We define E as follows:
$$E(U) = \sup \left\{ \sum_{B \in I} F(B) : I \in \mathcal{I}(U) \right\}$$
where $\mathcal{I}(U)$ denotes the collection of all finite sets consisting of pairwise disjoint open balls $B \subseteq U$. By Lemma 5.3.17 (page 116), $\mathcal{I}(U)$ is nonempty whenever $U \neq \emptyset$. Whenever $I \in \mathcal{I}(M)$ we put $F_I = \sum_{B \in I} F(B)$. We show that E is an evaluation: we have $E(M) = 1$ since $I = \{M\} \in \mathcal{I}(M)$ and $F(M) = 1$. The monotonicity of E is clear since whenever $U \subseteq V$ then $\mathcal{I}(U) \subseteq \mathcal{I}(V)$. Let U, $V \subseteq M$ be nonempty opens. We show that
$$E(U \cap V) + E(U \cup V) = E(U) + E(V).$$

Step 1: We show that $E(U \cap V) + E(U \cup V) \leq E(U) + E(V)$. Let $\epsilon > 0$. We show that there exists $I_U \in \mathcal{I}(U)$ and $I_V \in \mathcal{I}(V)$ with
$$E(U \cup V) + E(U \cap V) - \epsilon \leq F_{I_U} + F_{I_V}.$$
(Then, we may conclude that $E(U) + E(V) \geq E(U \cup V) + E(U \cap V) - \epsilon$ for all $\epsilon > 0$. Hence, $E(U) + E(V) \geq E(U \cup V) + E(U \cap V)$.)

Let $J \in \mathcal{I}(U \cup V)$, $K \in \mathcal{I}(U \cap V)$ with $F_J \geq E(U \cup V) - \frac{1}{4} \cdot \epsilon$ and $F_K \geq E(U \cap V) - \frac{1}{4} \cdot \epsilon$. Then,

$$(*) \qquad F_J + F_K \geq E(U \cup V) + E(U \cap V) - \tfrac{1}{2} \cdot \epsilon$$

Claim: Each ball $B \in J$ can be written as disjoint union of open balls C satisfying $C \subseteq U$ or $C \subseteq V$.

Proof: Let $B \in J$. For each $x \in B$ we put:
$$r(x) = \begin{cases} \sup \{r > 0 : B(x,r) \subseteq B \cap U\} & : \text{ if } x \in B \cap U, \\ \sup \{r > 0 : B(x,r) \subseteq B \cap V\} & : \text{ if } x \in B \cap (V \setminus U). \end{cases}$$
Then, $r(x) > 0$ for all $x \in B$. We put $B_x = B(x, r(x))$. Then, either $B_x \subseteq U$ or $B_x \subseteq V$. Let X be the set of elements $x \in B \cap U$ with $B_x \subseteq B_y$ for some $y \in B \cap (V \setminus U)$. We

can deal with the set of balls $C = B_x$ where $x \in V$ or $x \in U \setminus X$. (Note that for all x, $y \in B \cap (V \cup (U \setminus X))$ either $B_x = B_y$ or $B_x \cap B_y = \emptyset$.) $\rfloor$

Let $|J|$ be the cardinality of J and let $B \in J$. By assumption there exists a finite set I_B consisting of pairwise disjoint open balls $C \subseteq U$ or $C \subseteq V$ with

$$F(B) - \frac{1}{2|J|} \cdot \epsilon \ \leq \ F_{I_B}.$$

Let J' be the set of all balls $C \in I_B$, $B \in J$. Then, J' is finite and

$$F_J \ - \ \frac{1}{2} \cdot \epsilon \ = \ \sum_{B \in J'} F(B) \ - \ \frac{1}{2} \cdot \epsilon \ \leq \ \sum_{B \in J'} \sum_{C \in I_B} F(C).$$

We put:

$$\begin{aligned} J'_U \ &= \ \{B \in J' : B \subseteq U\}, \quad J'_V \ = \ J' \setminus J'_U, \\ K_U \ &= \ \{ \, C' \in K \ : \ C \cap B = \emptyset \ \forall \ B \in J'_U \, \}, \\ K_V \ &= \ K \setminus K_U, \quad I_U \ = \ J'_U \cup K_U, \quad I_V \ = \ J'_V \cup K_V. \end{aligned}$$

Then, $I_U \in \mathcal{I}(U)$. We show $I_V \in \mathcal{I}(V)$. It is clear that all balls $B \in I_V$ are contained in V and that the balls of J'_V (and the balls of K_V) are pairwise disjoint. Suppose there are balls $B \in J'_V$ and $C \in K_V$ with $B \cap C \neq \emptyset$. Then, either $B \subseteq C$ or $C \subseteq B$. The first case is impossible since $B \not\subseteq U$ (by definition of J'_V) and $C \subseteq U \cap V$. The second case is impossible since then $C \cap B' \neq \emptyset$ for some $B \in J'_U$ and hence either $B' \subseteq C \subseteq B$ or $C \subseteq B \cap B'$ (which contradict the assumption that the balls B, B' are disjoint). We obtain

$$\begin{aligned} F_{I_U} + F_{I_V} \ &= \ \sum_{B \in J'} \sum_{C \in I_B} F(C) \ + \ F_K \\ &\geq \ F_J - \frac{1}{2} \cdot \epsilon \ + \ F_K \ \geq \ E(U \cup V) + E(U \cap V) \ - \ \epsilon. \end{aligned}$$

Step 2: We show that $E(U \cap V) + E(U \cup V) \ \geq \ E(U) + E(V)$. Let $\epsilon > 0$ and let $I_U \in \mathcal{I}(U)$, $I_V \in \mathcal{I}(V)$ such that

$$F_{I_U} \ \geq \ E(U) - \frac{1}{2} \cdot \epsilon, \quad F_{I_V} \ \geq \ E(V) - \frac{1}{2} \cdot \epsilon.$$

Then, $K \ = \ \{B \cap C : B \in I_U, C \in I_V, B \cap C \neq \emptyset\}$ is a finite set of disjoint open balls which are contained in $U \cap V$. Let J be the set consisting of the following balls:

- $B \cup C$ where $B \in I_U$, $C \in I_V$, $B \cap C \neq \emptyset$
- $B \in I_U$ where $B \cap C = \emptyset$ for all $C \in I_V$
- $B \in I_V$ where $B \cap C = \emptyset$ for all $B \in I_U$

J is a finite set of pairwise disjoint open balls contained in $U \cup V$. (Note that whenever B, C are open balls with $B \cap C \neq \emptyset$ then either $B \subseteq C$ or $C \subseteq B$.) It is easy to see that $F_K + F_J = F_{I_U} + F_{I_V}$. Hence,

$$E(U \cup V) + E(U \cap V) \ \geq \ F_K + F_J \ = \ F_{I_U} + F_{I_V} \ \geq \ E(U) + E(V) - \epsilon$$

for all $\epsilon > 0$. Therefore, $E(U \cup V) + E(U \cap V) \geq E(U) + E(V)$.

Step 3: We show that E is continuous. Let U be a nonempty open set and let $(U_i)_{i \in I}$ be a directed family of open sets with $U = \bigcup U_i$. Since $U_i \subseteq U$ we have $E(U_i) \leq E(U)$ and therefore $\sup E(U_i) \leq E(U)$. For each $x \in U$ and $i \in I$ we put $r_i(x) = 0$ if $x \notin U_i$ and

$$r_i(x) = \sup \{r > 0 : B(x, r) \subseteq U_i\}$$

if $x \in U_i$. Let $r(x) = \sup_{i \in I} r_i(x)$. Then, $r_i(x) > 0$ and $B_x = B(x, r(x)) \subseteq U$. We define an equivalence relation $\equiv$ on U by $x \equiv y$ iff $B_x = B_y$. It is easy to see that $x \not\equiv y$ iff $B_x \cap B_y = \emptyset$ and that B_x is the equivalence class of x. For each equivalence class A, we define $B_A = B_x$ and $r_A = r(x)$, where x is a representative of A. We choose a real number ρ_A with $0 < \rho_A < r_A$. Then, U is the disjoint union of the balls B_A where A ranges over all equivalence classes. Let $\mathcal{C}_A$ be the set of all balls $B(x, \rho_A)$ where $x \in A$. B_A is the disjoint union of the balls $C \in \mathcal{C}_A$. For each ball $C \in \mathcal{C}_A$ there exists $i \in I$ and $x \in C$ such that $\rho_A < r_i(x) \leq r_A$. (Here, we use the fact that $x \equiv y$ implies $r_i(x) = r_i(y)$ for all $i \in I$.) Then, $C \subseteq B(x, r_i(x)) \subseteq U_i$. Hence, U is the disjoint union of the balls $C \in \mathcal{C}_A$ where A ranges over the equivalence classes. By definition of E there exists a finite set J of open balls where each ball $C \in J$ is contained in some of the sets U_i and which satisfies:

$$E(U) - \epsilon \leq \sum_{C \in J} F(C)$$

Since $(U_i)_{i \in I}$ is directed and since J is finite there exists an index $i \in I$ with $C \subseteq U_i$ for all $C \in J$. Then, $\bigcup_{C \in J} C \subseteq U_i$. Thus, $E(U_i) \geq \sum_{C \in J} F(C) \geq E(U) - \epsilon$. ■

Theorem 5.3.22 (cf. Theorem 5.1.16, page 97) *If M is an ultrametric space then $Eval(M)$ is an ultrametric space. If M is complete then also $Eval(M)$ is complete.*

Proof: It is clear that the distance d on $Eval(M)$ is a pseudo-ultrametric. If $d(E_1, E_2) = 0$ we have to show that $E_1(U) = E_2(U)$ for all opens U. Since $d(E_1, E_2) = 0$ we have $E_1(B) = E_2(B)$ for all $B \in Balls(M)$. U can be written as disjoint union of open balls: $U = \bigcup_{i \in I} B_i$. The family of sets $U_J = \bigcup_{i \in J} B_i$ where J is a finite subset of I is directed. For each finite subset J we have:

$$E_1(U_J) = \sum_{i \in J} E_1(B_i) = \sum_{i \in J} E_2(B_i) = E_2(U_J).$$

Since $U = \bigcup_J U_J$ we get $E_1(U) = \sup_J E_1(U_J) = \sup_J E_2(U_j) = E_2(U)$. We conclude that $Eval(M)$ is an ultrametric space.

Now we suppose M to be complete. We show that $Eval(M)$ is complete. Let $(E_n)_{n \geq 0}$ be a Cauchy sequence of evaluations on M. W.l.o.g. $d(E_n, E_N) \leq 1/2^{N+1}$ for all $0 \leq N \leq n$. For each open ball B, $(E_n(B))_{n \geq 0}$ is a Cauchy sequence. Note that the sequence $(E_n(B))$ is eventually constant as, for $B \in Balls_\rho(M)$, $E_n(B) = E_N(B)$ for all $n \geq N$ where $1/2^N \leq \rho$. We define a function $F : Balls(M) \to [0, 1]$ satisfying the conditions of Lemma 5.3.21 (page 117) as follows:

$$F(B) = \lim_{n \to \infty} E_n(B)$$

Then, $F(B) = E_n(B)$ for all $n \geq N$ and $B \in Balls_{\frac{1}{2^N}}(M)$. We show that F satisfies the conditions of Lemma 5.3.21 (page 117).

1. It is clear that $F(M) = 1$.
2. Let B be an open ball and $B_1, \ldots, B_n$ be disjoint open balls with $B_1 \cup \ldots \cup B_n \subseteq B$. We choose some $\rho > 0$ such that B, $B_i \in Balls_\rho(M)$, $i = 1, \ldots, n$, and some natural number N with $1/2^N < \rho$. Then,

$$F(B) \;=\; E_N(B) \;\geq\; \sum_{i=1}^{n} E_N(B_i) \;=\; \sum_{i=1}^{n} F(B_i).$$

3. Let $B \in Balls_\rho(M)$ and $(B_i)_{i\in I}$ a family of disjoint open balls B_i with $\bigcup B_i = B$. Let $\epsilon > 0$. We choose a natural number N with $1/2^N < \rho$. Then, $F(B) = E_n(B)$ for all $n \geq N$. Because of Lemma 5.3.18 (page 117) there exist finite subsets I'_n of I with

$$\sum_{i\in I'_n} E_n(B_i) \;\geq\; F(B) - \frac{1}{2}\cdot\epsilon.$$

We put $I_n \;=\; I'_N \cup I'_{N+1} \cup \ldots \cup I'_n$. Then, I_n are finite subsets of I with $I_N \subseteq I_{N+1} \subseteq \ldots$ and

$$a_{k,n} \;=\; \sum_{i\in I_k} E_n(B_i) \;\geq \sum_{i\in I'_n} E_n(B_i) \;\geq\; F(B) - \frac{1}{2}\cdot\epsilon$$

for all $k \geq n$. Since $a_{k,n} \;\leq\; a_{k+1,n} \;\leq\; \ldots \;\leq\; E_n(B) \;=\; F(B)$ the limit $\lim_{k\to\infty} a_{k,n}$ exists. With

$$a_n \;=\; \lim_{k\to\infty} a_{k,n}$$

we have: $a_n \;\geq\; a_{n,n} \;\geq\; F(B) - \frac{1}{2}\cdot\epsilon$. Since I_k is finite we may choose some $\rho_k > 0$ such that $B_i \in Balls_{\rho_k}(M)$ for all $i \in I_k$. For all $n \geq N$ with $1/2^n < \rho_k$, $E_n(B_i) \;=\; F(B_i)$ for all $i \in I_k$. Thus,

$$a_{k,n} \;=\; \sum_{i\in I_k} E_n(B_i) \;=\; \sum_{i\in I_k} F(B_i) \;=\; A_k.$$

Then, $A_k \;\leq\; A_{k+1} \;\leq\; \ldots \;\leq\; F(B)$ (since $(B_i)_{i\in I_k}$ is a finite family of pairwise disjoint open balls in B). Hence, $\lim_{k\to\infty} A_k$ exists, i.e.

$$\sum_{i\in K} F(B_i) \;\text{ where }\; K \;=\; \bigcup_{n\geq N} I_n$$

is convergent. Moreover,

$$\sum_{i\in K} F(B_i) \;=\; \lim_{k\to\infty}\lim_{n\to\infty} a_{k,n} \;=\; \lim_{n\to\infty}\lim_{k\to\infty} a_{k,n} \;=\; \lim_{n\to\infty} a_n \;\geq\; F(B) - \frac{1}{2}\cdot\epsilon.$$

Hence, there exists a finite subset J of K with $\sum_{j\in J} F(B_j) \;\geq\; F(B) - \epsilon$.

We conclude that F satisfies the conditions of Lemma 5.3.21 (page 117). Hence, there exists a unique evaluation E on M which extends F. Then, $E(B) = F(B) = E_N(B)$ for all $B \in Balls_\rho(M)$ where $\rho > 1/2^N$. Therefore, $d(E, E_N) \leq 1/2^N$ and $E = \lim E_n$. ■

Recall that, for M to be a complete ultrametric space, the function $Distr(M) \to Eval(M)$, $\mu \mapsto E_\mu$, is injective. Thus, $Distr(M)$ can be viewed as a subspace of $Eval(M)$.

Theorem 5.3.23 (cf. Theorem 5.1.16, page 97) *Let M be a complete ultrametric space. Then, $Eval(M)$ is the completion of $Distr(M)$.*

Proof: We have to show that $\{E_\mu : \mu \in Distr(M)\}$ is a dense subspace of $Eval(M)$. Let $E \in Eval(M)$. For all $n \geq 0$, there exists a countable subset N_n of M such that $E(B(x, 1/2^n)) \neq 0$ implies $d(x, y) < 1/2^n$ for some $y \in N_n$ (Lemma 5.3.20, page 117). For each ball $B \in Balls_{1/2^n}(M)$, we choose an element $x_B \in B$ and put

$$\mu_n(y) \;=\; \begin{cases} E(B) & : \text{ if } y = x_B \text{ for some } B \in Balls_{1/2^n}(M) \\ 0 & : \text{ otherwise.} \end{cases}$$

Then, μ_n is a distribution on M and $d(E_{\mu_n}, E) \;\leq\; 1/2^n$. Hence, $E = \lim E_{\mu_n}$. ■

5.3.4 The domain $\mathbb{M}$

This section gives the proof of Theorem 5.1.21 (page 98) that states that $\mathbb{P}$ can be viewed as a subspace of $\mathbb{M}$ and shows that the semantic operators for modelling restriction, relabelling and parallelism are non-expansive.

Recall that $\mathbb{M}$ denotes the unique fixed point of the functor $\mathcal{F}_{\text{cum}} = Pow_{Hoare} \circ \mathcal{F}^{cum}_{Act} \circ Eval : CUM \to CUM$ where we deal with the isomorphism as an equality (Notation 5.1.20, page 98). The following is standard.

Notation 5.3.24 [The n-th projection $proj^{\mathbb{M}}_n$] *The functions $proj^{\mathbb{M}}_n : \mathbb{M} \to \mathbb{M}$ are defined as follows. Let $proj^{\mathbb{M}}_0 : \mathbb{M} \to \mathbb{M}$, $proj^{\mathbb{M}}_0(x) = \emptyset$ for all $x \in \mathbb{M}$ and*

$$proj^{\mathbb{M}}_{n+1} \;=\; \mathcal{F}_{\text{cum}}\left(proj^{\mathbb{M}}_n\right).$$

Then, $proj^{\mathbb{M}}_{n+1}(x) \;=\; \{(a, Eval(proj^{\mathbb{M}}_n)(E)) : (a, E) \in x\}$. We have

$$proj^{\mathbb{M}}_n \circ proj^{\mathbb{M}}_k = proj^{\mathbb{M}}_k \circ proj\mathbb{M}_n = proj^{\mathbb{M}}_n$$

for all $0 \leq n \leq k$ and

$$x \;=\; \lim_{n\to\infty} proj^{\mathbb{M}}_n(x)$$

for all $x \in \mathbb{M}$. The distance on $\mathbb{M}$ is given by

$$d(x, y) \;=\; \inf\left\{ \frac{1}{2^n} \;:\; proj^{\mathbb{M}}_n(x) = proj^{\mathbb{M}}_n(y) \right\}.$$

Let $f : \mathbb{M} \to \mathbb{M}$ be a function. Then,

- f is non-expansive iff $proj^{\mathbb{M}}_n \circ f \;=\; proj^{\mathbb{M}}_n \circ f \circ proj^{\mathbb{M}}_n$ for all $n \geq 1$.
- f is contracting iff $proj^{\mathbb{M}}_n \circ f \;=\; proj^{\mathbb{M}}_n \circ f \circ proj^{\mathbb{M}}_{n-1}$ for all $n \geq 1$.

In particular, every function $f : \mathbb{M} \to \mathbb{M}$ with $proj^{\mathbb{M}}_n \circ f \;=\; f \circ proj^{\mathbb{M}}_{n-1}$ for all $n \geq 1$ is contracting.

Recall that, for X to be a set, the set of functions $X \to \mathbb{M}$ is supposed to be equipped with the distance $d(f_1, f_2) \;=\; \sup_{x\in X}\; d(f_1(x), f_2(x))$. Then, $X \to \mathbb{M}$ is a complete ultrametric space (cf. Section 12.1.2, page 310). We often use the following fact.

Lemma 5.3.25 *Let X be a set. Every operator $F : (X \to \mathbb{M}) \to (X \to \mathbb{M})$ with*

$$F\left(proj_n^{\mathbb{M}} \circ f\right) = proj_{n+1}^{\mathbb{M}} \circ F(f)$$

is contracting and hence has a unique fixed point.

Proof: It is easy to see that F is contracting and hence has a unique fixed point (Banach's fixed point theorem). ■

Lemma 5.3.26 *For all $n \geq 1$ and $x \in \mathbb{M}$, $proj_n^{\mathbb{M}}(x)$ is a finite set consisting of pairs (a, E_μ) where $a \in Act$ and $\mu \in Distr(\mathbb{M})$ such that $Supp(\mu) \subseteq proj_{n-1}^{\mathbb{M}}(\mathbb{M})$.*

Proof: For simplicity, $proj_n = proj_n^{\mathbb{M}}$.

Claim 1: If $E \in Eval(\mathbb{M})$ then $Eval(proj_{n-1})(E) = E_\mu$ for some distribution $\mu \in Distr(\mathbb{M})$ with $Supp(\mu) \subseteq proj_{n-1}(\mathbb{M})$.

Proof: $\mathbb{M}$ can be written as disjoint union of the open balls $B(x, 1/2^{n-2})$, $x \in proj_{n-1}(\mathbb{M})$. By Lemma 5.3.20 (page 117), there exists a countable subset N of $proj_{n-1}(\mathbb{M})$ with $E(\ B(x, 1/2^{n-2})\ ) \neq 0$ implies $x \in N$. For all opens U,

$$Eval(proj_{n-1})(E)(U) = E(proj_{n-1}^{-1}(U)) = \sum_{x \in N \cap U} E(\ B(x, 1/2^{n-2})\ ) = \mu[U]$$

where $\mu \in Distr(\mathbb{M})$ is given by $\mu(x) = 0$ if $x \notin N$, $\mu(x) = E(B(x, 1/2^{n-2}))$ if $x \in N$. ⌋

Claim 2: If E, $E' \in Eval(\mathbb{M})$ and $d(\ Eval(proj_{n-1})(E),\ Eval(proj_{n-1})(E')\ ) \leq 1/2^{n-1}$ then

$$Eval(proj_{n-1})(E) = Eval(proj_{n-1})(E').$$

Proof: Because of Claim 1 it suffices to show that $d(E_\mu, E_{\mu'}) \leq 1/2^{n-1}$ implies $\mu = \mu'$ where μ, $\mu' \in Distr(\mathbb{M})$ such that $Supp(\mu)$, $Supp(\mu') \subseteq proj_{n-1}(\mathbb{M})$. Let $x \in proj_{n-1}(\mathbb{M})$. Then, $B(x, 1/2^{n-2}) \cap proj_{n-1}(\mathbb{M}) = \{x\}$. Hence,

$$\mu(x) = \mu[B(x, 1/2^{n-2})] = \mu'[B(x, 1/2^{n-2})] = \mu'(x)$$

for all $x \in proj_{n-1}(\mathbb{M})$. Therefore, $\mu = \mu'$. ⌋

Claim 3: $proj_n(x)$ is a finite set consisting of pairs (a, E_μ) where $\mu \in Distr(\mathbb{M})$ with $Supp(\mu) \subseteq proj_{n-1}(\mathbb{M})$.

Proof: The elements of $proj_n(x)$ are of the form (a, E_μ) where $\mu \in Distr(\mathbb{M})$ such that $Supp(\mu) \subseteq proj_{n-1}(\mathbb{M})$ (see Claim 1). Since $proj_n(x)$ is compact (as a subset of $Act \times Eval(\mathbb{M})$) and since $proj_n(x) \subseteq \bigcup_{\xi \in proj_n(x)} B(\xi, 1/2^n)$ there exists a finite subset Ξ of $proj_n(x)$ with

$$proj_n(x) \subseteq \bigcup_{\xi \in \Xi} B(\xi, 1/2^n).$$

Note that $B(\xi, \rho)$ is an open ball in $Act \times Eval(\mathbb{M})_{\frac{1}{2}}$. We show that $proj_n(x) = \Xi$. Let $(a, E_\mu) \in proj_n(x)$. There exists $\xi \in \Xi$ with $d(\xi, (a, E_\mu)) < 1/2^n$. ξ is of the form (b, E_ν) where $\nu \in Distr(\mathbb{M})$ with $Supp(\nu) \subseteq proj_{n-1}(\mathbb{M})$. Then, $a = b$ and $d(E_\mu, E_\nu) < 1/2^{n-1}$. Claim 2 yields $\mu = \nu$. Therefore, $\xi = (a, E_\mu) \in \Xi$. Hence, $proj_n(x) = \Xi$ is finite. ⌋■

Theorem 5.3.27 (cf. Theorem 5.1.21, page 98) *$\mathbb{P}$ is a dense subspace of $\mathbb{M}$. More precisely, there exists a unique function $\imath_{\mathbb{M}} : \mathbb{P} \to \mathbb{M}$ such that for all $\mathcal{T} \in \mathbb{P}$,*

$$\imath_{\mathbb{M}}(\mathcal{T}) \;=\; \{(a, E_{Distr(\imath_{\mathbb{M}})(\mu)}) : \mathcal{T} \xrightarrow{a} \mu\}.$$

This function $\imath_{\mathbb{M}}$ is injective and $\imath_{\mathbb{M}}(\mathbb{P})$ is a dense subspace of $\mathbb{M}$.

Proof: We shortly write $proj_n$ and $\imath$ instead of $proj_n^{\mathbb{M}}$ and $\imath_{\mathbb{M}}$. Let $F : (\mathbb{P} \to \mathbb{M}) \to (\mathbb{P} \to \mathbb{M})$ be given by $F(f)(\mathcal{T}) = \{(a, E_{Distr(f)(\mu)}) : \mathcal{T} \xrightarrow{a} \mu\}$. Note that – since $\mathcal{T}$ is finitely branching – $F(f)(\mathcal{T})$ is finite and hence compact. We have to show that F has a unique fixed point $\imath$ and that this function $\imath$ is injective and $\imath(\mathbb{P})$ a dense subspace of $\mathbb{M}$. It is easy to see that $F(proj_n \circ f) \;=\; proj_{n+1} \circ F(f)$ for all functions $f : \mathbb{P} \to \mathbb{M}$. Hence, F is contracting (Lemma 5.3.25, page 123).

Definition: Let $\imath$ be the unique fixed point of F (Banach's fixed point theorem).

Claim 1: $\imath$ is injective.

Proof: Because of Lemma 3.4.8 (page 56) it suffices to show that $\imath(\mathcal{T}) = \imath(\mathcal{T}')$ implies $\mathcal{T} \sim_n \mathcal{T}'$ for all $n \geq 0$. Since $\mathcal{T}$, $\mathcal{T}'$ are finitely branching (and therefore image-finite) we get $\mathcal{T} \sim \mathcal{T}'$. Hence, $\mathcal{T} = \mathcal{T}'$ (Corollary 5.1.6, page 93).

We show by induction on n that $proj_n(\imath(\mathcal{T})) = proj_n(\imath(\mathcal{T}'))$ iff $\mathcal{T} \sim_n \mathcal{T}'$. In the basis of induction ($n = 0$) there is nothing to show. In the induction step $n \Longrightarrow n+1$ we suppose that, for all $\mathcal{T}_1$, $\mathcal{T}_1' \in \mathbb{P}$, $proj_n(\imath(\mathcal{T}_1)) = proj_n(\imath(\mathcal{T}_1'))$ iff $\mathcal{T}_1 \sim_n \mathcal{T}_1'$.

1. Let $proj_{n+1}(\imath(\mathcal{T})) = proj_{n+1}(\imath(\mathcal{T}'))$ and $\mathcal{T} \xrightarrow{a} \mu$. Then, $(a, E_\nu) \in proj_{n+1}(\imath(\mathcal{T}))$ where $\nu \;=\; Distr(proj_n \circ \imath)(\mu)$. Since $(a, E_\nu) \in proj_{n+1}(\imath(\mathcal{T}'))$ there exists a transition $\mathcal{T}' \xrightarrow{a} \mu'$ with $\nu \;=\; Distr(proj_n \circ \imath)(\mu')$. Let $A \in \mathbb{P}/\sim_n$, $\mathcal{T}_1 \in A$ and $x = proj_n(\imath(\mathcal{T}_1))$. By induction hypothesis,

   $$A = \{\mathcal{T}_1' \in \mathbb{P} : x = proj_n(\imath(\mathcal{T}_1'))\}.$$

   Hence, $A = \imath^{-1}(proj_n^{-1}(x))$. Thus, $\mu[A] = \nu(x) = \mu'[A]$. By symmetry, $\mathcal{T} \sim_{n+1} \mathcal{T}'$.
2. Let $\mathcal{T} \sim_{n+1} \mathcal{T}'$. By symmetry it suffices to show that $proj_{n+1}(\imath(\mathcal{T})) \;\subseteq\; proj_{n+1}(\imath(\mathcal{T}'))$. Let $(a, E) \in proj_{n+1}(\imath(\mathcal{T}))$. There is a transition $\mathcal{T} \xrightarrow{a} \mu$ with $E = E_{Distr(proj_n \circ \imath)(\mu)}$. Since $\mathcal{T} \sim_{n+1} \mathcal{T}'$ there exists a transition $\mathcal{T}' \xrightarrow{a} \mu'$ with $\mu[A] = \mu'[A]$ for all $A \in \mathbb{P}/\sim_n$. We show

   $$Distr(proj_n \circ \imath)(\mu) = Distr(proj_n \circ \imath)(\mu').$$

   Let $x \in \mathbb{M}$ and $A = \imath^{-1}(proj_n^{-1}(x))$. By induction hypothesis, $A \in \mathbb{P}/\sim_n$. Thus, $Distr(proj_n \circ \imath)(\mu)(x) = \mu[A] = \mu'[A] = Distr(proj_n \circ \imath)(\mu')(x)$. ⌋

Claim 2: $\imath(\mathbb{P})$ is a dense subspace of $\mathbb{M}$.

Proof: Since $x = \lim proj_n(x)$ for all $x \in \mathbb{M}$ the set $\bigcup proj_n(M)$ is a dense subspace of $\mathbb{M}$. Hence, it suffices to show that $proj_n(\mathbb{M}) \subseteq \imath(\mathbb{P})$ for all $n \geq 0$. We use induction on n. The case $n = 0$ is clear. In the induction step $n \Longrightarrow n+1$ we suppose $proj_n(\mathbb{M}) \subseteq \imath(\mathbb{P})$. Since $\imath$ is injective there exists a unique function $\jmath : \mathbb{M} \to \mathbb{P}$ such that $\jmath \circ \imath = id_{\mathbb{P}}$. Let $x \in proj_{n+1}(\mathbb{M})$. By Lemma 5.3.26 (page 123), x is of the form $x = \{(a_i, E_{\mu_i}) : i = 1, \ldots, k\}$ where $\mu_i \in Distr(\mathbb{M})$ such that $Supp(\mu_i) \subseteq proj_n(\mathbb{M})$. Thus, $Supp(\mu_i) \;\subseteq\; \imath(\mathbb{P})$ which yields

(*) $Distr(\imath \circ \jmath)(\mu_i) = \mu_i$, $i = 1, \ldots, k$.

Let $X = \left\{(a_i, E_{Distr(\jmath)(\mu_i)}) : i = 1, \ldots, k\right\}$ and $\mathcal{T} = e^{-1}(X)$ where $e : I\!P \to Pow_{fin}(Act \times Distr(I\!P))$ is the bijection such that $(I\!P, e)$ is the final coalgebra of $Pow_{fin} \circ \mathcal{F}_{Act} \circ Distr$ (Theorem 5.1.7, page 93). By Remark 12.1.3 (page 313) and (*):

$$Eval(\imath)(E_{Distr(\jmath)(\mu_i)}) \;=\; E_{Distr(\imath \circ \jmath)(\mu_i)} \;=\; E_{\mu_i}, \quad i = 1, \ldots, k.$$

Hence, $\imath(\mathcal{T}) = \{(a, Eval(\imath)(E)) : (a, E) \in X\} = \{(a_i, E_{\mu_i}) : i = 1, \ldots, k\} = x$. Thus, $x \in \imath(I\!P)$. ⌋■

Remark 5.3.28 In Claim 2 in the proof of Theorem 5.3.27 (page 124) we saw that $proj_n^{I\!M}(I\!M) \subseteq \imath_{I\!M}(I\!P)$. This should be contrasted with the domain-theoretic setting where $proj_n^{I\!D}(I\!D) \not\subseteq \imath_{I\!D}(I\!P)$ (cf. Lemma 5.3.16, page 116). ■

Recall the definitions of the operators $F_L^{I\!M}$, $F_\ell^{I\!M} : (I\!M \to I\!M) \to (I\!M \to I\!M)$ and $F_\parallel^{I\!M} : (I\!M \times I\!M \to I\!M) \to (I\!M \times I\!M \to I\!M)$ (see page 101).

Lemma 5.3.29 *The operators $F_\ell^{I\!M}$, $F_L^{I\!M}$ and $F_\parallel^{I\!M}$ are contracting and the unique fixed points are non-expansive.*

Proof: Let $F \in \{F_\ell^{I\!M}, F_L^{I\!M}, F_\parallel^{I\!M}\}$. It is easy to see that $F(proj_n^{I\!M} \circ f) = proj_{n+1}^{I\!M} \circ F(f)$. Hence, by Lemma 5.3.25 (page 123), F has a unique fixed point f. To see that f non--expansive we observe that F maps non-expansive functions to non-expansive functions. Since the set of non-expansive functions $I\!M \to I\!M$ is a closed subspace of the complete metric space of all functions $I\!M \to I\!M$, the unique fixed point f is non-expansive. ■

Remark 5.3.30 In the metric approach – where $Eval(M)$ is a completion of $Distr(M)$ (Theorem 5.3.23, page 122) – the product of evaluations $E_1 * E_2$ (defined as in Section 12.1.4, page 314) can be defined without using the result of Heckmann [Heck95]; namely, as the canonical extension of the non-expansive operator

$$Distr(M) \times Distr(M) \to Distr(M \times M),\ (\mu_1, \mu_2) \mapsto \mu_1 * \mu_2.$$

(For the definition of $\mu_1 * \mu_2$ see Section 2.2, page 30.) An alternative definition of the product (which leads to the same operator) uses Lemma 5.3.21 (page 117): if E_1, E_2 are evaluations on M then $E_1 * E_2$ denotes the unique evaluation on $M \times M$ such that, for all open balls B, B' of M, $(E_1 * E_2)(B \times B') \;=\; E_1(B) \cdot E_2(B')$.[31] ■

5.3.5 Full abstraction

In this section we give the proof of the full abstraction result (Theorem 5.1.24, page 102) and show the "consistency" of the partial order and metric semantics on $I\!D$ and $I\!M$ (Theorem 5.1.26, page 102).

Notation 5.3.31 [The elements $[\![s]\!]_{decl}$] *If s is a PCCS statement and decl a declaration then $[\![s]\!]_{decl}$ denotes the bisimulation equivalence class of the operational meaning of the PCCS program $\langle decl, s\rangle$, i.e. of the probabilistic process $\mathcal{O}[\![\langle decl, s\rangle]\!]$.*

[31] Note that the open balls of the product space $M \times M$ have the form $B \times B'$ where B, B' are open balls of the same radius.

The basic lemma for the full abstraction result (Theorem 5.1.24, page 102) is the following. Recall that

- $\imath_{\mathbb{D}} : \mathbb{P} \to \mathbb{D}$ is the unique function with $\imath_{\mathbb{D}}(\mathcal{T}) = \{(a, E_{Distr(\imath)(\mu)}) : \mathcal{T} \xrightarrow{a} \mu\}^{cl}$,
- $\imath_{\mathbb{M}} : \mathbb{P} \to \mathbb{M}$ the unique function such that $\imath_{\mathbb{M}}(\mathcal{T}) = \{(a, E_{Distr(\imath)(\mu)}) : \mathcal{T} \xrightarrow{a} \mu\}$.

See Theorem 5.3.10 (page 111) and Theorem 5.3.27 (page 124).

Lemma 5.3.32 *Let $X = \mathbb{M}$ or $X = \mathbb{D}$. Then, for each declaration decl:*

0. $\imath_{\mathbb{M}}(\llbracket nil \rrbracket_{decl}) = \emptyset$, $\imath_{\mathbb{D}}(\llbracket nil \rrbracket_{decl}) = \bot_{\mathbb{D}}$.
1. $\imath_X (\llbracket\ a.(\sum_{i\in I} [p_i]s_i)\ \rrbracket_{decl}) = a.(\sum_{i\in I} [p_i]\ \imath_X(\llbracket s_i \rrbracket_{decl}))$
2. $\imath_X(\llbracket\ s_1 + s_2\ \rrbracket_{decl}) = \imath_X(\llbracket s_1 \rrbracket_{decl}) \cup \imath_X(\llbracket s_2 \rrbracket_{decl})$
3. $\imath_X(\llbracket\ s_1 \parallel s_2\ \rrbracket_{decl}) = \imath_X(\llbracket s_1 \rrbracket_{decl}) \parallel \imath_X(\llbracket s_2 \rrbracket_{decl})$
4. $\imath_X(\llbracket\ s[\ell]\ \rrbracket_{decl}) = \imath_X(\llbracket s \rrbracket_{decl})[\ell]$
5. $\imath_X(\llbracket\ s \setminus L\ \rrbracket_{decl}) = \imath_X(\llbracket s \rrbracket_{decl}) \setminus L$
6. $\imath_X(\llbracket Z \rrbracket_{decl})) = \imath_X(\llbracket\ decl(Z)\ \rrbracket_{decl})$

Proof: 0., 1. and 2. are clear. 6. is clear since $\mathcal{O}\llbracket\langle decl, Z\rangle\rrbracket \sim \mathcal{O}\llbracket\langle decl, decl(Z)\rangle\rrbracket$. Hence, by Lemma 5.1.5 (page 92), $\llbracket Z \rrbracket_{decl} = \llbracket\ decl(Z)\ \rrbracket_{decl}$.

In what follows, we shortly write $proj_n$, $\imath$ rather than $proj_n^X$ and $\imath_X$ and $\llbracket s \rrbracket$ instead of $\llbracket s \rrbracket_{decl}$. As before, we use the closure notation A^{cl} for subsets of $Act \times Eval(\mathbb{M})$ and for subsets of $\{\bot\} \cup Act \times Eval(\mathbb{D})$. When dealing with $\mathbb{D}$, A^{cl} denotes the Scott-closure of A (if $A \neq \emptyset$) and $\emptyset^{cl} = \bot_{\mathbb{D}}$, as before. When dealing with $\mathbb{M}$, we put $A^{cl} = A$. Then,

$$\imath(\llbracket s \rrbracket) = \left\{(a, E_{Distr(\imath\circ\llbracket\cdot\rrbracket)(\mu)}) : s \xrightarrow{a}_{decl} \mu\right\}^{cl}$$

for all $s \in Stmt$. We show 3. By induction on n we show that

$$proj_n(\imath(\llbracket\ s_1 \| s_2\ \rrbracket)) = proj_n(\imath(\llbracket s_1 \rrbracket)) \parallel proj_n(\imath(\llbracket s_2 \rrbracket))$$

for all s_1, $s_2 \in Stmt$. Then, by the non-expansitivity/d-continuity of $\parallel$ and the fact that $x = \lim proj_n(x)$ in $\mathbb{M}$ and $x = \bigsqcup proj_n(x)$ in $\mathbb{D}$ we obtain

$$\imath(\llbracket s_1 \| s_2 \rrbracket) = \imath(\llbracket s_1 \rrbracket) \parallel \imath(\llbracket s_2 \rrbracket).$$

The basis of induction ($n = 0$) is clear as $\emptyset \| \emptyset = \emptyset$ in $\mathbb{M}$ and $\bot_{\mathbb{D}} \| \bot_{\mathbb{D}} = \bot_{\mathbb{D}}$ in $\mathbb{D}$. In the induction step $n \Longrightarrow n+1$ we suppose that

$$proj_n(\imath(\llbracket t_1 \| t_2 \rrbracket)) = proj_n(\imath(\llbracket t_1 \rrbracket)) \parallel proj_n(\imath(\llbracket t_2 \rrbracket))$$

for all t_1, $t_2 \in Stmt$. Let s_1, $s_2 \in Stmt$. Since $E_{Distr(f)(\mu)} = Eval(f)(E_\mu)$ (Remark 12.1.3, page 313) and $E_{\mu_1 * \mu_2} = E_{\mu_1} * E_{\mu_2}$ we have

$$\begin{aligned}
\imath(\llbracket s_1 \| s_2 \rrbracket) &= \{(a, E_{Distr(\imath\circ\llbracket\cdot\rrbracket)(\mu)}) \ :\ s_1 \| s_2 \xrightarrow{a} \mu\}^{cl} \\
&= \{(\tau, E_{Distr(\imath\circ\llbracket\cdot\|\cdot\rrbracket)(\mu_1*\mu_2)}) : s_1 \xrightarrow{\alpha}_{decl} \mu_1, s_2 \xrightarrow{\overline{\alpha}}_{decl} \mu_2, \alpha \neq \tau\}^{cl} \\
&\quad \cup \{(a, E_{Distr(\imath\circ\llbracket\cdot\|s_2\rrbracket)(\mu)}) \ :\ s_1 \xrightarrow{a}_{decl} \mu\}^{cl} \\
&\quad \cup \{(a, E_{Distr(\imath\circ\llbracket s_1\|\cdot\rrbracket)(\mu)}) \ :\ s_2 \xrightarrow{a}_{decl} \mu\}^{cl}.
\end{aligned}$$

We define functions f, $g : Stmt \times Stmt \to X$ by

$$\begin{aligned} f(t_1,t_2) &= proj_n(\imath([\![t_1]\!])) \parallel proj_n(\imath([\![t_2]\!])),\\ g(t_1,t_2) &= proj_n(\imath([\![t_1 \parallel t_2]\!])). \end{aligned}$$

We have

$$proj_{n+1}(\ \imath([\![s_1 \| s_2]\!])\ ) \ = \ \bigcup_{a\in Act} \{(a,E_\nu) : \nu \in \mathcal{M}^a\}^{cl}$$

where $\mathcal{M}^\alpha = \mathcal{M}_1^\alpha \cup \mathcal{M}_2^\alpha$ if $\alpha \neq \tau$ and $\mathcal{M}^\tau = \mathcal{M}_1^\tau \cup \mathcal{M}_2^\tau \cup \mathcal{M}_{\rm syn}$,

$$\begin{aligned} \mathcal{M}_1^a &= \{Distr(g(\cdot,s_2))(\mu) : s_1 \xrightarrow{a}_{decl} \mu\},\\ \mathcal{M}_2^a &= \{Distr(g(s_1,\cdot)(\mu) : s_2 \xrightarrow{a}_{decl} \mu\},\\ \mathcal{M}_{\rm syn} &= \{Distr(g)(\mu_1 * \mu_2) : s_1 \xrightarrow{\alpha}_{decl} \mu_1,\ s_2 \xrightarrow{\overline{\alpha}}_{decl} \mu_2, \alpha \neq \tau\}. \end{aligned}$$

On the other hand,

$$proj_{n+1}(\ \imath([\![s_1]\!])\ ) \ \parallel \ proj_{n+1}(\ \imath([\![s_2]\!])\ ) \ = \ \bigcup_{a\in Act} \{(\alpha,E_\nu) : \nu \in \mathcal{N}^a\ \}^{cl}$$

where $\mathcal{N}^\alpha = \mathcal{N}_1^\alpha \cup \mathcal{N}_2^\alpha$ if $\alpha \neq \tau$ and $\mathcal{N}^\tau = \mathcal{N}_1^\tau \cup \mathcal{N}_2^\alpha \cup \mathcal{N}_{\rm syn}$,

$$\begin{aligned} \mathcal{N}_1^a &= \{Distr(f(\cdot,s_2))(\mu) : s_1 \xrightarrow{a}_{decl} \mu\},\\ \mathcal{N}_2^a &= \{Distr(f(s_1,\cdot)(\mu) : s_2 \xrightarrow{a}_{decl} \mu\},\\ \mathcal{N}_{\rm syn} &= \{Distr(f)(\mu_1 * \mu_2) : s_1 \xrightarrow{\alpha}_{decl} \mu_1,\ s_2 \xrightarrow{\overline{\alpha}}_{decl} \mu_2, \alpha \neq \tau\}. \end{aligned}$$

The induction hypothesis yields $f(t_1,t_2) = g(t_1,t_2)$ for all t_1, $t_2 \in Stmt$. Thus, $\mathcal{M}_1^a = \mathcal{N}_1^a$, $\mathcal{M}_2^a = \mathcal{N}_2^a$ and $\mathcal{M}_{\rm syn} = \mathcal{N}_{\rm syn}$. We conclude:

$$proj_{n+1}(\ \imath([\![s_1 \| s_2]\!])\ ) \ = \ proj_{n+1}(\ \imath([\![s_1]\!])\ ) \ \parallel \ proj_{n+1}(\ \imath([\![s_2]\!])\ ).$$

The proofs of 4. and 5. are similar to the proof of 3. ■

Recall that $f^{\mathbb{ID}}_{decl}$ denotes the *least* fixed point of the d-continuous operator $F^{\mathbb{ID}}_{decl}$ (see Section 5.1.4, page 101). In the next lemma we show that – as in the metric case where $f^{\mathbb{IM}}_{decl}$ is the unique fixed point of $F^{\mathbb{IM}}_{decl}$ – $f^{\mathbb{ID}}_{decl}$ is *unique* as a fixed point of $F^{\mathbb{ID}}_{decl}$.

Lemma 5.3.33 *Let decl be a declaration. Then, $f^{\mathbb{ID}}_{decl}$ is the unique fixed point of $F^{\mathbb{ID}}_{decl}$.*

Proof: It is easy to see that $proj^{\mathbb{ID}}_{n+1} \circ F^{\mathbb{ID}}_{decl}(proj^{\mathbb{ID}}_n \circ f) = proj^{\mathbb{ID}}_{n+1} \circ F^{\mathbb{ID}}_{decl}(f)$. Hence, if f, f' are fixed points of $F^{\mathbb{ID}}_{decl}$ then (by induction on n) $proj^{\mathbb{ID}}_n \circ f = proj^{\mathbb{ID}}_n \circ f'$. Hence,

$$f(s) \ = \ \bigsqcup_{n\geq 0} proj^{\mathbb{ID}}_n(f(s)) \ = \ \bigsqcup_{n\geq 0} proj^{\mathbb{ID}}_n(f'(s)) \ = \ f'(s)$$

for all $s \in Stmt$. Therefore $f = f'$. ■

Theorem 5.3.34 (cf. Theorem 5.1.24, page 102) *The denotational semantics $\mathcal{D}^{\mathbb{ID}}$ and $\mathcal{D}^{\mathbb{IM}}$ are fully abstract with respect to simulation and bisimulation respectively. More precisely:*

(a) If $\mathcal{P}$, $\mathcal{P}' \in PCCS$ then $\mathcal{D}^{\mathbb{ID}}[\![\mathcal{P}]\!] = \imath_{\mathbb{ID}}([\![\mathcal{P}]\!])$ and $\mathcal{P} \sqsubseteq_{\rm sim} \mathcal{P}'$ iff $\mathcal{D}^{\mathbb{ID}}[\![\mathcal{P}]\!] \subseteq \mathcal{D}^{\mathbb{ID}}[\![\mathcal{P}']\!]$.
(b) If $\mathcal{P}$, $\mathcal{P}' \in GPCCS$ then $\mathcal{D}^{\mathbb{IM}}[\![\mathcal{P}]\!] = [\![\mathcal{P}]\!]$ and $\mathcal{P} \sim \mathcal{P}'$ iff $\mathcal{D}^{\mathbb{IM}}[\![\mathcal{P}]\!] = \mathcal{D}^{\mathbb{IM}}[\![\mathcal{P}']\!]$.

Here, $\mathbb{IP}$ is considered as a subspace of $\mathbb{IM}$ (Theorem 5.3.27, page 124) and $\imath_{\mathbb{ID}} : \mathbb{IP} \to \mathbb{ID}$ is as in Theorem 5.3.10, page 111.

Proof: Using Lemma 5.3.32 (page 126) it can be shown by structural induction on the syntax of $s \in \mathit{Stmt}$ that $F^X_{decl}(\imath_X \circ [\![\cdot]\!]_{decl})(s) = \imath_X([\![s]\!]_{decl})$. By the uniqueness of f^X_{decl} as a fixed point of F^X_{decl} (Lemma 5.3.33, page 127), we get $f^X_{decl} = \imath_X \circ [\![\cdot]\!]_{decl}$. Hence,

$$\mathcal{D}^X[\![\langle decl, s\rangle]\!] = f^X_{decl}(s) = \imath_X([\![\langle decl, s\rangle]\!]).$$

Lemma 5.1.5 (page 92) yields $\mathcal{P} \sqsubseteq_{\mathrm{sim}} \mathcal{P}'$ iff $\mathcal{D}^{\mathbb{ID}}[\![\mathcal{P}]\!] \subseteq \mathcal{D}^{\mathbb{ID}}[\![\mathcal{P}']\!]$ and $\mathcal{P} \sim \mathcal{P}'$ iff $\mathcal{D}^{\mathbb{IM}}[\![\mathcal{P}]\!] = \mathcal{D}^{\mathbb{IM}}[\![\mathcal{P}']\!]$. ■

Theorem 5.3.35 (cf. Theorem 5.1.26, page 102) *There exists a unique function $f : \mathbb{IM} \to \mathbb{ID}$ such that $f(x) = \{(a, \mathit{Eval}(f)(x)) : (a, E) \in x\}^{cl}$ for all $x \in \mathbb{IM}$. This function f satisfies $f\left(\mathcal{D}^{\mathbb{IM}}[\![\mathcal{P}]\!]\right) = \mathcal{D}^{\mathbb{ID}}[\![\mathcal{P}]\!]$ for all $\mathcal{P} \in \mathit{GPCCS}$.*

Proof: We define a function $f : \mathbb{IM} \to \mathbb{ID}$ as follows. We consider the function

$$F : (\mathbb{IM} \to \mathbb{ID}) \to (\mathbb{IM} \to \mathbb{ID}), \quad F(f)(x) = \{(a, \mathit{Eval}(f)(x)) : (a, E) \in x\}^{cl}.$$

It is easy to see that F is d-continuous and $F(\mathit{proj}^{\mathbb{ID}}_n \circ f) = \mathit{proj}^{\mathbb{ID}}_{n-1} \circ F(f)$. Thus, F satisfies the conditions of Lemma 5.3.9 (page 111). Let $f : \mathbb{IM} \to \mathbb{ID}$ be the unique fixed point of F. It is easy to see that f is a "homomorphism" with respect to the semantic operators on $\mathbb{IM}$ and $\mathbb{ID}$. Using the results of [BMC97] it can be shown that, for fixed guarded declaration decl, $f \circ f^{\mathbb{IM}}_{decl}$ is a fixed point of $F^{\mathbb{ID}}_{decl}$. By Lemma 5.3.33 (page 127), $f^{\mathbb{ID}}_{decl}$ is the unique fixed point of $F^{\mathbb{ID}}_{decl}$. Hence, $f \circ f^{\mathbb{IM}}_{decl} = f^{\mathbb{ID}}_{decl}$ which yields the "consistency result" $f \circ \mathcal{D}^{\mathbb{IM}} = \mathcal{D}^{\mathbb{ID}}|_{\mathit{GPCCS}}$. ■

Chapter 6

Deciding bisimilarity and similarity

Bisimulation and simulation relations have proved very useful for the design and abstraction. For mechanised purposes, the development of methods for showing that two processes are bisimilar or related via simulation and the efficiency of such methods is a crucial aspect. Several techniques for checking bisimulation equivalence for fully probabilistic processes have been proposed; see [JoSm90, BBS92, LaSk92] for axiomatic methods and [HuTi92] for a decision procedure. The issue of axiomatizations for bisimulation and simulation in probabilistic systems with non-determinism has been considered in [HaJo90, Hans91, Yi94].[1] As far as the author knows, [Bai96] and the forthcoming work [PSS98, BSV98] are the first attempts to formulate algorithmic methods that deal with bisimulation and simulation for concurrent probabilistic processes. In this chapter we present a revised version of [Bai96] where algorithms for deciding bisimulation equivalence and for computing the simulation preorder in finite concurrent probabilistic systems are proposed. Moreover, we show that a variant of the method for simulation is applicable for fully probabilistic systems and the "satisfaction relation" of [JoLa91].

Deciding bisimulation equivalence: Huynh & Tian [HuTi92] presented an $\mathcal{O}(k \log n)$ algorithm for computing the bisimulation equivalence classes in finite fully probabilistic systems where n is the number of states and k the number of non-zero entries in the transition probability matrix $(\mathbf{P}(s,a,t))_{s,a,t}$. The method of [HuTi92] is a modification of the the partitioning/splitter-technique à la [KaSm83, PaTa87] which performs a sequence of refinement steps that replace a given partition $\mathcal{X}$ by a finer one, eventually resulting in the set of bisimulation equivalence classes. As in the non-probabilistic case, the underlying refinement operation *Refine*$(\mathcal{X})$ is based on a *splitter* of the current partition $\mathcal{X}$. This partition/splitter-technique also works for reactive systems but fails for general concurrent probabilistic systems. Our method for deciding bisimulation equivalence works – as in the non-probabilistic or fully probabilistic case – with a partitioning technique but avoids the use of splitters. It runs in time $\mathcal{O}(mn(\log m + \log n))$ where m is the number of transitions and n the number of states. In various applications, e.g. when the system arises from the interleaving of l "sequential" probabilistic systems, we may suppose that the number m of transitions is polynomial in n. In these cases we obtain the time complexity $\mathcal{O}(mn \log n)$.

[1] It should be mentioned that [Yi94] deals with a variant of action-labelled stratified systems where intervals of probabilities – rather than precise probabilities – are used. The underlying notion of a simulation is different from the one proposed by [SeLy94].

Computing the simulation preorder: The schema for computing the simulation preorder of a finite probabilistic system is the same as in the non-probabilistic case [HHK95]. We start with the relation $R = S \times S$ and successively remove those pairs (s, s') from R for which there is a step of s that cannot be "simulated" by a step of s'. In the probabilistic case, the test whether a step "simulates" another one amounts deciding whether two distributions μ, μ' are related via a weight function with respect to the current relation R, i.e. whether $\mu \preceq_R \mu'$ (see page 30). We show that the question whether $\mu \preceq_R \mu'$ can be reduced to a *maximum flow problem* in a suitable chosen network which yields an $\mathcal{O}((mn^6 + m^2n^3)/\log n))$ algorithm for computing the simulation preorder when applying the method of [CHM90] for solving the maximum flow problem.

Organization of that chapter: Section 6.1 presents an algorithm for deciding bisimulation equivalence where we first recall the results by Huynh & Tian [HuTi92] for fully probabilistic systems and then deal with concurrent probabilistic systems. Section 6.2 gives an algorithm for computing the simulation preorder where we first consider concurrent probabilistic systems (Section 6.2.2) and then the fully probabilistic case (Section 6.2.3). We also show how our method for computing the simulation preorder can be modified for the "satisfaction relation" introduced by Jonsson & Larsen [JoLa91].

In this chapter, we need the definitions of bisimulation and simulation (see Section 3.4, page 53 ff) where the latter uses the definition of weight functions for distributions (see Section 2.2, page 30). Moreover, we often use the notations for partitions as explained in Section 2.1 (page 29). For the computation of certain equivalence classes we propose the use of ordered balanced trees. Our notations can be found in Section 12.2 (page 314). Throughout this chapter, we deal with finite and action-labelled systems.

6.1 Computing the bisimulation equivalence classes

The main idea for computing the probabilistic bisimulation equivalence classes is the use of a partitioning technique as proposed by Kanellakis & Smolka [KaSm83] (and its improvement by Paige & Tarjan [PaTa87]) for the non-probabilistic case. We start with the trivial partition $\mathcal{X} = \{S\}$ and then successively refine $\mathcal{X}$ by splitting the blocks B of $\mathcal{X}$ into subblocks, eventually resulting in the bisimulation equivalence classes. This schema is sketched in Figure 6.1 on page 131.

In the non-probabilistic case, the refinement operator $\mathit{Refine}(\mathcal{X})$ depends on a "splitter" of $\mathcal{X}$. Intuitively, a splitter denotes a pair (a, C) consisting of an action a and a block $C \in \mathcal{X}$ that prevents the induced equivalence $R_{\mathcal{X}}$ to fulfill the condition of a bisimulation; that is, a splitter is a pair $(a, C) \in \mathit{Act} \times \mathcal{X}$ such that there are states s, $s' \in S$ that are identified in $\mathcal{X}$ (i.e. s and s' belong to the same block of $\mathcal{X}$) and where $s \xrightarrow{a} C$ while $s' \not\xrightarrow{a} C$.[2] For (a, C) to be a splitter of $\mathcal{X}$, the refinement operator $\mathit{Refine}(\mathcal{X}) = \mathit{Refine}(\mathcal{X}, a, C)$ devides each block $B \in \mathcal{X}$ into the subblocks $B_{(a,C)} = \{s \in B : s \xrightarrow{a} C\}$ and $B \setminus B_{(a,C)}$ and returns the partition

$$\{B_{(a,C)}, B \setminus B_{(a,C)} : B \in \mathcal{X}\} \setminus \{\emptyset\}.$$

[2] Here, we write $t \xrightarrow{a} C$ iff $t \xrightarrow{a} u$ for some $u \in C$.

Computing the bisimulation equivalence classes

Input: a finite (non-probabilistic or probabilistic) system with state space S

Output: the set $S/\sim$ of bisimulation equivalence classes

Method:

$\mathcal{X} = \{S\}$;

While $\mathcal{X}$ can be refined do $\mathcal{X} := \mathit{Refine}(\mathcal{X})$;

Return $\mathcal{X}$.

Figure 6.1: Schema for computing the bisimulation equivalence classes

Clearly, if $\mathcal{X}$ is coarser than $S/\sim$ then $s \not\sim s'$ for all $s \in B_{(a,C)}$ and $s' \in B \setminus B_{(a,C)}$. Hence, $\mathit{Refine}(\mathcal{X}, a, C)$ is again coarser than $S/\mathcal{X}$ and strict finer than $\mathcal{X}$ (provided that (a, C) is a splitter of $\mathcal{X}$). Thus, after at most $|S|$ refinement steps the current partition coincides with $S/\sim$. This method can be implemented in time $\mathcal{O}(m \log n)$ where n is the number of states and m the number of transitions (i.e. the size of $\longrightarrow$) [PaTa87] (see also [Fern89]).

6.1.1 The fully probabilistic case

The partitioning/splitter method is adapted in [HuTi92] for fully probabilistic transition systems, thus yielding an $\mathcal{O}(k \log n)$ algorithm for deciding bisimilarity in fully probabilistic transition systems where n is the number of states and k the number of tuples (s, a, t) such that $\mathbf{P}(s, a, t) > 0$. In the worst case, we have $k = |\mathit{Act}| \cdot n^2$. If we suppose Act to be fixed then we obtain the time complexity $\mathcal{O}(n^2 \log n)$ for deciding bisimulation equivalence in fully probabilistic systems. Moreover, we saw in Theorem 3.4.19 (page 61) that simulation equivalence $\sim_{\text{sim}}$ and bisimulation equivalence $\sim$ coincide for fully probabilistic systems. Thus:

Theorem 6.1.1 (cf. [HuTi92]) *In fully probabilistic systems, bisimulation and simulation equivalence can be decided in time $\mathcal{O}(n^2 \log n)$ and space $\mathcal{O}(n^2)$ where n is the number of states.*

The basic idea is for the fully probabilistic case is to define a splitter of a partition $\mathcal{X}$ to be a pair $(a, C) \in \mathit{Act} \times \mathcal{X}$ such that $\mathbf{P}(s, a, C) \neq \mathbf{P}(s', a, C)$ for some states s, s' that are identified in $\mathcal{X}$. Then, the refinement operator according to the splitter (a, C) replaces each block $B \in \mathcal{X}$ by the subblocks $B/\simeq_{(a,C)}$ where $s \simeq_{(a,C)} s'$ iff $\mathbf{P}(s, a, C) = \mathbf{P}(s', a, C)$.

6.1.2 The concurrent case

As mentioned in [HuTi92], the partitioning/splitter technique can easily be modified for reactive systems (with the same time complexity $\mathcal{O}(n^2 \log n)$). In the general case, i.e. dealing with concurrent probabilistic systems, the splitter technique fails (see Example 6.1.4,

Computing the bisimulation equivalence classes in reactive systems

Input: a finite reactive system $(S, Act, Steps)$

Output: the set $S/\sim$ of bisimulation equivalence classes

Method:

$\mathcal{X} = \{S\}$;

While there exists a splitter (a, C) of $\mathcal{X}$ do $\mathcal{X} := Refine(\mathcal{X}, a, C)$;

Return $\mathcal{X}$.

Figure 6.2: Partioning/splitter technique

page 133). We propose a method that can be implemented in time $\mathcal{O}(mn(\log m + \log n))$ where n is the number of states and m the number of transitions.

In what follows, we fix a finite set Act of actions and a finite action-labelled concurrent probabilistic system $(S, Act, Steps)$.

Definition 6.1.2 [The splitter-based refinement operator] *If $\mathcal{X}$ is a partition of S and $a \in Act$, B, $C \in \mathcal{X}$ then*

$$Refine(B, a, C) \;=\; B/\simeq_{(a,C)}$$

where the equivalence relation $\simeq_{(a,C)} = \trianglelefteq_{(a,C)} \cap \trianglelefteq_{(a,C)}^{-1}$ is the kernel of the relation $\trianglelefteq_{(a,C)} \subseteq B \times B$ which is given by:

$$s \trianglelefteq_{(a,C)} s' \text{ iff whenever } s \xrightarrow{a} \mu \text{ then there exists } s' \xrightarrow{a} \mu' \text{ with } \mu[C] = \mu'[C].$$

For $\mathcal{X}$ to be a partition of S, a splitter *of $\mathcal{X}$ is a pair $(a, C) \in Act \times \mathcal{X}$ such that $Refine(B, a, C) \neq \{B\}$ for some $B \in \mathcal{X}$.*

The method of [PaTa87] (or the method of [HuTi92] for fully probabilistic systems) modified for reactive systems is sketched in Figure 6.2 on page 132. For the implementation of this method we propose the use of a queue Q of possible splitters, initially containing the pairs (a, S), $a \in Act$. As long as Q is nonempty, we take the first element (a, C) of Q and remove (a, C) from Q. For each $B \in \mathcal{X}$, we compute the probabilities

$$p_s \;=\; \begin{cases} 0 & : \text{ if } Steps_a(s) = \emptyset \\ \mu[C] & : \text{ if } Steps_a(s) = \{\mu\} \end{cases}, \quad s \in B.$$

We construct an ordered balanced tree $Tree_{(B,a,C)}$ for the values p_s, $s \in B$, with additional labels $v.states$ for each node v such that finally $v.states = \{s \in B : v.key = p_s\}$.[3] The nodes in the final tree represent $Refine(B, a, C)$; more precisely,

$$Refine(B, a, C) \;=\; \{v.states : v \text{ is a node in } Tree_{(B,a,C)}\}.$$

If $Refine(B, a, C) \neq \{B\}$ then for each $B' \in Refine(B, a, C)$ but one of the largest we add the pairs (b, B'), $b \in Act$, to the end of Q.[4] Using an implementation similar to the one in

[3] See Section 12.2, page 314, for the notations that we use for ordered balanced trees.

[4] By the largest blocks we mean those blocks $B' \in Refine(B, a, C)$ where $|B'|$ is maximal.

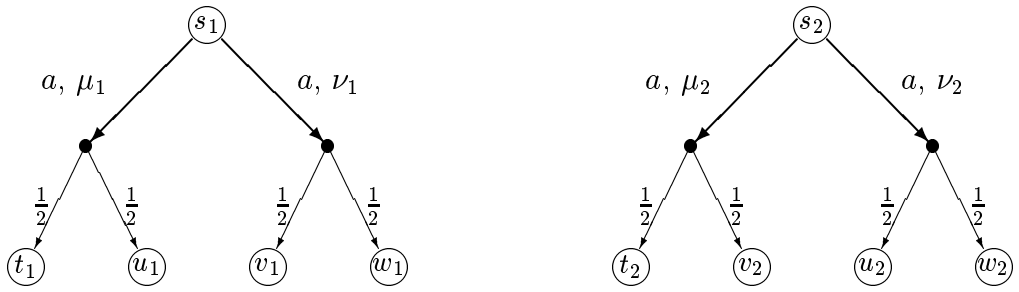


Figure 6.3: $s_1 \not\sim s_2$, but s_1 and s_2 cannot be distinguished by splitters.

[PaTa87] we obtain the time complexity $\mathcal{O}(n^2 \log n)$. For reactive systems, bisimulation equivalence $\sim$ and simulation equivalence $\sim_{\text{sim}}$ are the same (Theorem 3.4.15, page 59). Hence:

Theorem 6.1.3 *In reactive systems, bisimulation and simulation equivalence can be decided in time $\mathcal{O}(n^2 \log n)$ and space $\mathcal{O}(n^2)$ where n is the number of states.*

Before we present our method for computing the bisimulation equivalence classes in arbitrary finite action-labelled concurrent probabilistic systems we give an example which explains why the splitter technique fails in the general case.

Example 6.1.4 We consider a system as shown in Figure 6.3 (page 133) where we suppose that $t_1 \sim t_2$, $u_1 \sim u_2$, $v_1 \sim v_2$, $w_1 \sim w_2$ and that t_1, u_1, v_1, w_1 are pairwise non-bisimilar.[5] Then, $s_1 \not\sim s_2$. On the other hand, s_1, s_2 cannot be distinguished by splitters.[6] Thus, the algorithm for deciding bisimilarity based on the splitter technique would return that s_1 and s_2 are bisimilar. ■

For the general case, we maintain the schema sketched in Figure 6.1 (page 131) but use a refinement operator that does not depend on a splitter. In each refinement step, we replace each block B of the given partition $\mathcal{X}$ by the equivalence classes of B with respect to the equivalence relation $\equiv_{\mathcal{X}}$ which identifies exactly those states s, $s' \in B$ such that for each transition $s \xrightarrow{a} \mu$ there exists a transition $s' \xrightarrow{a} \mu'$ with $\mu[C] = \mu'[C]$ for all $C \in \mathcal{X}$.

Notation 6.1.5 [The vector $\mu[\mathcal{X}]$ and the equivalence $\equiv_{\mathcal{X}}$] *Let $\mathcal{X}$ be a partition of S. $\mu[\mathcal{X}]$ denotes the vector $(\mu[B])_{B \in \mathcal{X}}$. $\mathcal{X}$ is associated with the equivalence relation $\equiv_{\mathcal{X}}$ on S that is given by:*

$$s \equiv_{\mathcal{X}} s' \quad \textit{iff} \quad \{(a, \mu[\mathcal{X}]) : s \xrightarrow{a} \mu\} \;=\; \{(a, \mu'[\mathcal{X}]) : s' \xrightarrow{a} \mu'\}.$$

Definition 6.1.6 [The refinement operator] *We define*

$$\mathit{Refine}(\mathcal{X}) \;=\; \bigcup_{B \in \mathcal{X}} B/\equiv_{\mathcal{X}} .$$

Lemma 6.1.7 *Let $\mathcal{X}$ be a partition of S which is coarser than $S/\sim$. Then:*

(a) $\mathit{Refine}(\mathcal{X})$ is a partition which is coarser than $S/\sim$.

[5] The outgoing transitions of the states t_i, u_i, v_i, w_i are omitted in the picture.

[6] I.e. $s_1 \simeq_{(b,C)} s_2$ for all actions b and all blocks C of a partition $\mathcal{X}$ of S that is coarser than $S/\sim$.

(b) If $\mathit{Refine}(\mathcal{X}) = \mathcal{X}$ *then* $\mathcal{X} = S/\sim$.

Proof: easy verification. ■

Lemma 6.1.7 ensures the total correctness of our schema for deciding bisimilarity sketched in Figure 6.1 (page 131). We state our main result:

Theorem 6.1.8 *In concurrent probabilistic systems, bisimulation equivalence can be decided in time* $\mathcal{O}(mn(\log m + \log n))$ *and space* $\mathcal{O}(mn)$ *where* n *is the number of states and* m *the number of transitions.*

Remark 6.1.9 In many situations, m is polynomial in n. For example, when the system arises from the interleaving of l "sequential" probabilistic systems then $m \leq l \cdot n$. In these cases, the time complexity for deciding bisimulation equivalence is $\mathcal{O}(mn \log n)$. ■

In the remainder of this section we decribe how to implement the algorithm sketched in Figure 6.1 (page 131) for concurrent probabilistic systems to obtain the desired time and space complexity where we use the refinement operator $\mathit{Refine}(\mathcal{X})$ of Definition 6.1.6 (page 133). The main idea for the implementation of the operator $\mathit{Refine}(\mathcal{X})$ is first to compute the set of "step classes" with respect to $\mathcal{X}$ from which the sets $B/\equiv_{\mathcal{X}}$, $B \in \mathcal{X}$, can be derived.

Definition 6.1.10 [Step classes] *Let* $\mathcal{X}$ *be a partition,* $B \in \mathcal{X}$ *and* $a \in \mathit{Act}$. *Then,*

$$\mathit{Steps}_a(B) = \bigcup_{s \in B} \mathit{Steps}_a(s).$$

Two distributions μ, $\mu' \in \mathit{Steps}_a(B)$ *are called* $\mathcal{X}$-equivalent *(denoted* $\mu \equiv_{\mathcal{X}} \mu'$*) iff* $\mu[\mathcal{X}] = \mu'[\mathcal{X}]$. *A* step class *of* B *with respect to* $\mathcal{X}$ *is a pair* (a, h) *consisting of an action* $a \in \mathit{Act}$ *and a function* $h : B \to 2^{\mathit{Distr}(S)}$ *with* $h(s) \subseteq \mathit{Steps}_a(s)$ *for all* $s \in B$ *such that*

$$\bigcup_{s \in B} h(s) \in \mathit{Steps}_a(B)/\equiv_{\mathcal{X}}.$$

For $B' \in \mathit{Refine}(\mathcal{X})$, *we define a step class of* B' *with respect to* $\mathcal{X}$ *to be a pair* (a, h') *where* $a \in \mathit{Act}$ *and* $h' = h|_{B'}$ *for some step class* (a, h) *of* B *with respect to* $\mathcal{X}$ *where* B *denotes the unique block in* $\mathcal{X}$ *which contains* B'. $\mathit{StepCl}_{\mathcal{X}}(\cdot)$ *denotes the set of step classes of* $(\cdot)$ *with respect to* $\mathcal{X}$.

Clearly, if $B \in \mathcal{X}$ and s, $s' \in B$ then $s \equiv_{\mathcal{X}} s'$ iff, for each step class (a, h) of B with respect to $\mathcal{X}$, $h(s) \neq \emptyset$ iff $h(s') \neq \emptyset$. Let $\mathcal{X}$ be the current partition for which we want to compute $\mathit{Refine}(\mathcal{X})$ and let $\mathcal{X}_{old}$ be the partition in the previous refinement step.[7] For each step class (a, h') of a subblock $B' \in \mathit{Refine}(\mathcal{X})$ of B (with respect to $\mathcal{X}$) there is step class (a, h_{old}) of B with respect to $\mathcal{X}_{old}$ such that $h'(s) \subseteq h_{old}(s)$ for all $s \in B'$. More precisely, the subblocks $B' \in \mathit{Refine}(\mathcal{X})$ of B and their step classes with respect to $\mathcal{X}$ can be derived from the step classes of B with respect to $\mathcal{X}_{old}$ as follows. For (a, h_{old}) to be a step class of B with respect to $\mathcal{X}_{old}$, let $\mathcal{C}_{(a,h_{old})}$ be the set of all tuples (a, L, h) where

- $\emptyset \neq L \subseteq B$

[7] I.e. we assume that the current partition $\mathcal{X}$ is $\mathit{Refine}(\mathcal{X}_{old})$.

- $h : L \to 2^{Distr(S)}$ is a function with $\emptyset \neq h(s) \subseteq h_{old}(s)$ for all $s \in L$
- there exists a real vector $\mathbf{p} = (p_C)_{C \in \mathcal{X}}$ such that
  - $\mu[\mathcal{X}] = \mathbf{p}$ for all $\mu \in h(s)$, $s \in L$
  - If $s \in B \setminus L$ then $h(s) \cap \{\mu \in Distr(S) : \mu[\mathcal{X}] = \mathbf{p}\} = \emptyset$.
  - If $s \in L$ and $\mu \in h_{old}(s) \setminus h(s)$ then $\mu[\mathcal{X}] \neq \mathbf{p}$.

For s, $s' \in B$ we have $s \equiv_{\mathcal{X}} s'$ iff

$$\forall (a, h_{old}) \in StepCl_{\mathcal{X}_{old}}(B)\ \forall (a, L, h) \in \mathcal{C}_{(a,h_{old})}\ [\ s \in L \text{ iff } s' \in L\ ].$$

We reformulate this observation as follows. Let $(a_1, L_1, h_1), \ldots, (a_r, L_r, h_r)$ be an enumeration of

$$\bigcup_{(a,h_{old}) \in StepCl_{\mathcal{X}_{old}}(B)} \mathcal{C}_{(a,h_{old})}.$$

Let $L_i^1 = L_i$, $L_i^0 = B \setminus L_i$ and $L^{\overline{b}} = L_1^{b_1} \cap L_2^{b_2} \cap \ldots \cap L_r^{b_r}$ if $\overline{b} = (b_1, \ldots, b_r) \in \{0,1\}^r$. Then,

$$B/\equiv_{\mathcal{X}} \ = \ \{L^{\overline{b}} : \overline{b} \in \{0,1\}^r\} \setminus \{\emptyset\}.$$

Moreover, for the new subblock $L^{(b_1,\ldots,b_r)}$ (where $\overline{b} = (b_1, \ldots, b_r)$) we have

$$StepCl_{\mathcal{X}}(L^{(b_1,\ldots,b_r)}) \ = \ \{(a_i, h_i') : i = 1, \ldots, r, b_i = 1\}$$

where $h_i' : L^{(b_1,\ldots,b_r)} \to 2^{Distr(S)}$ is given by $h_i'(s) = h_i(s)$. (I.e. $h_i' = h_i|_{L^{(b_1,\ldots,b_r)}}$ is the restriction of h_i on the states of $L^{(b_1,\ldots,b_r)}$.) Clearly, for computing the sets $L^{\overline{b}}$ and their step classes, the tuples (a_i, L_i, h_i) where $L_i = B$ and $(a_i, h_i) \in StepCl_{\mathcal{X}}(B)$ are not of importance. Therefore, we divide the tuples (a_i, L_i, h_i) into two classes:

- $OldCl_{\mathcal{X}}(B)$ denotes the set of tuples (a_i, L_i, h_i) that represent an "old step class" (i.e. $L_i = B$ and $(a_i, h_i) \in StepCl_{\mathcal{X}_{old}}(B)$).

$$OldCl_{\mathcal{X}}(B) \ = \ \left\{(a, B, h_{old}) \in \mathcal{C}_{(a,h_{old})} : (a, h_{old}) \in StepCl_{\mathcal{X}}(B)\right\}$$

- $NewCl_{\mathcal{X}}(B)$ the set of tuples (a_i, L_i, h_i) that represent a "new step class" (i.e. either $L_i \neq B$ or $(a_i, h_i) \notin StepCl_{\mathcal{X}_{old}}(B)$).

$$NewCl_{\mathcal{X}}(B) \ = \ \left\{(a, L, h) \in \mathcal{C}_{(a,h_{old})} : (a, h_{old}) \in StepCl_{\mathcal{X}}(B), (L, h) \neq (B, h_{old})\right\}.$$

For the test whether $\mu[\mathcal{X}] = \mu'[\mathcal{X}]$ we use the following facts. Let μ, $\mu' \in Distr(S)$ such that $\mu[\mathcal{X}_{old}] = \mu'[\mathcal{X}_{old}]$ and $B \in \mathcal{X}$, $B_{old} \in \mathcal{X}_{old}$.

(1) If $B \in \mathcal{X}_{old}$ (i.e. B is a block that has not been refined in the last refinement step) then $\mu[B] = \mu'[B]$.

(2) If B_{old} is refined into the subblocks $B_1, \ldots, B_{k+1} \in \mathcal{X}$ then
$$\mu[B_i] = \mu'[B_i],\ i = 1, \ldots, k+1 \quad \text{iff} \quad \mu[B_i] = \mu'[B_i],\ i = 1, \ldots, k.$$

Because of (1), we only have to consider the "new" blocks, i.e. the blocks $B \in \mathcal{X} \setminus \mathcal{X}_{old}$. Because of (2), for computing $Refine(\mathcal{X})$, it suffices to consider for each block $B_{old} \in \mathcal{X}_{old} \setminus \mathcal{X}$ all subblocks $B \in \mathcal{X}$ of B_{old} but one of the largest. These observations (1) and (2) lead to the use of a set

$$New \subseteq \mathcal{X} \setminus \mathcal{X}_{old}$$

such that:

Computing the bisimulation equivalence classes in concurrent probabilistic systems

Input: a finite action-labelled concurrent probabilistic system $(S, Act, Steps)$

Output: the set $S/\sim$ of bisimulation equivalence classes

Method:

(0) compute $\mathcal{X}_{init}$, New_{init} and $StepCl_{\mathcal{X}_{trivial}}(B)$ for all $B \in \mathcal{X}_{init}$

(1) $\mathcal{X} := \mathcal{X}_{init}$; $New := New_{init}$;

(2) While $New \neq \emptyset$ do begin

  (2.1) $New' := \emptyset$ and $\mathcal{X}' := \emptyset$;

  (2.2) For all $B \in \mathcal{X}$ do

    (2.2.1) compute $NewCl_{\mathcal{X}}(B)$ and $OldCl_{\mathcal{X}}(B)$ with the method of Figure 6.6 (page 142);

    (2.2.2) compute $B/\equiv_{\mathcal{X}}$, New_B and $StepCl_{\mathcal{X}}(C)$ for $C \in B/\equiv_{\mathcal{X}}$ with the method explained on page 139;

    (2.2.3) $New' := New \cup New_B$ and $\mathcal{X}' := \mathcal{X}' \cup B/\equiv_{\mathcal{X}}$;

  (2.3) $\mathcal{X} := \mathcal{X}'$; $New := New'$;

(3) Return $\mathcal{X}$.

Figure 6.4: Algorithm for deciding bisimulation equivalence in concurrent systems

(*) If $B_{old} \in \mathcal{X}_{old} \setminus \mathcal{X}$ (i.e. the refinement operation $Refine(\mathcal{X}_{old})$ splits the block B_{old} into two or more subblocks) then there exist $k \geq 1$ and $B_1, \ldots, B_k \in New$ such that
- $B_1, \ldots, B_k \subseteq B_{old}$
- $B_{old} \setminus (B_1 \cup \ldots \cup B_k) \in \mathcal{X} \setminus New$
- $|B_{old} \setminus (B_1 \cup \ldots \cup B_k)| \geq |B_i|$, $i = 1, \ldots, k$.

Then (by (1) and (2)), if μ, $\mu' \in Distr(S)$ such that $\mu[\mathcal{X}_{old}] = \mu'[\mathcal{X}_{old}]$ then

$$\mu[\mathcal{X}] = \mu'[\mathcal{X}] \text{ iff } \mu[C] = \mu'[C] \text{ for all } C \in New.$$

The algorithm for computing the bisimulation equivalence classes is shown in Figure 6.4, page 136.

Initialization (step (0) in Figure 6.4 (page 136)): We skip the first refinement step and start with the partition $\mathcal{X}_{init} = S/\equiv$ where $s \equiv s'$ iff $act(s) = act(s')$.[8] $\mathcal{X}_{init}$ can be computed with the following method. We choose with a fixed enumeration $a_1, \ldots, a_k$ of Act and construct a binary tree $Tree$ by successively inserting nodes and edges. Each node v is labelled by

- its depth $v.depth$ in $Tree$,
- a subset $v.actions$ of Act,
- the names $v.left$ and $v.right$ of the left and right son of v in $Tree$.

[8] I.e. we deal with the initial partition $\mathcal{X}_{init} = Refine(\{S\})$ rather than trivial partition $\{S\}$.

In the case where v does not have a left (right) son $v.left$ ($v.right$) is undefined ($\perp$). Each node v of depth k is a leaf and is additionally labelled by

- a subset $v.states$ of S,
- a natural number $v.counter$ that counts the number of elements in $v.states$.

Initially, *Tree* consists of its root v_0 where $v_0.depth = 0$, $v_0.actions = \emptyset$, $v_0.left = v_0.right = \perp$. Then, for each state $s \in S$, we traverse the tree starting in the root v_0. If we have reached a node v with $v.depth = i < k$ then

- if $Steps_{a_{i+1}}(s) \neq \emptyset$ then
  - if $v.left \neq \perp$ then we go to $v.left$
  - if $v.left = \perp$ then we create a node w with $w.depth = i+1$, $w.actions = v.actions \cup \{a_{i+1}\}$ and $w.left = w.right = \perp$, put $v.left := w$ and go to w
- if $Steps_{a_{i+1}}(s) = \emptyset$ then
  - if $v.right \neq \perp$ then we go to $v.right$
  - if $v.right = \perp$ then we create a node w with $w.depth = i+1$, $w.actions = v.actions$ and $w.left = w.right = \perp$, put $v.right := w$ and go to w.

In both cases, when creating a leaf w (i.e. when $v.depth = i = k-1$), we put $w.states := \emptyset$ and $w.counter := 0$. If we have reached a leaf v (i.e. if $v.depth = k$) then we insert s into $v.states$ (i.e. we put $v.states := v.states \cup \{s\}$) and increment $v.counter$. Then, the leaves of the final tree represent the blocks of $\mathcal{X}_{init}$, i.e.

$$\mathcal{X}_{init} \;=\; \{v.states : v \text{ is a leaf in } Tree\}.$$

Moreover, we use the components $v.counter$ to select some of the largest initial blocks, i.e. we choose some leaf v where $v.counter$ is maximal and put

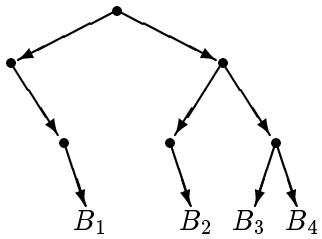
$$New_{init} \;=\; \mathcal{X}_{init} \setminus \{v.states\}.$$

The initial step classes are taken with respect to the "previous" partition $\mathcal{X}_{trivial} = \{S\}$ and are obtained by

$$StepCl_{\mathcal{X}_{trivial}}(v.states) \;=\; \{(a, Steps_a|_{v.states}) : a \in v.actions\}\,.$$

Note that $\mu \;\equiv_{\mathcal{X}_{trivial}}\; \mu'$ for all μ, $\mu' \in Distr(S)$.

Example 6.1.11 We consider the system of Figure 6.5 (page 138) and compute the initial partition $\mathcal{X}_{init}$. We use the ordering $a_1 = a$, $a_2 = b$, $a_3 = c$ of Act and construct the following tree.



$$\begin{array}{lcl} B_1 & = & \{s_1, s_2, s_3, s_4\} \\ B_2 & = & \{t_1, t_2, t_3, t_4\} \\ B_3 & = & \{w\} \\ B_4 & = & \{u_1, u_1', u_2, u_2', u_2'', v, v_1, v_2, v_3, v_4\} \end{array}$$

We obtain $\mathcal{X}_{init} = \{B_1, B_2, B_3, B_4\}$, $New_{init} = \{B_1, B_2, B_3\}$ and

$$\begin{array}{lcll} StepCl_{\mathcal{X}_{trivial}}(B_1) & = & \{(a, h_1)\} & h_1(s_i) = \{\mu^1_{u_i}, \mu_i\},\ i = 1, 2, \\ & & & h_1(s_3) = \{\mu^1_{t_3}\},\ h_1(s_4) = \{\mu_4\} \\ StepCl_{\mathcal{X}_{trivial}}(B_2) & = & \{(b, h_2)\} & h_2(t_i) = \{\mu^1_{v_i}\},\ i = 1, 2, 3, 4 \\ StepCl_{\mathcal{X}_{trivial}}(B_3) & = & \{(c, h_3)\} & h_3(w) = \{\mu^1_v\} \end{array}$$

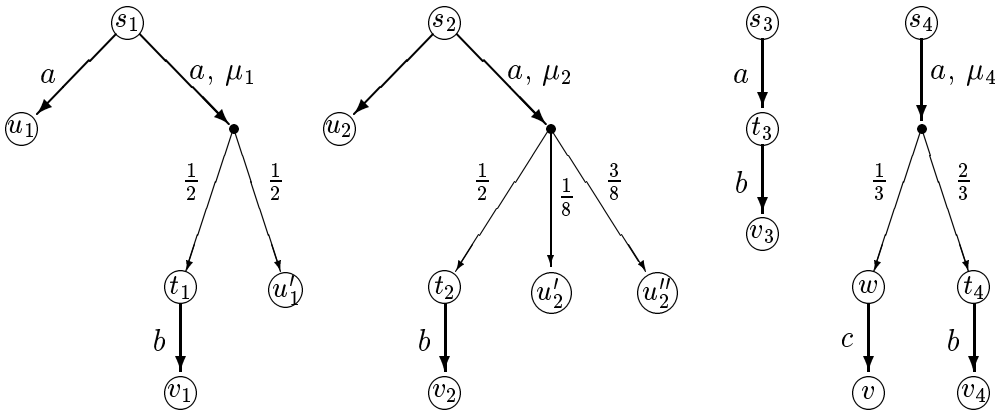


Figure 6.5:

and $StepCl_{\mathcal{X}_{trivial}}(B_4) = \emptyset$. ■

The refinement step (step (2) in Figure 6.4 (page 136)): As before, let $\mathcal{X} = Refine(\mathcal{X}_{old})$ be the current partition and $\mathcal{X}_{old}$ the partition of the previous refinement step. Moreover, *New* is a proper subset of $\mathcal{X} \setminus \mathcal{X}_{old}$ that satisfies condition (*) (see page 136).

Step (2.2.1) in Figure 6.4 (page 136): Let $C_1, \ldots, C_l$ be an enumeration of *New*. For each $B \in \mathcal{X}$, we compute $NewCl_{\mathcal{X}}(B)$ and $OldCl_{\mathcal{X}}(B)$ with the method sketched in Figure 6.6 (page 142). We use a set Q of tuples $\langle j, b, a, L, h\rangle$ where $0 \leq j \leq l$, $b \in \{old, new\}$, $a \in Act$, $L \subseteq B$ and $h : L \to 2^{Distr(S)}$ is a function with $h(s) \subseteq Steps_a(s)$ such that:

- If $b = old$ then $L = B$ and $(a, h) \in StepCl_{\mathcal{X}_{old}}(B)$
- For all s, $s' \in L$ and $\mu \in h(s)$, $\mu' \in h(s')$, $\mu[C_i] = \mu'[C_i]$, $i = 1, \ldots, j$.

The elements of Q can be viewed as nodes of a forest (a collection of trees) where the roots are the nodes of the form $\langle 0, old, a, B, h_{old}\rangle$ with $(a, h_{old}) \in StepCl_{\mathcal{X}_{old}}(B)$. The first component j of a node $\langle j, b, a, L, h\rangle$ stands for the depth of that node. The sons of the node $\langle j, b, a, L, h\rangle$ are of the form $\langle j+1, b', a, L', h'\rangle$ where $L' \subseteq L$ and $h'(s) \subseteq h(s)$ for all $s \in L'$. More precisely, if $\langle j+1, b_i, a, L_i, h_i\rangle$, $i = 1, \ldots, r$, are the sons of $\langle j, b, a, L, h\rangle$ then we have:

$$b_1 = \ldots = b_r = \begin{cases} new & : \text{ if } r \geq 2 \\ b & : \text{ if } r = 1. \end{cases}$$

Moreover, if $H = \bigcup_{s \in L} h(s)$ and $H_i = \bigcup_{s \in L_i} h_i(s)$, $i = 1, \ldots, r$, then

$$(**) \qquad \{H_1, \ldots, H_r\} \;=\; H/\equiv_{\mathcal{X}}^{j+1} \text{ where } \mu \equiv_{\mathcal{X}}^{j+1} \mu' \text{ iff } \mu[C_{j+1}] = \mu'[C_{j+1}].$$

The nodes on the path from the root to a node of the form $\langle j, old, a, B, h\rangle$ are the nodes $\langle i, old, a, B, h\rangle$, $i = 0, 1, \ldots, j$. The leaves are

- all nodes of the form $\langle l, b, a, L, h\rangle$ (i.e. all nodes of depth l where $l = |New|$)
- all nodes $\langle j, b, a, L, h\rangle$ where $|L| = 1$.[9]

[9] For these nodes, the possible splittings of the step classes of the set L are not of interest since we have found a bisimulation equivalence class consisting of a single state.

A leaf of the form $\langle l, old, a, B, h\rangle$ represents an element of $OldCl_{\mathcal{X}}(B)$ (because $(a,h) \in StepCl_{\mathcal{X}_{old}}(B)$). This case corresponds to step (1.7) where the ordered balanced tree T constructed in step (1.2) consists of its root. Leaves of the form $\langle j, new, a, L, h\rangle$ stand for "new" step classes.

Step (2.2.2) in Figure 6.4 (page 136): Having obtained the sets $NewCl_{\mathcal{X}}(B)$ and $OldCl_{\mathcal{X}}(B)$, we derive $B/\equiv_{\mathcal{X}}$ as described on page 135 where we only consider the tuples $(a,L,h) \in NewCl_{\mathcal{X}}(B)$ (rather than all tuples $(a,L,h) \in \bigcup_{(a,h_{old})} \mathcal{C}_{(a,h_{old})}$). We choose an enumeration $(a_1,L_1,h_1),\ldots,(a_r,L_r,h_r)$ of $NewCl_{\mathcal{X}}(B)$ and construct a binary tree $Tree_B$ such that each leaf v of depth r is labelled by the set $v.states = L_1^v \cap \ldots \cap L_r^v$ where $v_0v_1\ldots v_r = v$ is the path from the root of $Tree_B$ to v and

$$L_i^v = \begin{cases} L_i & : \text{ if } v_i \text{ is the left son of } v_{i-1} \\ B \setminus L_i & : \text{ if } v_i \text{ is the right son of } v_{i-1}. \end{cases}$$

The construction of this tree is similar to the initialization step. We start with the tree consisting of its root. Then, for each state $s \in B$, we traverse the tree starting in its root. If we reached a node v of depth $i < r$ then we go to the left son $v.left$ if $s \in L_{i+1}$ and to the right son $v.right$ if $s \notin L_{i+1}$ (possibly inserting the left or right son if necessary). If he have reached a leaf (a node v of depth r) then we insert s into the set $v.states$. Thus, $B/\equiv_{\mathcal{X}} = \{v.states : v \text{ is a leaf}\}$. As in the initialization step, we use an auxiliary component $v.counter$ for the leaves such that $v.counter = |v.states|$. Finally, we choose some leaf w in $Tree_B$ where $w.counter$ is maximal and define

$$New_B = \{v.states : v \text{ is a leaf in } Tree_B,\ v \neq w\}.$$

For all blocks $C \in B/\equiv_{\mathcal{X}}$, the step classes with respect to $\mathcal{X}$ (the set $StepCl_{\mathcal{X}}(C)$) are the pairs $(a, h|_C)$ where $(a,L,h) \in NewCl_{\mathcal{X}}(B) \cup OldCl_{\mathcal{X}}(B)$ for some $L \subseteq B$ with $C \subseteq L$.

Example 6.1.12 We consider the system of Figure 6.5 (page 138) and compute the bisimulation equivalence classes. The initialization step yields

$$\mathcal{X}_{init} = \{B_1,B_2,B_3,B_4\}, \quad New_{init} = \{B_1,B_2,B_3\}$$

$StepCl_{\mathcal{X}_{trivial}}(B_i) = \{(a_i,h_i)\}$, $i = 1,2,3$, where $a_1 = a$, $a_2 = b$, $a_3 = c$, $h_i(\cdot) = Steps_{a_i}(\cdot)$ and $StepCl_{\mathcal{X}_{trivial}}(B_4) = \emptyset$ (see Example 6.1.11 on page 137). Here,

$$B_1 = \{s_1,s_2,s_3,s_4\},\ B_2 = \{t_1,t_2,t_3,t_4\},\ B_3 = \{w\},\ B_4 = S \setminus (B_1 \cup B_2 \cup B_3).$$

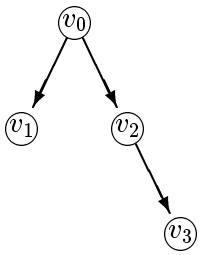
In the first refinement step, for each block $B \in New_{init}$, we compute $NewCl_{\mathcal{X}_{init}}(B)$ and $OldCl_{\mathcal{X}_{init}}(B)$ with the method explained in Figure 6.6 (page 142). Here, we assume the enumeration $B_1 = C_1$, $B_2 = C_2$, $B_3 = C_3$ of New_{init}. Let us consider the block $B = B_1$. In step (1.2) of Figure 6.6 (page 142), for the element $\langle 0, old, a, B_1, h_1\rangle$ of Q, we construct an ordered balanced tree for the values

$$\mu^1_{u_1}[B_1] = \mu^1_{u_2}[B_1] = \mu^1_{t_3}[B_1] = \mu_4[B_1] = \mu_1[B_1] = \mu_2[B_1] = 0$$

which yields a tree consisting of its root. Thus, in step (1.4) of Figure 6.6 (page 142), we insert the element $\langle 1, old, a, B_1, h_1\rangle$ into Q which – again in step (1.2) – leads to the construction of an ordered balanced tree for the values

$$\mu^1_{u_1}[B_2] = \mu^1_{u_2}[B_2] = 0, \quad \mu^1_{t_3}[B_2] = 1, \quad \mu_4[B_2] = \tfrac{2}{3}, \quad \mu_1[B_2] = \mu_2[B_2] = \tfrac{1}{2}.$$

This tree might be of the following form.



$$\begin{array}{lll} v_0.key = 1/2 & v_0.states = \{s_1, s_2\} & v_0.steps(s_i) = \{\mu_i\} \\ v_1.key = 0 & v_1.states = \{s_1, s_2\} & v_1.steps(s_i) = \{\mu^1_{u_i}\} \\ v_2.key = 2/3 & v_2.states = \{s_4\} & v_2.steps(s_4) = \{\mu_4\} \\ v_3.key = 1 & v_3.states = \{s_3\} & v_3.steps(s_3) = \{\mu^1_{t_3}\} \end{array}$$

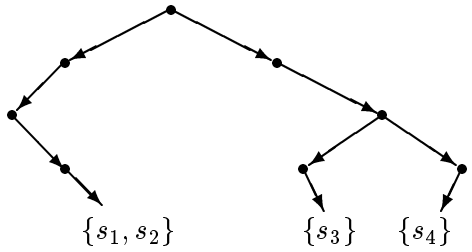
We get $V^1 = \{v_2, v_3\}$ and $V^{\geq 2} = \{v_0, v_1\}$. Thus, in step (1.4) of Figure 6.6 (page 142), we add the elements

$$\langle 2, new, a, \{s_1, s_2\}, h_{1,1}\rangle \text{ and } \langle 2, new, a, \{s_1, s_2\}, h_{1,2}\rangle$$

to Q where $h_{1,1}(s_i) = \{\mu_i\}$ and $h_{1,2}(s_i) = \{\mu^1_{u_i}\}$, $i = 1, 2$. In step (1.5) of Figure 6.6 (page 142), the tuples $(a, \{s_4\}, v_2.steps)$ and $(a, \{s_3\}, v_3.steps)$ are inserted into $NewCl_{\mathcal{X}_{init}}(B_1)$. Then, again in step (1.2) of Figure 6.6 (page 142), for the elements $\langle 2, new, a, \{s_1, s_2\}, h_{1,1}\rangle$ and $\langle 2, new, a, \{s_1, s_2\}, h_{1,2}\rangle$ of Q we construct ordered balanced trees for the values $\mu_1[B_3] = \mu_2[B_3] = 0$ and $\mu^1_{u_1}[B_3] = \mu^1_{u_2}[B_3] = 0$ respectively. In both cases, the resulting tree consists of a single node; thus, in step (1.6) of Figure 6.6 (page 142), the tuples $(a, \{s_1, s_2\}, h_{1,1})$ and $(a, \{s_1, s_2\}, h_{1,2})$ are inserted into $NewCl_{\mathcal{X}_{init}}(B_1)$. Hence, we obtain $OldCl_{\mathcal{X}_{init}}(B_1) = \emptyset$ and

$$NewCl_{\mathcal{X}_{init}}(B_1) \;=\; \{(a, \{s_1, s_2\}, h_{1,1}), (a, \{s_1, s_2\}, h_{1,2}), (a, \{s_3\}, h'_3), (a, \{s_4\}, h'_4)\}$$

where $h'_3(s_3) = \{\mu^1_{t_3}\}$, $h'_4(s_4) = \{\mu_4\}$. In step (2.2.2) of the main algorithm (Figure 6.4, page 136), we apply the method described on page 139 and construct the tree $Tree_{B_1}$ which is of the following form.



Thus, we obtain $B_1/\equiv_{\mathcal{X}_{init}} \;=\; \{\{s_1, s_2\}, \{s_3\}, \{s_4\}\}$, $New_{B_1} \;=\; \{\{s_3\}, \{s_4\}\}$ and

$$\begin{array}{lcl} StepCl_{\mathcal{X}_{init}}(\{s_1, s_2\}) & = & \{(a, h_{1,1}), (a, h_{1,2})\} \\ StepCl_{\mathcal{X}_{init}}(\{s_i\}) & = & \{(a, h'_i)\}\,, \quad i = 3, 4. \end{array}$$

where $h_{1,1}$, $h_{1,2}$, h'_3 and h'_4 are as above.

For the blocks $B \in \{B_2, B_3, B_4\}$, we obtain $NewCl_{\mathcal{X}_{init}}(B_i) = OldCl_{\mathcal{X}_{init}}(B_4) = \emptyset$ and

$$OldCl_{\mathcal{X}_{init}}(B_2) = \{(b, B_2, Steps_b|_{B_2})\}, \quad OldCl_{\mathcal{X}_{init}}(B_3) = \{(c, B_3, Steps_c|_{B_3})\}.$$

In step (2.2.1) of the main algorithm (Figure 6.4, page 136), we apply the method of Figure 6.6 (page 142) and obtain $NewCl_{\mathcal{X}_{init}}(B_i) = OldCl_{\mathcal{X}_{init}}(B_4) = \emptyset$ and

$$OldCl_{\mathcal{X}_{init}}(B_2) = \{(b, B_2, Steps_b|_{B_2})\},\; OldCl_{\mathcal{X}_{init}}(B_3) = \{(c, B_3, Steps_c|_{B_3})\}$$

For all three blocks B_2, B_3, B_4, in step (2.2.2) of the main algorithm (Figure 6.4, page 136), the construction of the tree $Tree_{B_i}$ is skipped because $NewCl_{\mathcal{X}_{init}}(B_i) = \emptyset$. Hence, we get $B_i/\equiv_{\mathcal{X}_{init}} = \{B_i\}$, $New_{B_i} = \emptyset$ and $StepCl_{\mathcal{X}_{init}}(B_i) = StepCl_{\mathcal{X}_{trivial}}(B_i)$, $i = 1, 2$. In summary, the first refinement step yields the partition

$$\mathcal{X} \;=\; Refine(\mathcal{X}_{init}) \;=\; \{\{s_1, s_2\}, \{s_3\}, \{s_4\}, B_2, B_3, B_4\}$$

and $New = \{B_1, B_2, B_3\}$. In the second refinement step, all trees constructed in step (2.2.1) of the main algorithm of Figure 6.4 (i.e. in step (1.2) of method of Figure 6.6 (page 142)) consist of a single node. Thus, $B/\equiv_{\mathcal{X}} = \{B\}$, $New_B = \emptyset$ for all $B \in \mathcal{X}$. In step (3), our algorithm returns $\mathcal{X}$ as the set of bisimulation equivalence classes. ■

Complexity: Let $n = |S|$ be the number of states, m the number of transitions, i.e. $m = \sum_{s \in S} |Steps(s)|$. It is clear that our method can be implemented in space $\mathcal{O}(nm)$. Clearly, the initialization (step (0) of Figure 6.4, page 136) takes $\mathcal{O}(n \cdot |Act|) = \mathcal{O}(n)$ time as, for each state $s \in S$, we traverse a tree of depth $|Act|$.[10] In what follows, let N be the total number of refinement steps (the number of executions of the loop in step (2) in Figure 6.4) and let $\mathcal{X}_i$ be the partition which we obtain in the $(i+1)$-st refinement step, i.e. $\mathcal{X}_0 = \mathcal{X}_{init}$ $\mathcal{X}_{i+1} = Refine(\mathcal{X}_i)$, $i = 0, 1, \ldots, N-1$, $\mathcal{X}_N \;=\; S/\sim$. Let $Cost^i_{(2.2.1)}$ be the cost for the executions of step (2.2.1) of the main algorithm (Figure 6.4, page 136) where we range over all blocks $B \in \mathcal{X}_i$, $i = 0, 1, \ldots, N-1$ and let

$$Cost_{(2.2.1)} \;=\; \sum_{i=0}^{N-1} Cost^i_{(2.2.1)}$$

be the total cost caused by step (2.2.1). Similarly, we define $Cost_{(2.2.2)}$ to be the total cost that arises from the executions of step (2.2.2) of Figure 6.4 where we range over all refinement steps. We show in Section 6.3 (Lemma 6.3.4 (page 155) and Lemma 6.3.6 (page 157)):

$$Cost_{(2.2.1)} \;=\; \mathcal{O}(mn(\log m + \log n)), \quad Cost_{(2.2.2)} \;=\; \mathcal{O}(mn).$$

Thus, we obtain the time complexity $\mathcal{O}(mn(\log m + \log n))$ for computing the bisimulation equivalence classes.

[10] Recall that we assume Act to be fixed.

Computing $NewCl_{\mathcal{X}}(B)$ *and* $OldCl_{\mathcal{X}}(B)$

Input:

- a partition $\mathcal{X}$ and $B \in \mathcal{X}$
- an enumeration $C_1, \ldots, C_l$ of *New*
- $StepCl_{\mathcal{X}_{old}}(B)$ (the step classes of B with respect to the previous partition)

Method:

(0) We set $NewCl_{\mathcal{X}}(B) := \emptyset$, $OldCl_{\mathcal{X}}(B) := \emptyset$ and

$$Q \;:=\; \Big\{\langle 0, old, a, B, h_{old}\rangle : (a, h_{old}) \in StepCl_{\mathcal{X}_{old}}(B)\Big\};$$

(1) While $Q \neq \emptyset$ do

(1.1) Choose some $\langle j, b, a, L, h\rangle \in Q$ and set $Q := Q \setminus \{\langle j, b, a, L, h\rangle\}$;

(1.2) Construct an ordered balanced tree T for $p_\mu = \mu[C_{j+1}]$, $\mu \in h(s)$, $s \in L$ where each node v is labelled by a record $(v.key, v.states, v.steps)$ such that

- $v.states = \{s \in B : v.key = \mu[C_{j+1}] \text{ for some } \mu \in h(s)\}$,
- $v.steps$ is the function that assigns to each state $s \in v.states$ the set

$$v.steps(s) \;=\; \{\mu \in h(s) : v.key = \mu[C_{j+1}]\};$$

We define:

$$\begin{aligned} V^{\geq 2} &:= \{v : v \text{ node in } T \text{ with } |v.states| \geq 2\};\\ V^{1} &:= \{v : v \text{ node in } T \text{ with } |v.states| = 1\}; \end{aligned}$$

(1.3) If T consists of two or more nodes then $b' := new$ else $b' := b$;

(1.4) If $j < l$ then $Q := Q \cup \Big\{\langle j+1, b', a, v.states, v.steps\rangle : v \in V^{\geq 2}\Big\}$;

(1.5) If $j < l \wedge b' = new$ then

$$NewCl_{\mathcal{X}}(B) := NewCl_{\mathcal{X}}(B) \cup \Big\{(a, v.states, v.steps) : v \in V^{1}\Big\};$$

(1.6) If $j = l \wedge b' = new$ then

$$NewCl_{\mathcal{X}}(B) := NewCl_{\mathcal{X}}(B) \cup \{(a, v.states, v.steps) : v \text{ node in } T\};$$

(1.7) If $j = l \wedge b' = old$ then $OldCl_{\mathcal{X}}(B) := \{(a, B, v.steps)\}$ where v is the root of T;

(2) Return $NewCl_{\mathcal{X}}(B)$ and $OldCl_{\mathcal{X}}(B)$.

Figure 6.6:

6.2 Computing the simulation preorder

We present an algorithm that computes the simulation preorder of a finite action-labelled concurrent probabilistic system $(S, Act, Steps)$. The schema of our algorithm is as in the non-probabilistic case [HHK95]: We start with the trivial preorder $R = S \times S$ and then successively remove those pairs (s, s') from R where s has a transition that cannot be "simulated" by a transition of s'. This schema is sketched in Figure 6.7 (page 144). In the non-probabilistic case, $s \sqsubseteq_R s'$ iff for each transition $s \xrightarrow{a} t$ there is a transition $s' \xrightarrow{a} t'$ with $(t, t') \in R$. For $(S, Act, Steps)$ to be an action-labelled concurrent probabilistic system and s, $s' \in S$, the relation $\sqsubseteq_R$ is given by: $s \sqsubseteq_R s'$ iff for each transition $s \xrightarrow{a} \mu$ there is a transition $s' \xrightarrow{a} \mu'$ with $\mu \preceq_R \mu'$.[11] In the fully probabilistic case, $s \sqsubseteq_R s'$ is defined as in Definition 3.4.16 (page 60), i.e. $s \sqsubseteq_R s'$ iff either s is terminal or $\mathbf{P}(s, \cdot) \preceq_{R'} \mathbf{P}(s', \cdot)$ where $R' = \{\langle a, t\rangle, \langle a, t'\rangle) : (t, t') \in R, a \in Act\}$.

For non-probabilistic systems, the schema of Figure 6.7 can be implemented in time $\mathcal{O}(mn)$ [HHK95]. It seems to be hard to modify the method of [HHK95] for the probabilistic case because it successively removes those pairs (s, s') of R where s is an a-predecessor of some state t (in the sense that there is a transition $s \xrightarrow{a} t$) and s' does not have an a-successor in $\{t' : (t, t') \in R\}$. The problem in the probabilistic case is that the induced predecessor/successor relations on states[12] does not give enough information. Even the probabilities for the a-successors/predecessors of the states do not contain the necessary information for computing the simulation preorder since there might be non-similar states that cannot be distinguished with these predecessor/successor relations (cf. Remark 3.4.11, page 57).

In the probabilistic case, we implement the schema of Figure 6.7 in such a way that the test whether $s \sqsubseteq_R s'$ is done with the help of a method for deciding whether $\mu \preceq_R \mu'$ for some distributions μ, μ' on a finite set X and a binary relation R on X. We show that the question whether $\mu \preceq_R \mu'$ can be reduced to a maximum flow problem in a suitable chosen network.

We proceed in the following way. In Section 6.2.1, we explain how to test whether $\mu \preceq_R \mu'$ via a maximum flow problem. Then, in Section 6.2.2 we describe our algorithm for concurrent probabilistic systems while Section 6.2.3 deals with fully probabilistic systems.

6.2.1 The test whether $\mu \preceq_R \mu'$

We show that the question whether two distributions are related via a weight function (i.e. whether $\mu \preceq_R \mu'$) can be reduced to a maximum flow problem in a suitable chosen network.

Networks and their maximum flow: We briefly recall the basic definitions of networks. For further details about networks and maximum flow problems see e.g. [Even79]. A *network* is a tuple $\mathcal{N} = (N, E, \bot, \top, c)$ where (N, E) is a finite directed graph (i.e. N is a set of nodes, $E \subseteq N \times N$ a set of edges) with two specified nodes $\bot$ (the *source*) and $\top$ (the *sink*) and a *capacity cap*, i.e. a function $cap : E \to \mathbb{R}_{\geq 0}$ which assigns to each edge

[11] Here, $\preceq_R$ is the weight-function-based relation defined as in Section 2.2, page 30.

[12] E.g. in concurrent probabilistic systems, s is an a-predecessor of t iff $\mu(t) > 0$ for some $\mu \in Steps_a(s)$

Computing the simulation preorder

Input: a finite (probabilistic or non-probabilistic) system with state space S

Output: the simulation preorder $\sqsubseteq_{\mathrm{sim}}$ on S

Method:

$R := S \times S$;

While there exists $(s, s') \in R$ with $s \not\sqsubseteq_R s'$ do $R := R \setminus \{(s, s')\}$;

Return R.

Figure 6.7: General schema for computing the simulation preorder

$(v, w) \in E$ a non-negative real number $cap(v, w)$. A *flow function* f for $\mathcal{N}$ is a function which assigns to each edge e a real number $f(e)$ such that:

- $0 \leq f(e) \leq cap(e)$ for all edges e.
- Let $in(v)$ be the set of incoming edges to node v and $out(v)$ the set of outgoing edges from node v. Then, for each node $v \in N \setminus \{\bot, \top\}$:

$$\sum_{e \in in(v)} f(e) \;=\; \sum_{e \in out(v)} f(e)$$

The *flow* $Flow(f)$ of f is given by

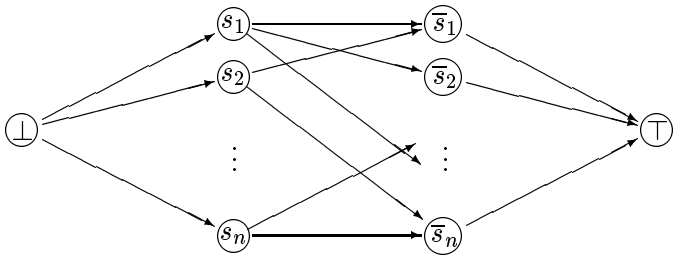
$$Flow(f) \;=\; \sum_{e \in out(\bot)} f(e) \;-\; \sum_{e \in in(\bot)} f(e).$$

The *maximum flow* in $\mathcal{N}$ is the supremum (maximum) over the values $Flow(f)$ where f ranges over all flow functions in $\mathcal{N}$. Algorithms to compute the maximum flow are given e.g. in [FoFu62, Dini70, MPM78, CHM90].

The test whether $\mu \preceq_R \mu'$: Let S be a finite set, R a subset of $S \times S$ and let μ, $\mu' \in Distr(S)$. Let $\overline{S} = \{\overline{t} : t \in S\}$ where $\overline{t}$ are pairwise distinct "new" states (i.e. $\overline{t} \notin S$). We choose new elements $\bot$ and $\top$ not contained in $S \cup \overline{S}$, $\bot \neq \top$. We associate with (μ, μ') the following network $\mathcal{N}(\mu, \mu', R)$. The nodes are the elements of $S \cup \overline{S}$ and $\bot$ (the source) and $\top$ (the sink), i.e. $N = \{\bot, \top\} \cup S \cup \overline{S}$. The edges are

$$E \;=\; \{(s, \overline{t}) : (s, t) \in R\} \;\cup\; \{\; (\bot, s) \;:\; s \in S \;\} \;\cup\; \{\; (\overline{t}, \top) \;:\; t \in S \;\}.$$

I.e. the underlying graph (N, E) is of the form



where $S = \{s_1, \ldots, s_n\}$ and where the arrows between the nodes s_i and the nodes $\overline{s}_j$ describe the relation R in the sense that there is an arrow from s_i to $\overline{s}_j$ iff $(s_i, s_j) \in R$. The capacities $cap(e) \in [0, 1]$ are given by:

$$cap(\bot, s) = \mu(s),\ cap(\overline{t}, \top) = \mu'(t),\ cap(s, \overline{t}) = 1.$$

As $in(\bot) = \emptyset$ we get

$$Flow(f) \ = \ \sum_{s \in Supp(\mu)} f(\bot, s)$$

for each flow function f in $\mathcal{N}(\mu, \mu', R)$.

Lemma 6.2.1 *$\mu \preceq_R \mu'$ iff the maximum flow in $\mathcal{N}(\mu, \mu', R)$ is 1.*

Proof: First we assume that $\mu \preceq_R \mu'$. For each flow function f in $\mathcal{N}(\mu, \mu', R)$:

$$Flow(f) \ = \ \sum_{s \in S} f(\bot, s) \ \leq \ \sum_{s \in S} cap(\bot, s) \ = \ \sum_{s \in S} \mu(s) \ = \ 1.$$

Let *weight* be a weight function for (μ, μ') with respect to R. We define a flow function f as follows: $f(\bot, s) = \mu(s)$, $f(\overline{t}, \top) = \mu'(t)$ and $f(s, \overline{t}) = weight(s, t)$. Then,

$$Flow(f) \ = \ \sum_{s \in Supp(\mu)} f(\bot, s) \ = \ \sum_{s \in Supp(\mu)} \mu(s) \ = \ 1.$$

Hence, the maximum flow of $\mathcal{N}(\mu, \mu', R)$ is 1.

Next, we assume that the maximum flow is 1. Let f be a flow function with $Flow(f) = 1$. Since $f(\bot, s) \leq cap(\bot, s) = \mu(s)$ and since

$$\sum_{s \in S} f(\bot, s) \ = \ Flow(f) \ = \ 1 \ = \ \sum_{s \in S} \mu(s)$$

we get $f(\bot, s) = \mu(s)$ for all $s \in S$. Similarly, we get $f(\overline{t}, \top) = \mu'(t)$ for all $t \in S$. Let $weight(s, t) = f(s, \overline{t})$ for all $(s, t) \in R$ and $weight(s, t) = 0$ if $(s, t) \notin R$. Then,

$$\sum_{t \in S} weight(s, t) \ = \ \sum_{t \in S} f(s, \overline{t}) \ = \ f(\bot, s) \ = \ \mu(s),$$

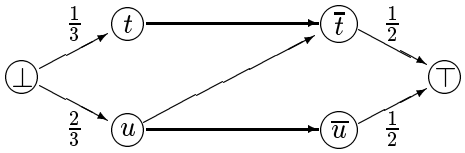
and similarly, $\sum_{s \in S} weight(s, t) \ = \ \mu'(t)$. Hence, *weight* is a weight function for (μ, μ') with respect to R. Thus, $\mu \preceq_R \mu'$. ■

Lemma 6.2.1 (page 145) yields a method for deciding whether $\mu \preceq_R \mu'$. We construct the network $\mathcal{N}(\mu, \mu', R)$ and compute the maximum flow with well-known methods (see Figure 6.8, page 146).

Example 6.2.2 Let $S = \{t, u\}$, $R = \{(t, t), (u, u), (u, t)\}$ and $\mu,\ \mu' \in Distr(S)$ with

$$\mu(t) = \tfrac{1}{3}, \quad \mu(u) = \tfrac{2}{3}, \quad \mu'(t) = \mu'(u) = \tfrac{1}{2}.$$

The associated network $\mathcal{N}(\mu, \mu', R)$ is of the following form.



Test whether $\mu \preceq_R \mu'$

Input: a nonempty finite set S, distributions μ, $\mu' \in \mathit{Distr}(S)$ and $R \subseteq S \times S$

Output: "Yes" if $\mu \preceq_R \mu'$, "No" otherwise.

Method:

compute the maximum flow F of the network $\mathcal{N}(\mu, \mu', R)$;

if $F < 1$ then return "No" else return "Yes".

Figure 6.8: Test whether $\mu \preceq_R \mu'$

The flow function f with

$$f(\bot, t) = f(t, \overline{t}) = \tfrac{1}{3},\ \ f(\bot, u) = \tfrac{2}{3},\ \ f(u, \overline{t}) = \tfrac{1}{6},\ \ f(u, \overline{u}) = f(\overline{u}, \top) = f(\overline{t}, \top) = \tfrac{1}{2}$$

yields the maximum flow $\mathit{Flow}(f) = 1$. Thus, $\mu \preceq_R \mu'$. ■

To our knowledge, the best known algorithm for computing the maximum flow in a network is those of [CHM90] which has time and space complexity $\mathcal{O}(n^3/\log n)$ and $\mathcal{O}(n^2)$ respectively where n is the number of nodes in the network. Hence:

Lemma 6.2.3 *The test whether* $\mu \preceq_R \mu'$ *can be done in time* $\mathcal{O}(n^3/\log n)$ *and space* $\mathcal{O}(n^2)$ *where* $n = |S|$.

Remark 6.2.4 Another possibility for testing whether $\mu \preceq_R \mu'$ is to consider the following linear inequality system with the variables $x_{s,t}$, $(s,t) \in R$:

$$\sum_{\substack{t \in S \\ (s,t) \in R}} x_{s,t} \;=\; \mu(s) \ \text{ for all } s \in S$$

$$\sum_{\substack{s \in S \\ (s,t) \in R}} x_{s,t} \;=\; \mu'(s) \ \text{ for all } t \in S$$

and $x_{s,t} \geq 0$ for all $(s,t) \in R$. Then, $\mu \preceq_R \mu'$ iff the system above has a solution. In that case, the solution $(x_{s,t})_{(s,t) \in R}$ yields a weight function for (μ, μ') with respect to R. The above system has $|R| = \mathcal{O}(n^2)$ variables and $|R| + 2|S| = \mathcal{O}(n^2)$ equations. To our knowledge, there is no method for solving inequality systems of this type that beat the time complexity $\mathcal{O}(n^3/\log n)$. ■

6.2.2 The concurrent case

In the sequel, $(S, \mathit{Act}, \mathit{Steps})$ is a finite action-labelled concurrent probabilistic system. If R is a binary relation on S and s, $s' \in S$ then

$$s \sqsubseteq_R s' \text{ iff whenever } s \xrightarrow{a} \mu \text{ then there is some } s' \xrightarrow{a} \mu' \text{ with } \mu \preceq_R \mu'.$$

The algorithm for computing the simulation preorder is sketched in Figure 6.9 (page 148). We first compute the set $R = \{(s, s') \in S \times S : \mathit{act}(s) \subseteq \mathit{act}(s'), s \neq s'\}$. If

$R = \{(s, s') : s \neq s'\}$ then all states are similar and we are done. In what follows, we suppose that R does not contain all pairs (s, s'), $s \neq s'$. We organize R as a queue Q where the ordering in the initial queue is arbitrary. We use the usual operators $\mathit{Front}(Q)$ which yields the first element of Q, $\mathit{Remove}(Q)$ which removes the first element of Q (both under the assumption that Q is not empty) and $\mathit{Add}(Q, x)$ which adds x at the end of Q. For $(s, s') \in R$ and $(a, \mu) \in \mathit{Steps}(s)$, we use a set $\mathit{Sim}_{(s,a,\mu)}(s')$ of distributions $\mu' \in \mathit{Steps}_a(s')$ that are still candidates to match the transition $s \xrightarrow{a} \mu$, i.e. $\mu \not\preceq_R \mu'$ is not yet detected. Initially, we deal with $\mathit{Sim}_{(s,a,\mu)}(s') = \mathit{Steps}_a(s')$. A distribution $\mu' \in \mathit{Steps}_a(s')$ is removed from $\mathit{Sim}_{(s,a,\mu)}(s')$ just in the moment where $\mu \not\preceq_R \mu'$ is detected. Then, $\mu \not\preceq_R \mu'$ in all following iterations. We represent the set $\mathit{Sim}_{(s,a,\mu)}(s')$ as a list consisting of (pointers to) elements of $\mathit{Steps}_a(s')$. For these lists $\mathit{Sim}_{(s,a,\mu)}(s')$, we use the operations $\mathit{First}(\cdot)$ which yields the first element of $(\cdot)$ and $\mathit{Next}(\cdot)$ which removes the first element of $(\cdot)$, i.e. the list pointer is shifted to the second element.

In what follows, we refer to R as the set of pairs (s, s') that are contained in Q. By an iteration, we mean the execution of steps (1) and (2) (including the substeps (2.1)-(2.6)). We say a pair (s, s') is *investigated* in some iteration if it is those element of Q that is chosen in the else-branch of step (1).

Initially, we define *last* to be the last element of Q. In all iterations, *last* is either undefined ($\mathit{last} = \bot$) or the left most element (s, s') of Q such that – after the last investigation of (s, s') – no element (t, t') is removed from Q. Hence, if we investigate (s, s') where $(s, s') = \mathit{last}$ and obtain $s \sqsubseteq_R s'$ then we have $t \sqsubseteq_R t'$ for all pairs $(t, t') \in R$. Thus, R is the simulation preorder (cf. step (2.3)). If $s \not\sqsubseteq_R s'$ for the element (s, s') which is investigated then we set $\mathit{last} = \bot$ (step (2.5)) and "wait" for the next pair (t, t') in Q which we do not remove from Q (step (2.4.2)).

Computing the simulation preorder

Input: a finite action-labelled concurrent probabilistic system $(S, Act, Steps)$

Output: the simulation preorder $\sqsubseteq_{\mathrm{sim}}$

Method:

(0) [Initialization]
  (0.1) $Q := \emptyset$;
  (0.2) For all $(s, s') \in S \times S$ with $s \neq s'$ and $act(s) \subseteq act(s')$ do
    (0.2.1) $Add(Q, (s, s'))$;
    (0.2.2) $last := (s, s')$;
    (0.2.3) For all $(a, \mu) \in Steps(s)$ do $Sim_{(s,a,\mu)}(s') := Steps_a(s')$;
(1) If $Q = \emptyset$ then go to (3) else begin $(s, s') := Front(Q)$; $Remove(Q)$; end;
(2) For all $(a, \mu) \in Steps(s)$ do
  (2.1) $sim := false$;
  (2.2) While $\neg sim$ and $Sim_{(s,a,\mu)}(s') \neq \emptyset$ do
    (2.2.1) $\mu' := First(\ Sim_{(s,a,\mu)}(s')\ )$;
    (2.2.2) If $\mu \preceq_R \mu'$ then $sim := true$ else $Next(\ Sim_{(s,a,\mu)}(s')\ )$;
  (2.3) If sim and $last = (s, s')$ then go to (3);
  (2.4) If sim and $last \neq (s, s')$ then
    (2.4.1) $Add(Q, (s, s'))$;
    (2.4.2) If $last = \perp$ then $last := (s, s')$;
  (2.5) If $\neg sim$ then $last := \perp$;
  (2.6) go to (1);
(3) Return $R \cup \{(s, s) : s \in S\}$ where R is the set of pairs that are contained in Q.

Figure 6.9: Algorithm for computing the simulation preorder

Example 6.2.5 We apply our algorithm for computing the simulation preorder (Figure 6.9, page 148) to the system shown in Figure 6.10 (page 149). In the initialization step (step (0)) we obtain the queue Q containing the pairs

- (s_i, s_j), (s_i, v), (v, s_i), (v, v'), $i, j = 1, 2$, $i \neq j$, $v, v' \in \{v_1, v_2, w\}$, $v \neq v'$
- (u, u'), $u, u' \in U$, $u \neq u'$
- (t_1, t_2), (t_2, t_1)
- (u, t_1), (u, s_i), (u, v), $u \in U$, $v \in \{v_1, v_2, w\}$, $i = 1, 2$.

Here, $U = \{u_1^k, u_2^h : k = 0, \ldots, 4, h = 0, 1, 2\}$ denotes the set of terminal states. The pairs

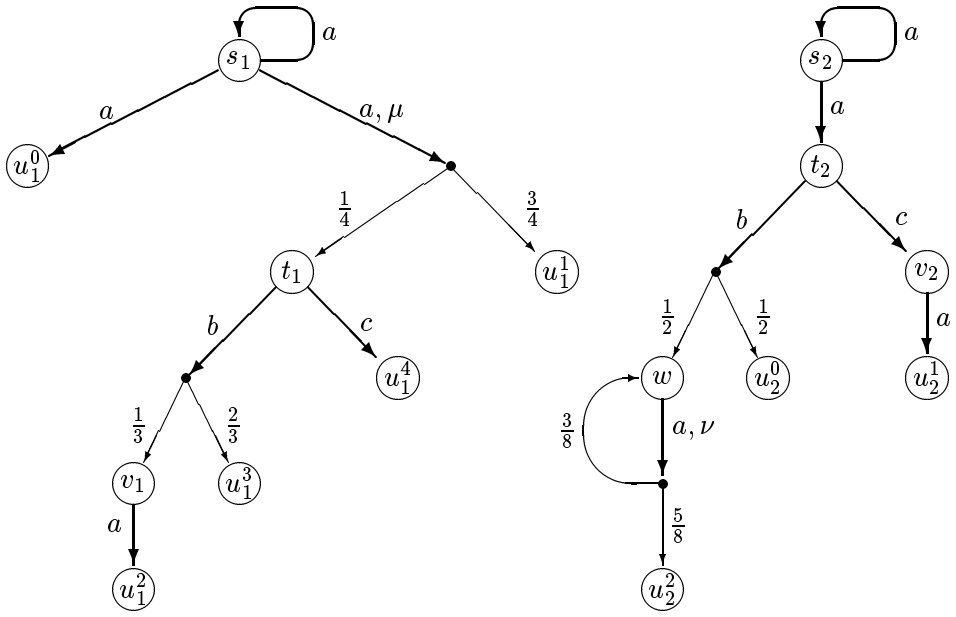
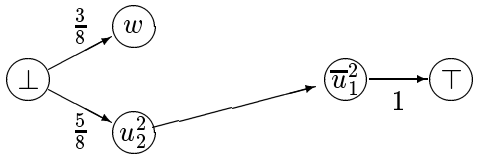


Figure 6.10:

(s_2, s_1), (s_i, w), (w, v_j), (s_i, v_j), $i, j = 1, 2$, are removed during their first investigation. For instance:

- For the pair (w, v_1) the algorithm computes the maximum flow of the network



  which is 5/8. Thus, there is no transition of v_1 that can "simulate" the transition $w \xrightarrow{a} \nu$.
- For the pair (s_2, s_1) the algorithm tries to find a transition of s_1 which can "simulate" the transition $s_2 \xrightarrow{a} \mu_{t_2}^1$. As $\rho \not\preceq_R \mu_{t_2}^1$ for all $\rho \in \mathit{Steps}_a(s_1) = \{\mu_{u_1^0}^1, \mu_{s_1}^1, \mu\}$ the pair (s_2, s_1) is removed from R.

The first investigation of (s_1, s_2) yields

$$\mathit{Sim}_{(s_1, a, \mu_{u_1^0}^1)}(s_2) = \{\mu_{t_2}^1, \mu_{s_2}^1\}$$

$$\mathit{Sim}_{(s_1, a, \mu_{s_1}^1)}(s_2) = \{\mu_{s_2}^1\}, \quad \mathit{Sim}_{(s_1, a, \mu)}(s_2) = \{\mu_{t_2}^1\}$$

as $\mu \not\preceq_R \mu^1_{s_2}$ and $\mu^1_{s_1} \not\preceq_R \mu^1_{t_2}$.

We suppose that initially the pair (t_2, t_1) is the last element of Q (i.e. $last = (t_2, t_1)$ after the initialization step). Then, the first investigation of (t_2, t_1) yields $t_2 \not\sqsubseteq_R t_1$ (as (w, v_1) is already removed from Q). After the removement of (t_2, t_1) we have $x \sqsubseteq_R y$ for all $(x, y) \in R$. Hence, if (x, y) is the element of Q which is investigated immediately after the removement of (t_2, t_1) then the algorithm sets $last = (x, y)$ after the (second) investigation of (x, y). After investigating all remaining elements of Q once more, we reach again the pair $(x, y) = last$. Thus, the condition of step (2.3) is fulfilled and the algorithm returns the simulation preorder which consists of the following pairs.

- (s_1, s_2), (t_1, t_2)
- (v_i, s_j), (w, s_j), (v_i, w), $i, j = 1, 2$
- (u, x), $u \in U$, $x \in \{s_1, s_2, t_1, t_2, v_1, v_2, w\}$

and all pairs (x, x), $x \in S$. ■

Complexity: Let $n = |S|$ be the number of states and m the number of transitions, i.e. $m = \sum_{s \in S} |\mathit{Steps}(s)|$. For $s \in S$ and $a \in \mathit{Act}$, let $m_s = |\mathit{Steps}(s)|$, $m_{a,s} = |\mathit{Steps}_a(s)|$. (Then, $m = \sum_{s \in S} m_s$.)

We suppose a representation of $(S, \mathit{Act}, \mathit{Steps})$ which contains for each state $s \in S$ and each action $a \in \mathit{Act}$ a pointer to a list whose elements represent the distributions $\mu \in \mathit{Steps}_a(s)$. Then, the initialization step (i.e. the computation of the initial set R, the sets $\mathit{Sim}_{(s,a,\mu,s')}$ and $last$) takes $\mathcal{O}(n^2 \cdot |\mathit{Act}|) = \mathcal{O}(n^2)$ time. We suppose that step (2.2.2) is executed N-times. Then, the time complexity of our algorithm is $\mathcal{O}(N \cdot n^3 / \log n)$ if the test whether $\mu \preceq_R \mu'$ is done by computing the maximum flow in $\mathcal{N}(\mu, \mu', R)$ with the algorithm of [CHM90]. We show that $N \leq mn^3 + m^2$.[13]

If in step (2.3) the condition $\mathit{sim} \wedge last = (s, s')$ is fulfilled, i.e. $t \sqsubseteq_R t'$ for all $(t, t') \in R$, then we reach step (3) where the algorithm halts. Hence, at most after n^2 iterations either some pair (s, s') is removed from R or the algorithms halts. Thus, each pair (s, s') is investigated at most n^2-times. For each pair (s, s') and $(a, \mu) \in \mathit{Steps}(s)$, there are at most $m_{a,s'}$ unsuccessful attempts to find $\mu' \in \mathit{Steps}_a(s')$ with $\mu \preceq_R \mu'$ (since as soon as $\mu \not\preceq_R \mu'$ is detected μ' is removed from $\mathit{Sim}_{(s,a,\mu)}$). Ranging over all iterations where (s, s') is investigated and for fixed $(a, \mu) \in \mathit{Steps}(s)$, step (2.2.2) is executed at most $N_{(s,a,\mu,s')}$-times where $N_{(s,a,\mu,s')} = n^2 + m_{a,s'}$. We obtain

$$N \leq \sum_{s \in S} \sum_{a \in \mathit{Act}} \sum_{\mu \in \mathit{Steps}_a(s)} \sum_{s' \in S} N_{(s,a,\mu,s')} \leq \sum_{s \in S} \sum_{(a,\mu) \in \mathit{Steps}(s)} (n^3 + m) = mn^3 + m^2.$$

The space complexity is $\mathcal{O}(mn + n^2)$ as the representation of the transition relation takes $\mathcal{O}(mn)$ space, the representation of R (i.e. the queue Q) $\mathcal{O}(n^2)$ space. For each of the lists $\mathit{Sim}_{(s,a,\mu)}(s')$ we need $\mathcal{O}(m_{a,s'})$ space. Summing up over all s', a, s, we need

$$\mathcal{O}\left(\sum_{s \in S} \sum_{a \in \mathit{Act}} \sum_{s' \in S} m_{a,s'} \right) = \mathcal{O}(mn)$$

space for the representation of the lists $\mathit{Sim}_{(s,a,\mu)}(s')$. We obtain:

[13] Intuitively, the number of unsuccessful tests whether $\mu \preceq_R \mu'$ is bounded by m^2 while mn^3 is an upper bound for the number of successful tests whether $\mu \preceq_R \mu'$.

Theorem 6.2.6 *In concurrent probabilistic systems, the simulation preorder can be computed in time* $\mathcal{O}((mn^6 + m^2n^3)/\log n)$ *and space* $\mathcal{O}(mn + n^2)$ *where* n *is the number of states and* m *the number of transitions.*

The following remark (Remark 6.2.7) shows that each algorithm for computing the simulation preorder which is based on the schema sketched in Figure 6.7 (page 144) and which tests whether $s \sqsubseteq_R s'$ via the condition

$$(*) \qquad \forall\ s \xrightarrow{a} \mu\ \exists\ s' \xrightarrow{a} \mu' :\ \mu \preceq_R \mu'$$

has to test whether $\mu \preceq_R \mu'$ for $\Omega(m^2)$ pairs (μ, μ').[14] Thus, for the worst case complexity the number of unsuccessful tests whether $\mu \preceq_R \mu'$ in step (2.2.2) cannot be reduced. However, it might be possible to improve the algorithm, e.g. by replacing (*) by a simpler condition or by reducing the number of successful tests in step (2.2.2).

Remark 6.2.7 Let $(S, Act, Steps)$ be an action-labelled concurrent probabilistic system where $S = \{s_0, \ldots, s_k\} \cup \{s, s'\}$, $s_i \sqsubseteq_{\text{sim}} s_j$ iff $i \leq j$ and $|Steps(s_i)| \leq 1$, $i = 0, \ldots, k$.[15] Moreover, we suppose that there is some action a such that:

- $Steps_b(s) = Steps_b(s') = \emptyset$ for all $b \neq a$
- If $\mu \in Steps_a(s)$, $\mu' \in Steps_a(s')$ then there is no weight function for (μ, μ') with respect to the simulation preorder $\sqsubseteq_{\text{sim}}$.

Then, for all pairs $(\mu, \mu') \in Steps_a(s) \times Steps_a(s')$ we have to test whether $\mu \preceq_R \mu'$. Since $m = m_s + m_{s'} + k$ we get for fixed k: $\Omega(m_s \cdot m_{s'}) = \Omega(m^2)$. Thus, the number of tests whether $\mu \preceq_R \mu'$ is $\Omega(m^2)$. ■

Dealing with a reactiv system, we have $N_{(s,a,\mu,s')} \leq n^2$ and $N \leq |Act| \cdot n^4$ (where N is the number of tests whether $\mu \preceq_R \mu'$) We obtain:

Theorem 6.2.8 *In reactive systems, the simulation preorder can be computed in time* $\mathcal{O}(n^7/\log n)$ *and space* $\mathcal{O}(n^2)$ *where* n *is the number of states.*

6.2.3 The fully probabilistic case

In what follows, we fix a finite action-labelled fully probabilistic system $(S, Act, \mathbf{P})$. Recall that $s \sqsubseteq_R s'$ iff either s is terminal or $\mathbf{P}(s, \cdot) \preceq_{R'} \mathbf{P}(s', \cdot)$ where

$$R' = \{\langle a, t\rangle, \langle a, t'\rangle) : (t, t') \in R, a \in Act\}$$

(see Definition 3.4.16, page 60). For s, $s' \in S$ and $R \subseteq S \times S$ we define $\mathcal{N}(s, s', R)$ to be the network (N, E, cap) where

$$\begin{aligned} N &= \{\bot, \top\} \cup Act \times (S \cup \overline{S}), \quad \overline{S} = \{\overline{t} : t \in S\} \\ E &= \{(\bot, \langle a, t\rangle), (\langle a, \overline{t}\rangle, \top) : t \in S, a \in Act\} \cup \{(\langle a, t\rangle, \langle a, \overline{u}\rangle) : (t, u) \in R\} \\ cap(\bot, \langle a, t\rangle) &= \mathbf{P}(s, a, t), \quad cap(\langle a, \overline{t}\rangle, \top) = \mathbf{P}(s', a, t), \quad cap(\langle a, t\rangle, \langle a, \overline{u}\rangle) = 1. \end{aligned}$$

[14] Here, $\Omega(\cdot)$ denotes asymptotic lower bounds.

[15] E.g. $Steps(s_0) = \emptyset$ and $Steps(s_i) = \{(a, \mu_i)\}$ where $\mu_i(s_0) = 1/i$ and $\mu_i(s_i) = 1 - 1/i$.

Similarly to Lemma 6.2.1 (page 145) it can be shown that

$$s \sqsubseteq_R s' \text{ iff either } s \text{ is terminal or the maximum flow in } \mathcal{N}(s, s', R) \text{ is 1.}$$

Thus, the simulation preorder of a fully probabilistic system can be computed by the following method. We start with the preorder

$$R \;=\; \{(s, s') \in S \times S' : \text{if } s' \text{ is terminal then } s \text{ is terminal}\}.$$

As long as there is a pair $(s, s') \in R$ where $s \not\sqsubseteq_R s'$ we remove (s, s') from R.[16] This method can be implemented similar to the one proposed in Section 6.2 (Figure 6.9, page 148). The time complexity is as in the reactive case. We obtain:

Theorem 6.2.9 *In fully probabilistic systems, the simulation preorder of can be computed in time $\mathcal{O}(n^7/\log n)$ and space $\mathcal{O}(n^2)$ where n is the number of states.*

In many applications, one wants only to give lower and upper bounds on the preobabilities of an acceptable system behaviour rather than the exact probabilities. Jonsson & Larsen [JoLa91] define a notion of "satisfaction relation" that relates the states of a given fully probabilistic system and the states of a *fully probabilistic specification system* which prescribes intervals of allowed probabilities. Note that in contrast to [JoLa91] we deal with action-labelled systems while [JoLa91] deals with systems where the states are labelled by atomic propositions. We modify the definitions of [JoLa91] as follows.

Definition 6.2.10 [Action-labelled fully probabilistic specification systems] *An* action-labelled fully probabilistic specification system *is a tuple $(\overline{S}, Act, \overline{\mathbf{P}})$ where $\overline{S}$ is a finite set of states, Act a finite set of actions and $\overline{\mathbf{P}} : \overline{S} \times Act \times \overline{S} \to 2^{[0,1]}$ is a function such that, for all $\overline{s}$, $\overline{t} \in \overline{S}$ and $a \in Act$, $\overline{\mathbf{P}}(\overline{s}, a, \overline{t})$ is a closed interval contained in $[0, 1]$.*

Definition 6.2.11 [The satisfaction relation *sat*] *Let $(S, Act, \mathbf{P})$ be a finite action-labelled fully probabilistic system and $(\overline{S}, Act, \overline{\mathbf{P}})$ an action-labelled fully probabilistic specification system. If $R \subseteq S \times \overline{S}$ and $s \in S$, $\overline{s} \in \overline{S}$ then we define the relation $sat_R \subseteq S \times \overline{S}$ as follows: s sat_R $\overline{s}$ iff either s is terminal or there exsits a* weight function *for $(s, \overline{s})$ with respect to R, i.e. a function $weight : S \times Act \times \overline{S} \to [0, 1]$ such that for all $a \in Act$ and $t \in S$, $\overline{t} \in \overline{S}$:*

1. *If $weight(t, a, \overline{t}) > 0$ then $(t, \overline{t}) \in R$.*
2. $$\sum_{\overline{u} \in \overline{S}} weight(t, a, \overline{u}) \;=\; \mathbf{P}(s, a, t), \qquad \sum_{u \in S} weight(u, a, \overline{t}) \;\in\; \overline{\mathbf{P}}(\overline{s}, a, \overline{t}).$$

A satisfaction relation *for $(S, Act, \mathbf{P})$ and $(\overline{S}, Act, \overline{\mathbf{P}})$ is a binary relation $R \subseteq S \times \overline{S}$ such that s sat_R $\overline{s}$ for all $(s, \overline{s}) \in R$. We write s sat s' iff $(s, \overline{s})$ is contained in some satisfaction relation for $(S, Act, \mathbf{P})$ and $(\overline{S}, Act, \overline{\mathbf{P}})$.*

The relation $sat \subseteq S \times \overline{S}$ can be computed similar to the way in which we compute the simulation preorder $\sqsubseteq_{\text{sim}}$ of a fully probabilistic system; the only difference being the use of networks with lower and upper bounds, see e.g. [Even79]. We start with the relation $R = S \times \overline{S}$ and successively remove those pairs $(s, \overline{s})$ from R where $\neg(s \; sat_R \; \overline{s})$. For

[16] The test $s \not\sqsubseteq_R s'$ can be done by computing the maximum flow in $\mathcal{N}(s, s', R)$.

the test whether $s\ sat_R\ \overline{s}$ we compute the maximum flow in the network $\mathcal{N}(s,\overline{s},R) = (N,E,cap_l,cap_u)$ where cap_l, cap_u are functions that assign to each edge $e \in E$ the lower bound $cap_l(e)$ and upper bound $cap_u(e)$ of the possible flow through e and where

$$N \;=\; \{\bot,\top\} \;\cup\; Act \times (S \uplus \overline{S})$$

$$E \;=\; \{(\bot,\langle a,t\rangle),(\langle a,\overline{u}\rangle,\top) : t \in S, a \in Act, \overline{u} \in \overline{S}\} \;\cup\; \{(\langle a,t\rangle,\langle a,\overline{u}\rangle) : (t,\overline{u}) \in R\}$$

$$cap_l(\bot,\langle a,t\rangle) \;=\; cap_l(\langle a,t\rangle,\langle a,\overline{u}\rangle) \;=\; 0, \quad cap_l(\langle a,\overline{t}\rangle,\top) \;=\; \min\ \overline{\mathbf{P}}(\overline{s},a,\overline{t})$$

$$cap_u(\bot,\langle a,t\rangle) \;=\; \mathbf{P}(s,a,t), \quad cap_u(\langle a,\overline{t}\rangle,\top) \;=\; \max\ \overline{\mathbf{P}}(\overline{s},a,\overline{t}),$$

$$cap_u(\langle a,t\rangle,\langle a,\overline{u}\rangle) \;=\; 1.$$

Similarly to Lemma 6.2.1 (page 145), $s\ sat_R\ \overline{s}$ iff either s is terminal or the maximum flow in $\mathcal{N}(s,\overline{s},R)$ is 1. The problem of finding the maximum flow in a network with lower and upper bounds can be reduced to the computation of the maximum flow in a "usual" network of the same asymptotic size (see e.g. [Even79]). Hence:

Theorem 6.2.12 *For a finite action-labelled fully probabilistic system $(S,Act,\mathbf{P})$ and a finite action-labelled fully probabilistic specification system $(\overline{S},Act,\overline{\mathbf{P}})$, the satisfaction relation $sat \subseteq S \times \overline{S}$ can be computed in time $\mathcal{O}((n+\overline{n})^7/\log(n+\overline{n}))$ and space $\mathcal{O}((n+\overline{n})^2)$ where $n = |S|$ and $\overline{n} = |\overline{S}|$.*

6.3 Proofs

This section completes the proof of Theorem 6.1.8 (page 134) by showing that the total cost $Cost_{(2.2.1)}$ of step (2.2.1) and $Cost_{(2.2.2)}$ of step (2.2.2) in the algorithm for comuting the bisimulation equivalence classes with the method sketched in Figure 6.4 (page 136) are $\mathcal{O}(mn(\log m + \log n))$ and $\mathcal{O}(mn)$ respectively.

Lemma 6.3.1 *Let X be a nonempty finite set and $(\mathcal{X}_0,\ldots,\mathcal{X}_r)$ a sequence of partitions on X such that $\mathcal{X}_i$ is finer than $\mathcal{X}_{i-1}$, $i = 1,\ldots,r$. Then,*

$$\sum_{i=0}^{r} |\mathcal{X}_i'| \;\leq\; 2(|X|-1)$$

where $\mathcal{X}_i' = \mathcal{X}_i \setminus \mathcal{X}_{i-1}$, $i = 0,\ldots,r$, and $\mathcal{X}_{-1} = \{X\}$.

Proof: Let $\mathcal{K}_X$ be the set of finite sequences $\overline{\mathcal{X}} = (\mathcal{X}_0,\ldots,\mathcal{X}_r)$ of partitions of X such that $r \geq 0$ and $\mathcal{X}_i$ is finer than $\mathcal{X}_{i-1}$, $i = 0,\ldots,r$. If $\overline{\mathcal{X}} = (\mathcal{X}_0,\ldots,\mathcal{X}_r) \in \mathcal{K}_X$ then we define $K_{\overline{\mathcal{X}}} \;=\; \sum_i |\mathcal{X}_i'|$ where $\mathcal{X}_i' = \mathcal{X}_i \setminus \mathcal{X}_{i-1}$, $i = 0,\ldots,r$. We set

$$K_X \;=\; \max\left\{K_{\overline{\mathcal{X}}} \;:\; \overline{\mathcal{X}} \in \mathcal{K}_X\right\}.$$

We show by induction on $|X|$ that $K_X \leq 2(|X|-1)$. The case $|X| = 1$ is clear. Let $|X| \geq 2$. By induction hypothesis, $K_B \leq 2|B| - 2$ for all nonempty proper subsets B

of X. Clearly, for each sequence $\overline{\mathcal{X}} = (\mathcal{X}_0, \ldots, \mathcal{X}_r) \in \mathcal{K}_X$: If $\mathcal{X}_i' = \emptyset$, $i = 0, \ldots, r$, then $K_{\overline{\mathcal{X}}} = 0$. Otherwise we may suppose w.l.o.g. that $\mathcal{X}_0' \neq \emptyset$. If $r = 0$ then

$$K_{\overline{\mathcal{X}}} \;=\; |\mathcal{X}_0'| \;\leq\; |X| \;\leq\; 2(|X|-1).$$

Now we assume that $\mathcal{X}_0' \neq \emptyset$ and $r \geq 1$. Then,

$$(*) \qquad \sum_{B \in \mathcal{X}_0'} |B| \leq |X|, \qquad |\mathcal{X}_0'| \geq 2.$$

For $B \in \mathcal{X}_0'$, let $\overline{\mathcal{X}}^B = (\mathcal{X}_1^B, \ldots, \mathcal{X}_r^B)$ where $\mathcal{X}_i^B = \{C \in \mathcal{X}_i : C \subseteq B\}$. Then, $\overline{\mathcal{X}}^B \in \mathcal{K}_B$ and $\mathcal{X}_i' = \bigcup_{B \in \mathcal{X}_0'} (\mathcal{X}_i^B \setminus \mathcal{X}_{i-1}^B)$, $i = 1, \ldots, r$. By induction hypothesis,

$$K_{\overline{\mathcal{X}}^B} \;\leq\; K_B \;\leq\; 2|B| - 2.$$

Hence, by (*):

$$K_{\overline{\mathcal{X}}} \;=\; |\mathcal{X}_0'| \;+\; \sum_{B \in \mathcal{X}_0'} K_{\overline{\mathcal{X}}^B} \;\leq\; |\mathcal{X}_0'| \;+\; 2 \cdot \sum_{B \in \mathcal{X}_0'} |B| \;-\; 2 \cdot |\mathcal{X}_0'|$$

$$\leq\; 2|X| - |\mathcal{X}_0'| \;\leq\; 2|X| - 2.$$

Thus, $K_X \;\leq\; 2(|X|-1)$. ■

Lemma 6.3.2 *Let X be a nonempty finite set and $(\mathcal{X}_0, \ldots, \mathcal{X}_r)$ a sequence of partitions on X such that $\mathcal{X}_i$ is finer than $\mathcal{X}_{i-1}$, $i = 1, \ldots, r$. Let $\mathcal{X}_{-1} = \{X\}$. For each $i \in \{0, \ldots, r\}$, let $\mathcal{Y}_i$ be a proper subset of $\mathcal{X}_i \setminus \mathcal{X}_{i-1}$ such that:*

(*) *For each $C \in \mathcal{X}_{i-1} \setminus \mathcal{X}_i$ there is some $A \in \mathcal{X}_i \setminus \mathcal{Y}_i$ with $A \subseteq C$, and $|B| \leq |A|$ for all $B \in \mathcal{Y}_i$ with $B \subseteq C$.*

Then,

$$\sum_{i=0}^{r} |\mathcal{Y}_i| \;\leq\; 2(|X|-1) \qquad \textit{and} \qquad \sum_{i=0}^{r} \sum_{B \in \mathcal{Y}_i} |B| \;\leq\; |X| \log |X|.$$

Proof: By Lemma 6.3.1 (page 153):

$$\sum_i |\mathcal{Y}_i| \;\leq\; \sum_i |\mathcal{X}_i \setminus \mathcal{X}_{i-1}| \;\leq\; 2(|X|-1).$$

We show by induction on $|X|$ that each element $x \in X$ is contained in $\bigcup_{B \in \mathcal{Y}_i} B$ for at most $\log |X|$ indices $i \in \{0, \ldots, r\}$. This yields

$$\sum_{i=0}^{r} \sum_{B \in \mathcal{Y}_i} |B| \;=\; \sum_{x \in X} \sum_{i \in I(x)} 1 \;\leq\; \sum_{x \in X} \log |X| \;=\; |X| \log |X|$$

where $I(x) = \{0 \leq i \leq r : x \in B \text{ for some } B \in \mathcal{Y}_i\}$.

We have to show that $|I(x)| \leq \log |X|$ for all $x \in X$. First we observe that (*) yields $|B| \leq |X|/2$ for all $B \in \mathcal{Y}_0 \cup \ldots \cup \mathcal{Y}_r$. If $|X| = 1$ then there is nothing to show since $\mathcal{Y}_i = \emptyset$ for all i. Let $|X| \geq 2$ and $x \in X$. We may suppose that $I(x) \neq \emptyset$, i.e. $x \in \bigcup_i \bigcup_{B \in \mathcal{Y}_i} B$.

Let i be the smallest index ≥ 0 such that $x \in B$ for some $B \in \mathcal{Y}_i$ (i.e. $i = \min I(x)$). By induction hypothesis,

$$|\{i+1, \ldots, r\} \cap I(x)| \;\leq\; \log|B|.$$

Since $|B| \leq |X|/2$ we get $|I(x)| \leq\; 1 + \log|B| \;=\; \log(2|B|) \;\leq\; \log|X|$. ■

As before, we assume $(S, \mathit{Act}, \mathit{Steps})$ to be a finite action-labelled concurrent probabilistic system. $n = |S|$ denotes the number of states, m the number of transitions, i.e. $m = \sum_{s \in S} |\mathit{Steps}(s)|$. For $a \in \mathit{Act}$, m_a is the number of a-transitions, i.e. $m_a = \sum_{s \in S} |\mathit{Steps}_a(s)|$. (Then, $m = \sum_{a \in \mathit{Act}} m_a$.) Recall that we assume Act to be fixed. Hence, we treat $|\mathit{Act}|$ as a constant.

Let $\mathcal{X}_0 = \mathcal{X}_{\mathit{init}}, \mathcal{X}_1, \ldots, \mathcal{X}_N$ be the sequence of partitions that are obtained by our algorithm (Figure 6.4, page 136). I.e.

$$\mathcal{X}_0 = \mathcal{X}_{\mathit{init}},\; \mathcal{X}_i = \mathit{Refine}(\mathcal{X}_{i-1}),\; i = 0, \ldots, N-1, \text{ and } \mathcal{X}_N = S/\sim$$

where $\mathcal{X}_{-1} = \mathcal{X}_{\mathit{trivial}} = \{S\}$. Let New_i denote the set New in the $(i+1)$-st refinement step. I.e. $\mathit{New}_0 = \mathit{New}_{\mathit{init}}$ and

$$\mathit{New}_i \;=\; \bigcup_{B \in \mathcal{X}_{i-1}} \mathit{New}_B,\; i = 1, \ldots, N-1.$$

Let $C_1^i, \ldots, C_{l_i}^i$ be the enumeration of New_i and let $C_1, \ldots, C_l$ be the sequence

$$C_1^0, \ldots, C_{l_0}^0, C_1^1, \ldots, C_{l_1}^1, C_2^1, \ldots, C_{l_2}^2, \ldots, C_1^{N-1}, \ldots, C_{l_{N-1}}^{N-1}.$$

Lemma 6.3.3 *We have $l \leq 2(n-1)$ and*

$$\sum_{i=1}^{l} |C_i| \;\leq\; n \log n.$$

Proof: We consider the set $X = S$ and the partitions $\mathcal{X}_0 = \mathcal{X}_{\mathit{init}}, \mathcal{X}_1, \ldots, \mathcal{X}_N$ of X. Then, New_i is a proper subset of $\mathcal{X}_i \setminus \mathcal{X}_{i-1}$ such that for each $C \in \mathcal{X}_{i-1} \setminus \mathcal{X}_i$ there is some $A \in \mathcal{X}_i \setminus \mathit{New}_i$ with $A \subseteq C$ and $|B| \leq |A|$ for all $B \in \mathit{New}_i$ where $B \subseteq C$ (cf. condition (*) on page 136). Lemma 6.3.2 (page 154) yields:

$$l \;=\; \sum_{i=0}^{N-1} |\mathit{New}_i| \;\leq\; 2(n-1)$$

and

$$\sum_{i=1}^{l} |C_i| \;=\; \sum_{i=0}^{N-1} \sum_{B \in \mathit{New}_i} |B| \;\leq\; n \log n$$

(where we deal with $\mathcal{Y}_i = \mathit{New}_i$.) ■

Lemma 6.3.4 *Ranging over all refinement steps, the executions of step (2.2.1) in Figure 6.4 on page 136 (i.e. the computations of $\mathit{NewCl}_*(\cdot)$ and $\mathit{OldCl}_*(\cdot)$ with the method of Figure 6.6, page 142) take $\mathcal{O}(mn(\log m + \log n))$ time.*

Proof: It suffices to show that, ranging over all blocks $B \in \mathcal{X}_0 \cup \ldots \cup \mathcal{X}_{N-1}$, the construction of the ordered balanced trees in step (1.2) of Figure 6.6 (page 142) takes $\mathcal{O}(mn(\log m + \log n))$ time.

For each $i \in \{1, \ldots, l\}$, we have to compute the probabilities $\mu[C_i]$, $\mu \in \bigcup_{a,s} \mathit{Steps}_a(s)$. For fixed i and μ, the computation of $\mu[C_i]$ takes $\mathcal{O}(|C_i|)$ time. Summing up over all distributions μ, for fixed i, the computation of the values $\mu[C_i]$ takes $\mathcal{O}(|C_i|m)$ time. Summing up over all blocks C_i and using Lemma 6.3.3 (page 155), we get the time complexity $\mathcal{O}(mn \log n)$ for the computation of the values $\mu[C_i]$, $i = 1, \ldots, l$, $\mu \in \bigcup_{a,s} \mathit{Steps}_a(s)$.

For each $i \in \{1, \ldots, l\}$, we construct an ordered balanced tree for the values $\mu[C_i]$, $\mu \in H$ and H is of the form $H = \bigcup_{s \in L} h(s)$ for some tuple (a, L, h). In that case, we speak about the (i, H)-execution of step (1.2) in Figure 6.6 (page 142). Let $\mathit{Exec}(i)$ be the set of all sets H for which there exists an (i, H)-execution of step (1.2) in Figure 6.6. Then:

- If H_1, $H_2 \in \mathit{Exec}(i)$ then $H_1 = H_2$ or $H_1 \cap H_2 = \emptyset$.
- For fixed i and H, there is at most one (i, H)-execution.

Let $T_{(i,H)}$ be the ordered balanced tree which is constructed in the (i, H)-execution. If $|T_{(i,H)}|$ denotes the number of nodes in $T_{(i,H)}$ then the (i, H)-execution causes the cost $\mathcal{O}(K_{(i,H)})$ where

$$K_{(i,H)} \;=\; |H| \log \Big(|T_{(i,H)}| + 1 \Big)$$

(and where the cost for computing the values $\mu[C_i]$, $\mu \in H$, are neglected). The total cost of all executions of step (1.2) in Figure 6.6 are $\mathcal{O}(K)$ where

$$K \;=\; \sum_{i=1}^{l} \sum_{H \in \mathit{Exec}(i)} K_{(i,H)} \;=\; \sum_{i=1}^{l} \sum_{H \in \mathit{Exec}(i)} |H| \log \Big(|T_{(i,H)}| + 1 \Big) .$$

We show that $K \leq 2nm \log m$. $\mathit{Exec}(i+1)$ consists of pairwise disjoint sets $H' \subseteq \mathit{Distr}(S)$, each of them contained in some $H \in \mathit{Exec}(i)$ (cf. condition (**) on page 138). For $i < l$ we define

$$\mathit{Split}(i, H) = \{H' : H' \in \mathit{Exec}(i+1), H' \subseteq H\}.$$

Then, $\mathit{Exec}(i+1) = \bigcup_{H \in \mathit{Exec}(i)} \mathit{Split}(i, H)$. Hence,

$$\text{(I)} \qquad K \;=\; \sum_{H \in \mathit{Exec}(1)} K'_{(1,H)}$$

where $K'_{(l,H)} \;=\; K_{(l,H)}$ and, for $i = 1, 2, \ldots, l-1$,

$$K'_{(i,H)} \;=\; |H| \log \Big(|T_{(i,H)}| + 1 \Big) \;+\; \sum_{H' \in \mathit{Split}(i,H)} K'_{(i+1,H')}.$$

By induction on i we show that

$$\text{(II)} \quad K'_{(C_{l-i},H)} \;\leq\; (i+1)|H| \log(|H| + 1) \quad \text{for all } H \in \mathit{Exec}(l-i).$$

In the basis of induction ($i = 0$) we have $|T_{(l,H)}| \leq |H|$. Hence,

$$K'_{(l,H)} \;\leq\; |H| \log(|H| + 1).$$

Induction step: Let $1 \leq i \leq l-1$, $H \in \mathit{Exec}(l-i)$ and $\mathit{Split}(l-i, H) = \{H_1, \ldots, H_r\}$. By induction hypothesis,

$$K'_{(l-i+1,H_j)} \;\leq\; i \cdot |H_j| \log(|H_j|+1),\; j = 1, \ldots, r.$$

Since $|H_1| + \ldots + |H_r| \leq |H|$ and $|T_{(l-i,H)}| \leq |H|$ we get

$$K'_{(l-i,H)} \;\leq\; |H| \log\left(|T_{(l-i,H)}|+1\right) \;+\; i \cdot \sum_{j=1}^{r} |H_j| \log\left(|H_j|+1\right)$$

$$\leq\; |H| \log(|H|+1) \;+\; i \cdot |H| \log(|H|+1) \;=\; (i+1)|H| \log(|H|+1).$$

Since $\mathit{Exec}(1) = \{\bigcup_{s \in S} \mathit{Steps}_a(s) : a \in \mathit{Act}\} \setminus \{\emptyset\}$ we get

$$\sum_{H \in \mathit{Exec}(1)} |H| = \sum_{a \in \mathit{Act}} \sum_{s \in S} |\mathit{Steps}_a(s)| \;=\; m.$$

Hence, by (I) and (II):

$$K \;=\; \sum_{H \in \mathit{Exec}(1)} K'_{(1,H)} \;\leq\; \sum_{H \in \mathit{Exec}(1)} l \cdot |H| \log |H| \;\leq\; lm \log m.$$

By Lemma 6.3.3 (page 155), we have $l \leq 2n$. Therefore,

$$K \;\leq\; lm \log m \;\leq\; 2nm \log m.$$

Thus, we get the time complexity $\mathcal{O}(mn \log m)$ for the constructions of the trees in step (1.2) in Figure 6.6 (page 142) where we neglect the cost for computing the values $\mu[C_i]$. Adding the cost for the computations of the values $\mu[C_i]$ we obtain the time complexity $\mathcal{O}(mn(\log m + \log n))$ for all executions of step (2.2.1) in the main algorithm. ■

Lemma 6.3.5

$$\sum_{i=0}^{N-1} \sum_{B \in \mathcal{X}_i} |\mathit{NewCl}_{\mathcal{X}_i}| \;\leq\; 2(m-1).$$

Proof: Let $M_i = \sum_{B \in \mathcal{X}_i} |\mathit{NewCl}_{\mathcal{X}_i}(B)|$. Then, $M_i = \sum_{a \in \mathit{Act}} |\mathcal{H}_{a,i}|$ where

$$\mathcal{H}_{a,i} \;=\; \left\{ \bigcup_{s \in L} h(s) : B \in \mathcal{X}_i, (a, L, h) \in \mathit{NewCl}_{\mathcal{X}_i}(B) \right\}.$$

We consider the set $X_a = \bigcup_{s \in S} \mathit{Steps}_a(s)$ of all a-labelled transitions in $(S, \mathit{Act}, \mathit{Steps})$. The sets $\mathcal{H}_{a,i}$ can be extended to partitions $\mathcal{X}_{a,i}$ of X_a such that $\mathcal{X}_{a,i}$ is finer than $\mathcal{X}_{a,i-1}$ and $\mathcal{H}_{a,i} \subseteq \mathcal{X}_{a,i} \setminus \mathcal{X}_{a,i-1}$ (cf. condition (**) on page 138). Thus,

$$\sum_{i=0}^{N-1} M_i \;=\; \sum_{a \in \mathit{Act}} \sum_{i=0}^{N-1} |\mathcal{H}_{a,i}| \;\leq\; \sum_{a \in \mathit{Act}} \sum_{i=0}^{N-1} |\mathcal{X}'_{a,i}|$$

$$\leq\; 2(|X|-1) \;\leq\; \sum_{a \in \mathit{Act}} 2(m_a - 1) \;=\; 2(m-1)$$

by Lemma 6.3.1 (page 153). Here, $\mathcal{X}'_{a,i} = \mathcal{X}_{a,i} \setminus \mathcal{X}_{a,i-1}$. ■

Lemma 6.3.6 *Summing up over all refinement steps, the executions of step (2.2.2) in Figure 6.4 on page 136 (the computations of $B/\equiv_{\mathcal{X}}$ with the method decsribed on page 139) take $\mathcal{O}(nm)$ time.*

Proof: We define $K_i = \max\{|NewCl_{\mathcal{X}_i}(B)| : B \in \mathcal{X}_i\}$. In the $(i+1)$-st refinement step (i.e. in the computation of $\mathcal{X}_{i+1} = \mathit{Refine}(\mathcal{X}_i)$), step (2.2.2) causes the cost $\mathcal{O}(n \cdot K_i)$ since, for each state $s \in S$, we traverse a binary tree of height $\leq K_i$. By Lemma 6.3.5 (page 157):

$$\sum_{i=0}^{N-1} K_i \;\leq\; \sum_{i=0}^{N-1} \sum_{B \in \mathcal{X}_i} |NewCl_{\mathcal{X}_i}| \;\leq\; 2(m-1).$$

Thus, step (2.2.2) causes the cost $\mathcal{O}(nm)$. ■

Chapter 7

Weak bisimulation for fully probabilistic processes

In the non-probabilistic case, weak [Miln80, Park81, Miln89] or branching [vGlWe89] bisimulation equivalence are fundamental for verification methods that exploit *abstraction from internal computation* as – being compositional with respect to parallel composition and other operators – they make it possible to replace components by equivalent ones that are minimized with respect to their internal behaviour. Clearly, also in the probabilistic setting, appropriate notions of weak equivalence together with efficient decision procedures are highly desirable. Testing equivalences for fully probabilistic systems that abstract from internal computations are proposed by Ivan and Linda Christoff [Chri90a, Chri90b, ChCh91, Chri93] (who also present polynomial time decision procedures) and Cleaveland et al [CSZ92, YCDS94]. For the latter, the authors present a proof technique but do not investigate the decidability. Segala & Lynch [SeLy94] introduce notions of weak and branching bisimulation for concurrent probabilistic systems that appear as natural extensions of weak and branching equivalences for non-probabilistic systems. Recent work shows that weak [PSS98] and branching [BSV98] bisimulation equivalence are decidable for finite concurrent probabilistic systems. This chapter (whose main results are developed in team work with Holger Hermanns [BaHe97]) proposes notions of weak bisimulation and branching bisimulation in the fully probabilistic setting and presents a polynomial decision procedure.

Weak bisimulation in non-probabilistic systems: The weak bisimulation equivalence classes of a finite (non-probabilistic) labelled transition system $(S, Act, \longrightarrow)$ can be computed as the (strong) bisimulation equivalence classes of the induced system $(S, (Act \setminus \{\tau\}) \cup \{\varepsilon\}, \Longrightarrow)$. Here, the "double arrow relation"

$$\Longrightarrow \subseteq \; S \times ((Act \setminus \{\tau\}) \cup \{\varepsilon\}) \times S$$

is defined with the help of the transitive, reflexive closure $(\stackrel{\tau}{\longrightarrow})^*$ of internal transitions.[1] Thus, the problem of deciding weak bisimulation equivalence is reduced to the computation of the transitive, reflexive closure $(\stackrel{\tau}{\longrightarrow})^*$ of the internal transitions and deciding

[1] For the empty word ε, the transition relation $\stackrel{\varepsilon}{\Longrightarrow}$ agrees with $(\stackrel{\tau}{\longrightarrow})^*$ (i.e. $s \stackrel{\varepsilon}{\Longrightarrow} t$ asserts that t is reachable from s via internal actions) while $\stackrel{\alpha}{\Longrightarrow} = (\stackrel{\tau}{\longrightarrow})^* \stackrel{\alpha}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^*$.

(strong) bisimulation equivalence in a finite system. Using the transitive closure operation from [CoWi87] and the partitioning/splitter technique by [PaTa87] the time complexity for deciding weak bisimulation equivalence is $\mathcal{O}(n^{2.3})$ where n is the number of states. Groote & Vaandrager [GroVa90] propose an algorithm for computing the branching bisimulation equivalence classes of a non-probabilistic system which works with a variant of the partitioning/splitter technique à la [PaTa87] that uses both transition relations $\longrightarrow$ and $\Longrightarrow$ and runs in time $\mathcal{O}(nm)$ where n is the number of states and m the number of transitions (i.e. the size of $\longrightarrow$).

Weak bisimulation in fully probabilistic systems: This chapter introduces notions of weak bisimulation and branching bisimulation for fully probabilistic systems. The basic idea is to replace Milner's "double arrow relation" $s \stackrel{\alpha}{\Longrightarrow} t$ by the probabilities to reach state t from s via a sequence of transitions labelled by a trace of the form $\tau^*\alpha\tau^*$. In contrast to the non-probabilistic case where branching bisimulation is strictly finer than weak bisimulation, weak and branching bisimulation equivalence coincide for finite fully probabilistic systems. The proposed notion of weak (or branching) bisimulation equivalence is *decidable* for finite systems. We present an algorithm to compute the weak bisimulation equivalence classes with a modification of the partitioning/splitter technique à la [KaSm83, PaTa87]. The time complexity of our method is *cubic* in the number of states; thus, it meets the worst case complexity for deciding branching bisimulation in the non-probabilistic case [GroVa90] (where, in the worst case, $\mathcal{O}(m) = \mathcal{O}(n^2)$). Moreover, weak bisimulation is shown to be a *congruence* with respect to the operators of *PLSCCS* (see Section 4.3, page 83 ff) with the exception of the probabilistic choice operator.[2] Therefore, weak bisimulation is applicable for mechanised compositional verification of probabilistic systems that work with the lazy product $\mathcal{P}_1 \otimes \mathcal{P}_2$ as parallel composition.

Organization of that chapter: Section 7.1 introduces weak and branching bisimulations. In Section 7.2 we present our algorithm for deciding weak bisimulation equivalence. Section 7.3 discusses the connection between weak (and branching) bisimulation equivalence and other equivalences for fully probabilistic systems. The congruence result is established in Section 7.4. Most of the proofs for the results of this chapter are given in the appendix (Section 7.5). The proofs use the regularity of certain matrices (with columns and rows for each state of the underlying system). Thus, the main results are only established for *finite* systems. It is an open question whether our results carry over to arbitrary fully probabilistic systems (with possibly infinitely many states).

This chapter makes use of the notations for partitions as explained in Section 2.1 (page 29) and for ordered balanced trees (see Section 12.2, page 314). Moreover, we often use the probabilities $Prob(s, \Omega, t)$ for s to reach a C-state via a path whose trace belong to Ω (see Section 3.3.1, page 49). Throughout this chapter, we deal with action-labelled fully probabilistic systems.

[2] The fact that weak (and branching) bisimulation equivalence are not preserved by probabilistic choice is not surprising as already in the non-probabilistic case, weak and branching bisimulation equivalence fail to be congruences with respect to non-deterministic choice.

7.1 Weak and branching bisimulation

In this section we define weak and branching bisimulation for fully probabilistic systems. While in the non-probabilistic case branching bisimulation equivalence is strictly finer than weak bisimulation equivalence, these two relations coincide for finite fully probabilistic systems (Theorem 7.1.10, page 163).

7.1.1 Weak bisimulation

For the definition of weak bisimulation, we replace Milner's "double arrow" relation $\stackrel{\varepsilon}{\Longrightarrow} = (\stackrel{\tau}{\longrightarrow})^*$ (where $s \stackrel{\varepsilon}{\Longrightarrow} t$ states that s can move to t via internal steps) by the probability $Prob(s, \tau^*, t)$ to reach state t from s via internal actions. Similarly, for $\alpha \in Act \setminus \{\tau\}$, we deal with the probabilities $Prob(s, \tau^*\alpha\tau^*, t)$ rather than Milner's weak transition relations $\stackrel{\alpha}{\Longrightarrow} = (\stackrel{\tau}{\longrightarrow})^* \stackrel{\alpha}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^*$.[3]

Definition 7.1.1 [**Weak bisimulation**] *A* weak bisimulation *on an action-labelled fully probabilistic system $(S, Act, \mathbf{P})$ is an equivalence relation R on S such that for all $(s, s') \in R$ and all equivalence classes $C \in S/R$:*

(1) $Prob(s, \tau^*, C) = Prob(s', \tau^*, C)$

(2) $Prob(s, \tau^*\alpha\tau^*, C) = Prob(s', \tau^*\alpha\tau^*, C)$ *for all* $\alpha \in Act \setminus \{\tau\}$.

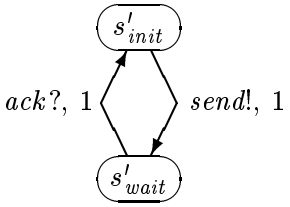
Two states s, s' are called weakly bisimilar *(denoted by $s \approx s'$) iff $(s, s') \in R$ for some weak bisimulation R.*

Remark 7.1.2 Note that $Prob(s, \tau^*, C) = 1$ if $s \in C$. Hence, condition (1) is always fulfilled for the equivalence class C of s and s' with respect to R. ■

In Section 7.5.1 (Lemma 7.5.16, page 185) we show that, for finite systems, $\approx$ is a weak bisimulation. Two fully probabilistic processes $\mathcal{P} = (S, Act, \mathbf{P}, s_{init})$, $\mathcal{P}' = (S', Act, \mathbf{P}', s'_{init})$ are said to be *weakly bisimilar* iff their initial states s_{init} and s'_{init} are weakly bisimilar in the composed system which is defined as explained in Section 3.5 (page 61).

Example 7.1.3 We consider the simple communication protocol of Example 3.3.2 (page 48) and the fully probabilistic system for the sender shown in Figure 3.2 on page 48.

Using weak bisimulation equivalence as the underlying implementation relation the sender can be verified against the specification given by the fully probabilistic process shown on the right.



For this, we have to show that the initial states s_{init} and s'_{init} are weakly bisimilar. Let R be the equivalence on $S = \{s_{init}, s_{del}, s_{wait}, s_{lost}, s'_{init}, s'_{wait}\}$ such that $S/R = \{C_I, C_W\}$ where $C_I = \{s_{init}, s'_{init}\}$ is the equivalence class of the initial states and $C_W = \{s_{del}, s_{wait}, s_{lost}, s'_{wait}\}$ the equivalence class of the other states. For $s_I \in C_I$ and $s_W \in C_W$,

[3] See Section 3.3.1, page 49, for the definition of $Prob(s, \Omega, t)$.

we have:

$$\begin{array}{ll}
Prob(s_I, \tau^*, C_I) = 1, & Prob(s_W, \tau^*, C_I) = 0, \\
Prob(s_I, \tau^*, C_W) = 0, & Prob(s_W, \tau^*, C_W) = 1, \\
Prob(s_I, \tau^*\ send!\ \tau^*, C_I) = 0, & Prob(s_W, \tau^*\ send!\ \tau^*, C_I) = 0, \\
Prob(s_I, \tau^*\ ack?\ \tau^*, C_I) = 0, & Prob(s_W, \tau^*\ ack?\ \tau^*, C_I) = 1, \\
Prob(s_I, \tau^*\ send!\ \tau^*, C_W) = 1, & Prob(s_W, \tau^*\ send!\ \tau^*, C_W) = 0, \\
Prob(s_I, \tau^*\ ack?\ \tau^*, C_W) = 0, & Prob(s_W, \tau^*\ ack?\ \tau^*, C_W) = 0.
\end{array}$$

Hence, R is a weak bisimulation. In particular, the initial states s_{init} of the sender and s'_{init} of its specification are weakly bisimilar. ■

In the non-probabilistic case, it holds for weakly bisimilar states s, s' that if $s \stackrel{\alpha_1 \ldots \alpha_k}{\Longrightarrow} t$ then $s' \stackrel{\alpha_1 \ldots \alpha_k}{\Longrightarrow} t'$ such that t and t' are weakly bisimilar. Here, $\stackrel{\alpha_1 \ldots \alpha_k}{\Longrightarrow}$ denotes $(\stackrel{\tau}{\longrightarrow})^* \stackrel{\alpha_1}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^* \ldots (\stackrel{\tau}{\longrightarrow})^* \stackrel{\alpha_k}{\longrightarrow} (\stackrel{\tau}{\longrightarrow})^*$. This result carries over to finite fully probabilistic systems.

Theorem 7.1.4 *Let $(S, Act, \mathbf{P})$ be a finite action-labelled fully probabilistic system and Ω a regular expression of the form $\tau^*\alpha_1\tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k$ or $\tau^*\alpha_1\tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k\tau^*$. Then:*

$$\textit{If } s \approx s' \textit{ then } Prob(s, \Omega, C) = Prob(s', \Omega, C) \textit{ for all } C \in S/\approx.$$

Proof: see Section 7.5.1, Theorem 7.5.17 (page 186). ■

7.1.2 Branching bisimulation

Van Glabbeek & Weijland [vGlWe89] introduce branching bisimulation which is strictly finer than weak bisimulation. The basic idea of branching bisimulation is that in order to simulate a step $s \stackrel{\alpha}{\longrightarrow} t$ by an equivalent state s', s' is allowed to perform arbitrary many internal actions leading to a state s'' which is still equivalent to s and s' (i.e. the intermediate states on the path from s' to s'' also fall in the equivalence class of s and s') and then to perform α reaching a state t' which is equivalent to t. In the probabilistic case, we require that for equivalent states s, s', the probabilities for s and s' to perform internal actions inside the equivalence class of s and s' and then to perform a visible action α leading to state of a certain equivalence class C are the same.

Notation 7.1.5 [**The symbols $\widehat{a}$,** $a \in Act$] *For $a \in Act$, let*

$$\widehat{a} \;=\; \begin{cases} a & : \textit{ if } a \neq \tau \\ \varepsilon & : \textit{ if } a = \tau. \end{cases}$$

Recall that ε denotes the empty word in Act^*. Hence, $\tau^*\widehat{a} = \tau^*$ if $a = \tau$.

Notation 7.1.6 [**The probabilities** $Prob_R(s, \tau^*\widehat{a}, C)$] *Let $(S, Act, \mathbf{P})$ be an action-labelled fully probabilistic system, R an equivalence relation on S, $s \in S$, $C \subseteq S$ and $a \in Act$. Then, $Path^R_{ful}(s, \tau^*\widehat{a}, C)$ denotes the set of fulpaths $\pi \in Path_{ful}(s)$ such that there is some $k \geq 0$ with*

- $(s, \pi(i)) \in R$, $i = 1, \ldots, k-1$,
- $trace(\pi^{(k)}) \in \tau^*\widehat{a}$,

- $\pi(k) \in C$.

Let $Prob_R(s, \tau^* \widehat{a}, C) = Prob(Path^R_{ful}(s, \tau^* \widehat{a}, C))$, $Prob_R(s, \tau^* \widehat{a}, t) = Prob((s, \tau^* \widehat{a}, \{t\})$.

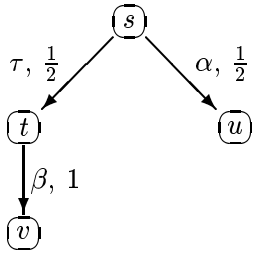
Remark 7.1.7 For $s \in C$, $Path_{ful}(s) = Path^R_{ful}(s, \tau^*, C)$. Hence, $Prob_R(s, \tau^*, C) = 1$ if $s \in C$. ■

Example 7.1.8 For the relation R in Example 7.1.3, page 161, we have

$$Prob_R(s, \tau^* \widehat{a} \tau^*, C) \;=\; Prob(s, \tau^* \widehat{a} \tau^*, C)$$

for all states s and $C \in \{C_I, C_W\}$ and $a \in \{\tau, send!, ack?\}$.

For the system shown on the right and the "identity relation" R (i.e. the equivalence relation R with $(x, y) \in R$ iff $x = y$) we have $Prob_R(s, \tau^* \beta, v) = 0$ while $Prob(s, \tau^* \beta, v) = 1/2$. ■



Definition 7.1.9 [Branching bisimulation] *Let* $(S, Act, \mathbf{P})$ *be an action-labelled fully probabilistic system. A* branching bisimulation *on* $(S, Act, \mathbf{P})$ *is an equivalence relation* R *on* S *such that for all* $(s, s') \in R$, $C \in S/R$*:*

(1) $Prob_R(s, \tau^*, C) \;=\; Prob_R(s, \tau^*, C)$
(2) $Prob_R(s, \tau^* \alpha, C) \;=\; Prob_R(s, \tau^* \alpha, C)$ *for all* $\alpha \in Act \setminus \{\tau\}$.

Two states s, s' *are called* branching bisimilar *(denoted* $s \approx_{br} s'$*) iff* $(s, s') \in R$ *for some branching bisimulation* R.

In Section 7.5.1, Lemma 7.5.15 (page 185) we show that, for $(S, Act, \mathbf{P})$ to be finite, branching bisimulation equivalence $\approx_{br}$ is a branching bisimulation. In contrast to the non-probabilistic case, branching bisimulation equivalence and weak bisimulation equivalence coincide for finite systems.

Theorem 7.1.10 *Let* $(S, Act, \mathbf{P})$ *be a finite action-labelled fully probabilistic system and* s, $s' \in S$. *Then,* $s \;\approx\; s'$ *iff* $s \;\approx_{br}\; s'$.

Proof: see Section 7.5.1, Corollary 7.5.13 (page 184). ■

The classical example for distinguishing weak and branching bisimulation equivalence is the system shown in Figure 7.1 on page 164 (see [vGlWe89]). In the non-probabilistic case, s and s' are weakly but not branching bisimilar. If we add non-zero probabilities (which turn the system of Figure 7.1 into a fully probabilistic system) then s and s' are no longer weakly bisimilar. This can be seen as follows. We assume $s \approx s'$. Then, $1 \;=\; Prob(s, \tau^* \alpha \tau^*, T) \;=\; Prob(s', \tau^* \alpha \tau^*, T)$ where T denotes the weak bisimulation equivalence class of t. Clearly, $v'' \not\approx t$ as t can perform γ (since $\mathbf{P}(t, \gamma) > 0$) while v'' cannot (since $Prob(v'', \tau^* \gamma) = 0$). Hence, $v'' \notin T$ and therefore $\mathbf{P}(s', \alpha, t') = 1$, $\mathbf{P}(s', \alpha, v'') = 0$. Contradiction (as we added non-zero probabilities to the non-probabilistic system of Figure 7.1).

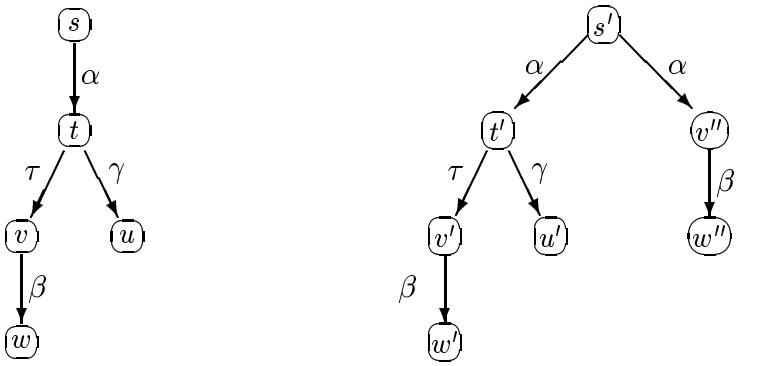


Figure 7.1: Distinguishing weak and branching bisimulation in the non-probabilistic case

7.2 Decidability of weak bisimulation equivalence

In this section we develop an algorithm to compute the weak bisimulation equivalence classes. The general idea of our algorithm is to use a partitioning/splitter-technique similar to the ones proposed by Kanellakis & Smolka [KaSm83] resp. Paige & Tarjan [PaTa87] for deciding strong bisimulation in the non-probabilistic case (cf. the schema sketched in Section 6.1, Figure 6.1 on page 131). The algorithm starts with some "simple" partition $\mathcal{X}_{init}$ that is coarser than $\approx$ and then successively refines the given partition $\mathcal{X}$ with the help of a "splitter" of $\mathcal{X}$, eventually resulting in the set of weak bisimulation equivalence classes. The crucial point is the definition of a splitter. A possible candidate for a "splitter" of a partition $\mathcal{X}$ is a pair $(a, C) \in Act \times \mathcal{X}$ that violates the condition for $\mathcal{X}$ to be a weak bisimulation, i.e.

$$(*)\ \ Prob(s, \tau^* \widehat{a} \tau^*, C) \ \neq \ Prob(s', \tau^* \widehat{a} \tau^*, C) \text{ for some } B \in \mathcal{X} \text{ and } s,\ s' \in B.$$

One idea for a partioning/splitter-algorithm would be to refine $\mathcal{X}$ according to a splitter in the sense of (*), i.e. to replace $\mathcal{X}$ by $Refine'(\mathcal{X}, a, C) \ = \ \{B/ \simeq_{(a,C)} \colon B \in \mathcal{X}\}$ where

$$s \simeq_{(a,C)} s' \ \text{ iff } \ Prob(s, \tau^* \widehat{a} \tau^*, C) = Prob(s', \tau^* \widehat{a} \tau^*, C).$$

The probabilities $Prob(s, \tau^* \widehat{a} \tau^*, C)$ can be computed by solving the linear equation system

$$\begin{array}{lcll} x_s & = & 1 & \text{if } a = \tau \text{ and } s \in C \\ x_s & = & 0 & \text{if } Path_{ful}(s, \tau^* \widehat{a} \tau^*, C) = \emptyset \\ x_s & = & \displaystyle\sum_{t \in S} \mathbf{P}(s, \tau, t) \cdot x_t \ + \ \mathbf{P}(s, a, C) & \\ & & & \text{if } Path_{ful}(s, \tau^* \widehat{a} \tau^*, C) \neq \emptyset \text{ and } a \neq \tau \vee s \notin C. \end{array}$$

(cf. Proposition 3.3.4, page 49). The test whether $Path_{ful}(s, \tau^* \widehat{a} \tau^*, C) = \emptyset$ can be done by a reachability analysis of the underlying directed graph, e.g. with a depth first search like method. Then, $\Omega(n^{3.8})$ is an asymptotic lower bound for the time complexity of this method.[4] Here, we present an alternative method that runs in time $\mathcal{O}(n^3)$. The

[4]Here, n is the number of states. Note that in the worst case we need n refinement steps and in each refinement step we have to solve a linear equation system with n variables and n equations (which takes $\Omega(n^{2.8})$ time with the method of [AHU74]).

basic idea is to replace (*) by a condition that asserts that $\mathcal{X}$ violates the conditions of a branching bisimulation. For this, we use an alternative definition of a splitter that is based on an characterization of branching bisimulations which uses the *conditional probabilities* $\mathbf{P}_{\mathcal{X}}(s,a,C)$ to reach a block C from a state s via an action a within one step under the condition that the system does not make an internal move inside the block that contains s. These conditional probabilities can be computed by simple *arithmetic operations*. Thus, the use of this kind of splitters has the advantage that in the refinement steps we do not have to solve linear equation systems.

7.2.1 The algorithm

In what follows, we fix a finite action-labelled fully probabilistic system $(S, Act, \mathbf{P})$ and a partition $\mathcal{X}$ of S. We say that $\mathcal{X}$ is a weak (branching) bisimulation iff the induced equivalence relation $R_{\mathcal{X}}$ is a weak (branching) bisimulation.

Notation 7.2.1 [The set S_{term} of terminal states] *S_{term} denotes the set of terminal states (i.e. all states $s \in S$ where $\mathbf{P}(s,a,t) = 0$ for all $a \in Act$ and $t \in S$).*

Notation 7.2.2 [The set $S_{\mathcal{X}}$] *We define* $S_{\mathcal{X}} = \{s \in S \setminus S_{term} : \mathbf{P}(s,\tau,[s]_{\mathcal{X}}) < 1\}$.[5]

Thus, $S_{\mathcal{X}}$ contains all states that, with non-zero probability, can either perform something visible (in the case where $\mathbf{P}(s,\tau) < 1$) or silently step into a different class (in the case where $\mathbf{P}(s,\tau,t) > 0$ for some $t \notin [s]_{\mathcal{X}}$).

Notation 7.2.3 [The conditional probabilities $\mathbf{P}_{\mathcal{X}}(s,a,t)$] *If $s \in S_{\mathcal{X}}$ and $(a,C) \in Act \times \mathcal{X}$ with $(a,C) \neq (\tau,[s]_{\mathcal{X}})$ then we define*

$$\mathbf{P}_{\mathcal{X}}(s,a,C) \;=\; \frac{\mathbf{P}(s,a,C)}{1-\mathbf{P}(s,\tau,[s]_{\mathcal{X}})}.$$

$\mathbf{P}_{\mathcal{X}}(s,a,C)$ is the conditional probability for s to reach C via action a under the condition that in state s the system does not make a τ-move inside the block $[s]_{\mathcal{X}}$ of s (i.e. the system either performs a visible action or makes a τ-move to another block). We get the following alternative characterization of branching bisimulations that refers to the conditional transition probabilities $\mathbf{P}_{\mathcal{X}}(\cdot)$ rather than the values $Prob_{R_{\mathcal{X}}}(\cdot)$.

Lemma 7.2.4 *$\mathcal{X}$ is a branching bisimulation iff, for all $B \in \mathcal{X}$ with $B \cap S_{\mathcal{X}} \neq \emptyset$:*

(1) $\mathbf{P}_{\mathcal{X}}(s,a,C) = \mathbf{P}_{\mathcal{X}}(s',a,C)$
for all s, $s' \in B \cap S_{\mathcal{X}}$ and $(a,C) \in Act \times \mathcal{X}$ with $(a,C) \neq (\tau,B)$.

(2) *If $s_0 \in B \setminus S_{\mathcal{X}}$ then there exists a finite path σ with*
- *$first(\sigma) = s_0$,*
- *$\sigma(i) \in B \setminus S_{\mathcal{X}}$, $i = 0,1,\ldots,|\sigma|-1$,*
- *$last(\sigma) \in B \cap S_{\mathcal{X}}$.*

Moreover, if $\mathcal{X}$ is a branching bisimulation then for all $B \in \mathcal{X}$ with $B \cap S_{\mathcal{X}} \neq \emptyset$ and $s \in B$:

[5] Recall that $[s]_{\mathcal{X}}$ denotes the unique block in $\mathcal{X}$ that contains s (cf. Section 2.1, page 29).

- $Prob_{R_{\mathcal{X}}}(s, \tau^*, C) = \mathbf{P}_{\mathcal{X}}(B, \tau, C)$ *for all* $C \in \mathcal{X}$
- $Prob_{R_{\mathcal{X}}}(s, \tau^*\alpha, C) = \mathbf{P}_{\mathcal{X}}(B, \alpha, C)$ *for all* $\alpha \in Act \setminus \{\tau\}$ *and* $C \in \mathcal{X}$.

Here,

$$\mathbf{P}_{\mathcal{X}}(B, a, C) \;=\; \begin{cases} \mathbf{P}_{\mathcal{X}}(t, a, C) & : \; \textit{if } (a, C) \neq (\tau, B) \textit{ and } t \in B \cap S_{\mathcal{X}} \\ 1 & : \; \textit{if } (a, C) = (\tau, B). \end{cases}$$

Proof: see Section 7.5.1, Lemma 7.5.9 (page 181). ■

Remark 7.2.5 Let $\mathcal{X}$ be a branching bisimulation and $B \in \mathcal{X}$ such that $B \cap S_{\mathcal{X}} = \emptyset$. For all $s \in B$, either s is terminal or $\mathbf{P}(s, \tau, B) = 1$. In either case, $Prob_{R_{\mathcal{X}}}(s, \tau^*\alpha, C) = 0$ and $Prob_{R_{\mathcal{X}}}(s, \tau^*, C) = 0$ if $C \in \mathcal{X} \setminus \{B\}$. Hence, if we put $\mathbf{P}_{\mathcal{X}}(B, \tau, B) = 1$ and $\mathbf{P}_{\mathcal{X}}(B, a, C) = 0$ if $(a, C) \neq (\tau, B)$. then we get

$$Prob_{R_{\mathcal{X}}}(s, \tau^*\widehat{a}, C) \;=\; \mathbf{P}_{\mathcal{X}}(B, a, C)$$

for all $s \in B$, $C \in \mathcal{X}$ and $a \in Act$. ■

Example 7.2.6 Consider the system of Figure 7.2 (page 167) and the partition $\mathcal{X} = \{B_1, B_2, B_3\}$ where $B_1 = \{s_0, s, s'\}$, $B_2 = \{t, t'\}$ and $B_3 = \{w, w', v, v'\}$. We show that all blocks of $\mathcal{X}$ satisfy the conditions of Lemma 7.2.4. We have $S_{term} = \{w, w', v\}$, $\mathbf{P}(s_0, \tau, [s_0]_{\mathcal{X}}) = 1$ and $\mathbf{P}(v', \tau, [v']_{\mathcal{X}}) = 1$. Thus, $S_{\mathcal{X}} = \{s, s', t, t'\}$.

- For the block B_1, we first consider the states s and s'. We have:
  $$\mathbf{P}_{\mathcal{X}}(s, \tau, B_2) = \mathbf{P}_{\mathcal{X}}(s', \tau, B_2) = \tfrac{1}{2}, \quad \mathbf{P}_{\mathcal{X}}(s, \alpha, B_3) = \mathbf{P}_{\mathcal{X}}(s', \alpha, B_3) = \tfrac{1}{2}$$
  and $\mathbf{P}_{\mathcal{X}}(s, a, C) = \mathbf{P}_{\mathcal{X}}(s', a, C) = 0$ for all $(a, C) \notin \{(\tau, B_2), (\alpha, B_3)\}$. Hence, B_1 satisfies condition (1). Second we show condition (2) for B_1. For this, we have to consider the state $s_0 \in B_1 \setminus S_{\mathcal{X}}$. The finite path leading from s_0 to a state of $B_1 \cap S_{\mathcal{X}}$ is given by $s_0 \xrightarrow{\tau} s$.
- For the block $B_2 = \{t, t'\}$ we get: $\mathbf{P}_{\mathcal{X}}(t, \beta, B_3) = \mathbf{P}_{\mathcal{X}}(t', \beta, B_3) = 1$ and $\mathbf{P}_{\mathcal{X}}(t, a, C) = \mathbf{P}_{\mathcal{X}}(t', a, C) = 0$ for all $(a, C) \neq (\beta, B_3)$. Hence, B_2 satisfies (1). As $B_2 \cap S_{\mathcal{X}} = \emptyset$, B_2 fulfills condition (2).
- As $B_3 \cap S_{\mathcal{X}} = \emptyset$ for the block B_3 there is nothing to show.

Lemma 7.2.4 yields that $\mathcal{X}$ is a branching bisimulation. ■

Remark 7.2.7 If $\mathcal{X}$ is a branching bisimulation, $B \in \mathcal{X}$ and s, $s' \in B \cap S_{\mathcal{X}}$ then

$$\frac{\mathbf{P}(s, \tau, B)}{1 - \mathbf{P}(s, \tau, B)} \;\neq\; \frac{\mathbf{P}(s', \tau, B)}{1 - \mathbf{P}(s', \tau, B)}$$

is possible. For instance, for the states s and s' in Example 7.2.6 (Figure 7.2 on page 167) we have $s \approx_{br} s'$ but $\mathbf{P}(s, \tau, B_1)/(1 - \mathbf{P}(s, \tau, B_1)) = 0$ while $\mathbf{P}(s', \tau, B_1)/(1 - \mathbf{P}(s', \tau, B_1)) = 1/2$ (where $B_1 = \{s_0, s, s'\}$ is the branching bisimulation equivalence class of s and s'). ■

Definition 7.2.8 [Splitter] *A* splitter *of a partition $\mathcal{X}$ is a tuple (a, C) consisting of an action $a \in Act$ and some $C \in \mathcal{X}$ such that there exists some $B \in \mathcal{X}$ with $(\tau, B) \neq (a, C)$ and $\mathbf{P}_{\mathcal{X}}(s, a, C) \;\neq\; \mathbf{P}_{\mathcal{X}}(s', a, C)$ for some states s, $s' \in B \cap S_{\mathcal{X}}$.*

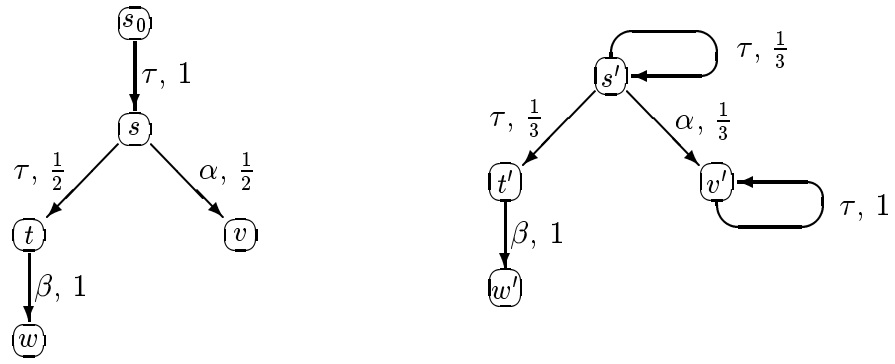


Figure 7.2:

The main idea for refining a given partition $\mathcal{X}$ via a splitter (a, C) is to isolate in each $B \in \mathcal{X}$ with $(\tau, B) \neq (a, C)$ those states s, $s' \in B \cap S_{\mathcal{X}}$ where $\mathbf{P}_{\mathcal{X}}(s, a, C) = \mathbf{P}_{\mathcal{X}}(s', a, C)$. By condition (2) of Lemma 7.2.4, each such equivalence class A of $B \cap S_{\mathcal{X}}$ has to be enriched with exactly those states $s \in B \setminus S_{\mathcal{X}}$ that can reach A via internal actions and that cannot reach any other equivalence class A' of $B \cap S_{\mathcal{X}}$ without passing A.

Notation 7.2.9 [**The set** $Split(B, a, C)$] *Let (a, C) be a splitter of a partition $\mathcal{X}$ and $B \in \mathcal{X}$ such that $(\tau, B) \neq (a, C)$. We define*

$$Split(B, a, C) \;=\; (B \cap S_{\mathcal{X}})/\equiv_{\mathcal{X}}$$

where, for s, $s' \in B \cap S_{\mathcal{X}}$, $s \equiv_{\mathcal{X}} s'$ iff $\mathbf{P}_{\mathcal{X}}(s, a, C) = \mathbf{P}_{\mathcal{X}}(s', a, C)$.

Notation 7.2.10 [**The closure** $\overline{A}$] *Let (a, C) be a splitter of a partition $\mathcal{X}$ and $B \in \mathcal{X}$ such that $(\tau, B) \neq (a, C)$. If $A \in Split(B, a, C)$ then we define the* closure $\overline{A}$ *of A in $\mathcal{X}$ with respect to (a, C) to be the largest set $V \subseteq B$ which contains A and such that for all $s \in V \setminus A$:*

- $\mathbf{P}(s, \tau, V \cup A) = 1$
- *There exists a finite path σ with*
  - *first*$(\sigma) = s$,
  - $\sigma(i) \in V$, $i = 0, 1, \ldots, |\sigma| - 1$,
  - *last*$(\sigma) \in A$.

Notation 7.2.11 [**The residuum** $Res(B, a, C)$] *Let $\mathcal{X}$, a, B and C be as before. The* residuum *of B with respect to (a, C) is given by*

$$Res(B, a, C) = \{B'\} \setminus \{\emptyset\} \quad \textit{where} \quad B' \;=\; B \setminus \bigcup_{A \in Split(B,a,C)} \overline{A}.$$

Remark 7.2.12 Note that the residuum $Res(B, a, C)$ is either empty (if all states $s \in S \setminus S_{\mathcal{X}}$ are contained in some closure $\overline{A}$) or a singleton set consisting of all states $s \in B \setminus S_{\mathcal{X}}$ that do not belong to any closure $\overline{A}$. If $A \in Split(B, a, C)$ then $\overline{A}$ consists of A and all states $s \in B \setminus S_{\mathcal{X}}$ such that

- *last*$(\sigma) \in A$ for some $\sigma \in \Sigma(s)$
- Whenever $\sigma \in \Sigma$ and *last*$(\sigma) \in S_{\mathcal{X}}$ then *last*$(\sigma) \in A$.

Here, $\Sigma(s) = \{\sigma \in Path_{fin}(s) : first(\sigma) = s, \sigma(i) \in B \setminus S_{\mathcal{X}},\ i = 0, 1, \ldots, |\sigma| - 1\}$. ■

Notation 7.2.13 [**The refinement operator** *Refine*$(\cdot)$] *Let $\mathcal{X}$ be a partition, (a,C) a splitter of $\mathcal{X}$. For $B \in \mathcal{X}$, we define:*

- *If $(a,C) = (\tau,B)$ then $\mathit{Refine}(B,a,C) = \{B\}$.*
- *If $(a,C) \neq (\tau,B)$ then*

$$\mathit{Refine}(B,a,C) \;=\; \{\overline{A} : A \in \mathit{Split}(B,a,C)\} \cup \mathit{Res}(B,a,C).$$

We define $\mathit{Refine}(\mathcal{X},a,C) \;=\; \bigcup_{B\in\mathcal{X}} \mathit{Refine}(B,a,C)$.

Clearly, for each partition $\mathcal{X}$ which is coarser than $S/\approx_{br}$ and each splitter (a,C) of $\mathcal{X}$, the partition $\mathit{Refine}(\mathcal{X},a,C)$ is coarser than $S/\approx_{br}$ and strictly finer than $\mathcal{X}$.

Our refinement operator preserves condition (2) of Lemma 7.2.4 (page 165). More precisely, if $B \in \mathcal{X}$ such that $B \cap S_{\mathcal{X}} = \emptyset$ and condition (2) of Lemma 7.2.4 is fulfilled then all blocks $A \in \mathit{Refine}(B,a,C)$ fulfill condition (2). Moreover, if $\mathcal{X}$ is a partition that fulfills condition (2) of Lemma 7.2.4 and that is coarser than $S/\approx_{br}$ and there is no splitter for $\mathcal{X}$ then $\mathcal{X} \;=\; S/\approx_{br} \;=\; S/\approx$. These observations lead to the following algorithm. We start with a "simple" partition $\mathcal{X}$ that satisfies condition (2) of Lemma 7.2.4 and that is coarser than $\approx$. Then – as long as $\mathcal{X}$ can be refined (i.e. as long as there exists a splitter for $\mathcal{X}$) – we apply the refinement operator to $\mathcal{X}$, eventually resulting in the partition $\mathcal{X} \;=\; S/\approx$.

As the initial partition has to fulfill condition (2) of Lemma 7.2.4 we cannot start with the "trivial" partition $\mathcal{X} = \{S\}$ that identifies all states as it might violate condition (2). For instance, for a system with two states, a terminal state t and a state s with $\mathbf{P}(s,\alpha,t) = 1$, the trivial partition $\mathcal{X}_{\mathit{trivial}} = \{\{s,t\}\}$ does not have a splitter. Hence, if we would start with $\mathcal{X}_{\mathit{trivial}}$ then our algorithm would return that s and t are weakly bisimilar which is not the case. Our initial partition consists two blocks: the weak bisimulation equivalence class of the terminal states and its complement. (Of course, if one of these blocks is empty then we only start with one block.) To be precise, the weak bisimulation equivalence class of the terminal states consists of all "divergent" states, i.e. all states that cannot reach a state where a visible action can be performed with some non-zero probability.

Definition 7.2.14 [**Divergent states**] *A state s is called* divergent *iff $\mathit{Path}_{\mathit{ful}}(s,\tau^*\alpha) = \emptyset$ for all $\alpha \in \mathit{Act} \setminus \{\tau\}$. Let Div be the set of divergent states.*

Note that $S_{\mathit{term}} \subseteq \mathit{Div}$. Our algorithm for computing the weak bisimulation equivalence classes is sketched in Figure 7.3 on page 169.

Example 7.2.15 Partitioning the system from Example 7.2.6 (Figure 7.2, page 167) proceeds as follows. The initial partition is $\mathcal{X} \;=\; \{\{s_0,s,s',t,t'\},\{w,w',v,v'\}\}$. Then, $S_{\mathcal{X}} \;=\; \{s,s',t,t'\}$. $(\alpha,\{w,w',v,v'\})$ and $(\beta,\{s_0,s,s',t,t'\})$ are splitters of $\mathcal{X}$. For the splitter $(\alpha,\{w,w',v,v'\})$ we obtain

$$\mathbf{P}_{\mathcal{X}}(s,\alpha,\{w,w',v,v'\}) \;=\; \frac{1}{2} \;=\; \frac{\frac{1}{3}}{1-\frac{1}{3}} \;=\; \mathbf{P}_{\mathcal{X}}(s',\alpha,\{w,w',v,v'\})$$

and $\mathbf{P}_{\mathcal{X}}(t,\alpha,\{w,w',v,v'\}) = 0 = \mathbf{P}_{\mathcal{X}}(t',\alpha,\{w,w',v,v'\})$. Hence, the split operator separates s and s' from t and t'. More precisely, we get:

$$\begin{aligned}\mathit{Split}(\{s_0,s,s',t,t'\},\alpha,\{w,w',v,v'\}) \;&=\; \{\{s,s'\},\{t,t'\}\},\\ \mathit{Split}(\{w,w',v,v'\},\alpha,\{w,w',v,v'\}) \;&=\; \{\{w,w',v,v'\}\}.\end{aligned}$$

Computing the weak bisimulation equivalence classes

Input: a finite action-labelled fully probabilistic system $(S, Act, \mathbf{P})$

Output: the set $S/\approx$ of weak bisimulation equivalence classes

Method:

Compute the set *Div* of divergent states;

$\mathcal{X} := \{Div, S \setminus Div\} \setminus \{\emptyset\}$;

While $\mathcal{X}$ contains a splitter (a, C) do $\mathcal{X} := Refine(\mathcal{X}, a, C)$;

Return $\mathcal{X}$.

Figure 7.3: Schema for computing the weak bisimulation equivalence classes

The closure operator yields $\overline{\{s, s'\}} = \{s_0, s, s'\}$. Hence, we get the partition

$$Refine(\mathcal{X}, \alpha, \{w, w', v, v'\}) \;=\; \{\{s_0, s, s'\}, \{t, t'\}, \{w, w', v, v'\}\}$$

for which no splitter exists. Thus, our algorithm returns $Refine(\mathcal{X}, \alpha, \{w, w', v, v'\})$ as the set of weak bisimulation equivalence classes. ■

7.2.2 Time complexity

In what follows, $n = |S|$. We suppose that the alphabet *Act* is fixed (thus, we treat the size $|Act|$ as a constant).

Theorem 7.2.16 *The algorithm of Figure 7.3 (page 169) can be implemented in time* $\mathcal{O}(n^3)$ *and space* $\mathcal{O}(n^2)$.

Proof: Clearly, *Div* can be computed by a reachability analysis in the underlying directed graph. We compute all states that can reach a state of $\{t \in S : \mathbf{P}(t, \alpha) > 0 \text{ for some } \alpha \in Act \setminus \{\tau\}\}$, e.g. by a depth first search. Thus, the computation of the initial partition $\mathcal{X}$ needs $\mathcal{O}(n^2)$ time and space.

Initialization of the refine step: Let $\mathcal{X}$ be the current partition. We compute the values $\mathbf{P}(s, a, C)$ and $\mathbf{P}_{\mathcal{X}}(s, a, C)$ for each $s \in S$, $a \in Act$ and $C \in \mathcal{X}$. The set $S_{\mathcal{X}}$ can be derived from the probabilities $\mathbf{P}_{\mathcal{X}}(s, \tau, C)$, $s \in C$. For each pair (a, C) (where $a \in Act$ and $C \in \mathcal{X}$) and $A \in \mathcal{X}$ we compute

$$min(A, a, C) \;=\; \min_{s \in A} \mathbf{P}_{\mathcal{X}}(s, a, C), \quad max(A, a, C) \;=\; \max_{s \in A} \mathbf{P}_{\mathcal{X}}(s, a, C).$$

Then, (a, C) is a splitter of $\mathcal{X}$ iff $min(A, a, C) < max(A, a, C)$ for some A with $(a, C) \neq (\tau, A)$. If there is no splitter of $\mathcal{X}$ then $\mathcal{X} = S/\approx$. Otherwise we choose some splitter (a, C) of $\mathcal{X}$.

Refinement step: For all $B \in \mathcal{X}$ with $(\tau, B) \neq (a, C)$ we compute the set $Refine(B, a, C)$ as follows. We construct an ordered binary tree $Tree(B)$ by successively inserting the

values $\mathbf{P}_{\mathcal{X}}(s,a,C)$, $s \in B \cap S_{\mathcal{X}}$. (See Section 12.2, page 314 for the notations that we use for ordered balanced trees.) Each node v of $\mathit{Tree}(B)$ is represented as a record with components $v.\mathit{key}$ and $v.\mathit{states}$. For each state $s \in B \cap S_{\mathcal{X}}$, we traverse the tree $\mathit{Tree}(B)$ starting in the root and we search for the value $\mathbf{P}_{\mathcal{X}}(s,a,C)$.

- If we reach a node v with $v.\mathit{key} = \mathbf{P}_{\mathcal{X}}(s,a,C)$ then we insert s into $v.\mathit{states}$.
- Otherwise, $\mathbf{P}_{\mathcal{X}}(s,a,C)$ is not yet represented in $\mathit{Tree}(B)$ and we insert a node v with $v.\mathit{key} = \mathbf{P}_{\mathcal{X}}(s,a,C)$ and $v.\mathit{states} = \{s\}$.

In the final tree, $v.\mathit{states}$ is the set of states $s \in B \cap S_{\mathcal{X}}$ with $\mathbf{P}_{\mathcal{X}}(s,a,C) = v.\mathit{key}$. Thus, the nodes of the final tree $\mathit{Tree}(B)$ represent the sets $A \in \mathit{Split}(B,a,C)$. More precisely,

$$\mathit{Split}(B,a,C) \;=\; \{v.\mathit{states} : v \text{ is a node in } \mathit{Tree}(B)\}.$$

We derive $\mathit{Refine}(B,a,C)$ as follows. Let G_B be the directed graph (B, E_B) where $(s,t) \in E_B$ iff $\mathbf{P}(t,\tau,s) > 0$ and $t \in B \setminus S_{\mathcal{X}}$. We compute the sets $\overline{A}$, $A \in \mathit{Split}(B,a,C)$, by the following breadth first search like method. We use three kinds of labels for the states:

- $\mathit{label}(s) = \bot$ iff $s \in B \setminus S_{\mathcal{X}}$ and s is not yet visited.
- $\mathit{label}(s) = A \in \mathit{Split}(B,a,C)$ iff s is reachable in G_B from some state in A but there is no other $A' \in \mathit{Split}(B,a,C)$ where a path from a state of A' to s in G_B is already detected.
- $\mathit{label}(s) = *$ iff there are two sets A, $A' \in \mathit{Split}(B,a,C)$ such that s is reachable from a state in A and from a state in A'. (In particular,, all successors of a $*$-labelled state in G_B are also labelled by $*$.)

Initially, we define $\mathit{label}(s) = A$ for all $s \in A$ and $A \in \mathit{Split}(B,a,C)$ and $\mathit{label}(s) = \bot$ for all $s \in B \setminus S_{\mathcal{X}}$. We use a queue Q which initially contains the states $s \in A$, $A \in \mathit{Split}(B,a,C)$. While Q is not empty we take the first element s of Q, remove s from Q and, if $\mathit{label}(s) \neq *$ then, for all $t \in B \setminus S_{\mathcal{X}}$ with $(s,t) \in E_B$, we do:

(1) If $\mathit{label}(t) = \bot$ then we add t to Q and set $\mathit{label}(t) = \mathit{label}(s)$.

(2) If $\mathit{label}(t) \in \mathit{Split}(B,a,C)$, $\mathit{label}(t) \neq \mathit{label}(s)$, then we set $\mathit{label}(u) = *$ for $u = t$ and all successors u of t in G_B.[6]

Then, $\overline{A} = \{s \in B : \mathit{label}(s) = A\}$, $\mathit{Res}(B,a,C) = \{\{s \in B : \mathit{label}(s) \in \{\bot, *\}\}\} \setminus \{\emptyset\}$.

Complexity: It is clear that the method described above can be implemented in space $\mathcal{O}(n^2)$. We show that the time complexity of our method is $\mathcal{O}(n^3)$. First, we observe that there are at most n iterations of the refinement step. Thus, it suffices to show that each refinement step takes time $\mathcal{O}(n^2)$.

Clearly, for each iteration (i.e. each refinement step), the initialization requires $\mathcal{O}(n^2)$ time.[7] Ranging over all B, the construction of the trees $\mathit{Tree}(B)$ (thus, the computation of the sets $\mathit{Split}(B,A,C)$) takes $\mathcal{O}(n \log n)$ time if one uses some kind of ordered balanced trees (see Section 12.2, page 314). We show that, ranging over all $B \in \mathcal{X}$, the sets $\overline{A}$ and

[6] For this, we might use a depth first search starting in t to find all successors of t. States that are already labelled by $*$ are ignored.

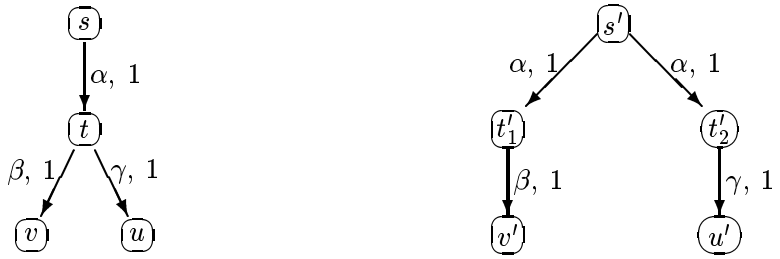
[7] Note that, for each tuple (s,a,C), we have to calculate $\sum_{t \in C} \mathbf{P}(s,a,t)$. Hence, for fixed a and ranging over all $s \in S$, $C \in \mathcal{X}$, we get the time complexity $\mathcal{O}(n^2)$. Since we suppose Act to be fixed, the values $\mathbf{P}(s,a,C)$ can be computed in time $\mathcal{O}(n^2)$.

$Res(B, a, C)$ can be derived in time $\mathcal{O}(n^2)$: For fixed $B \in \mathcal{X}$, the directed graph G_B can be constructed in time $\mathcal{O}(|B|^2)$. Each state $s \in B$ is added to Q at most once.[8] Each state t which is visited by a depth first search in step (2) is labelled by $*$. Thus, it can never be visited in step (2) once again. As a consequence, each state causes time costs (at most) of order $2n$ in the computation of $Refine(B, a, C)$: as an element of Q and as a state with label $\neq *$ that is visited in step (2). Either case involves $\mathcal{O}(n)$ computations. Summing up over all $s \in B$, the computation of $Refine(B, a, C)$ has time complexity $\mathcal{O}(|B| \cdot n)$. So, we obtain $Refine(\mathcal{X}, a, C)$ in time $\mathcal{O}(n^2)$. ■

7.3 Connection to other equivalences

In this section we discuss how the proposed notion of weak bisimulation equivalence relates to other equivalences for fully probabilistic systems.

Clearly, weak bisimulation $\approx$ is strictly coarser than (strong) bisimulation à la Larsen & Skou [LaSk89] (Definition 3.4.3, page 54) which does not abstract from internal moves. Formally, if $(S, Act, \mathbf{P})$ is a fully probabilistic system and s, s' are bisimilar states then s and s' are weakly bisimilar. Moreover, if the system is τ-free (i.e. $\mathbf{P}(t, \tau) = 0$ for all states t) then weak bisimulation equivalence $\approx$ and (strong) bisimulation equivalence $\sim$ coincide.[9] Of course, we cannot expect $\approx$ to be comparable with strong trace, failure or ready equivalence in the sense of Jou & Smolka [JoSm90] as $\approx$ abstracts from the internal steps while the equivalences of [JoSm90] do not treat the τ-steps in a special way and are strictly coarser than strong (and weak) bisimulation for τ-free systems. For instance, the states s and s' of the system below are strongly trace equivalent but not (strongly or weakly) bisimilar.



Vice versa, the states s and s' of the system $(\{s, s', t\}, \{\tau, \alpha\}, \mathbf{P})$ where $\mathbf{P}(s, \tau, s') = \mathbf{P}(s', \alpha, t) = 1$ and $\mathbf{P}(\cdot) = 0$ in all other cases are weakly bisimilar but not strong trace, failure or ready equivalent in the sense of [JoSm90]. Dealing with the "weak" counterparts of the equivalences proposed in [JoSm90], Theorem 7.1.4 (page 162) yields that, for finite systems, $\approx$ is strictly finer than weak trace, failure or ready equivalence. Here, e.g. s, s' are called *weakly trace equivalent* iff

$$Prob(s, \tau^*\alpha_1\tau^* \ldots \tau^*\alpha_k\tau^*) = Prob(s', \tau^*\alpha_1\tau^* \ldots \tau^*\alpha_k\tau^*)$$

for all $k \geq 0$ and $\alpha_1, \ldots, \alpha_k \in Act \setminus \{\tau\}$.

Christoff [Chri90b, Chri90a] and Cleaveland et al [CSZ92] (see also [YCDS94]) introduce *testing equivalences* for finite action-labelled fully probabilistic processes that relate two

[8] Note that only states with label $\perp$ can be added to Q.

[9] Note that, for $(S, Act, \mathbf{P})$ to be τ-free, $Prob(s, \tau^*\alpha\tau^*, C) = \mathbf{P}(s, \alpha, C)$.

processes in terms of the reliability in certain environments. While [Chri90b] deal with deterministic environments [CSZ92] consider probabilistic testing scenarios. Both abstract from internal computations. In the remainder of this section we discuss the relation between these testing preorders and our notion of weak bisimulation. For this, we fix a finite action-labelled fully probabilistic system $(S, Act, \mathbf{P})$.

Testing equivalence à la Christoff: We show that weak bisimulation $\approx$ is stronger than the testing equivalences introduced by Christoff [Chri90b] (see also [Chri90a, ChCh91]). [Chri90b] distinguishes fully probabilistic processes through the conditional probabilities of certain deterministic testing scenarios. The several testing scenarios lead to the definitions of *probabilistic trace equivalence* $=_{tr}$, *weak probabilistic testing equivalence* $=_{wte}$ and *strong probabilistic testing equivalence* $=_{ste}$. As shown in [Chri90b], $=_{tr} \supseteq =_{wte} \supseteq =_{ste}$. We show that weak bisimulation equivalence $\approx$ is stronger than strong probabilistic testing equivalence $=_{ste}$ (and thus, it is also stronger than $=_{wte}$ and $=_{tr}$).

We briefly recall the definition of strong probabilistic testing equivalence. More precisely, we use an equivalent characterization of $=_{ste}$ which is given in [ChCh91].

Notation 7.3.1 [**The set** *Offerings*] *Let Offerings be the set of nonempty subsets of* $Act \setminus \{\tau\}$ *(the set of* offerings*) and* $Offerings^*$ *the set of (finite) strings of offerings.* ε_{Off} *denotes the empty string of offerings.*

For $L_1, \ldots, L_k \in Offerings$ and $\alpha_1, \ldots, \alpha_r \in Act \setminus \{\tau\}$, $Q(s, L_1 \ldots L_k, \alpha_1 \ldots \alpha_r, t)$ denotes the probability for performing the string $\tau^*\alpha_1 \ldots \tau^*\alpha_r$ ending up in t when offered a string of $L_1 \ldots L_k$. The formal definition of $Q(\cdot)$ is as follows.

Notation 7.3.2 [**The values** $Q(s, \tilde{L}, \tilde{\alpha}, C)$] *The function*

$$Q : S \times Offerings^* \times (Act \setminus \{\tau\})^* \times 2^S \to [0,1]$$

is defined as follows. Let $s \in S$, $C \subseteq S$, $L \in Offerings$, $\alpha \in Act \setminus \{\tau\}$, $\tilde{L} \in Offerings^*$, $\tilde{\alpha} \in (Act \setminus \{\tau\})^*$.

$$\begin{aligned}
Q(s, \varepsilon_{Off}, \tilde{\alpha}, C) &= 0 \quad \textit{if } \tilde{\alpha} \neq \varepsilon \\
Q(s, \tilde{L}, \varepsilon, C) &= \begin{cases} 1 & : \textit{ if } s \in C \\ 0 & : \textit{ otherwise} \end{cases} \\
Q(s, L\tilde{L}, \alpha\tilde{\alpha}, C) &= \sum_{u \in S} Q(s, L, \alpha, C) \cdot Q(u, \tilde{L}, \tilde{\alpha}, C) \\
Q(s, L, \alpha, C) &= 0 \quad \textit{if } \alpha \notin L
\end{aligned}$$

If $\alpha \in L$ *then the values* $Q(s, L, \alpha, C)$, $s \in S$, $C \subseteq S$ *are the unique solution of the following linear equation system.*

1. $Q(s, L, \alpha, C) = 0$ *if* $Prob(s, \tau^*\alpha, C) = 0$.
2. *If* $Prob(s, \tau^*\alpha, C) > 0$ *then*

$$Q(s, L, \alpha, C) = \frac{\mathbf{P}(s, \alpha, C)}{\mathbf{P}(s, \tau) + \mathbf{P}(s, L)} + \sum_{u \in S} \frac{\mathbf{P}(s, \tau, u)}{\mathbf{P}(s, \tau) + \mathbf{P}(s, L)} \cdot Q(s, L, \alpha, C).$$

Note that $Prob(s, \tau^*\alpha, C) > 0$, $\alpha \in L$ implies $\mathbf{P}(s, \tau) + \mathbf{P}(s, L) > 0$.

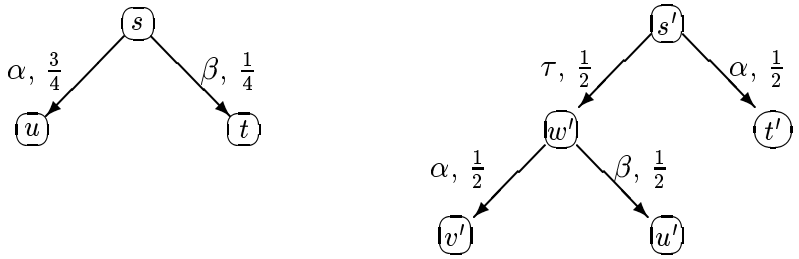


Figure 7.4: $s =_{ste} s'$ but $s \not\approx s'$

Notation 7.3.3 [The values $Q(s, \tilde{L}, \tilde{\alpha})$] *If $\tilde{L} \in Offerings^*$ and $\tilde{\alpha} \in (Act \setminus \{\tau\})^*$ then we put $Q(s, \tilde{L}, \tilde{\alpha}) = Q(s, \tilde{L}, \tilde{\alpha}, S)$.*

Definition 7.3.4 [The testing equivalence $=_{ste}$, cf. [Chri90b, ChCh91]]

$$s =_{ste} s' \text{ iff } Q(s, \tilde{L}, \tilde{\alpha}) = Q(s', \tilde{L}, \tilde{\alpha}) \text{ for all } \tilde{L} \in Offerings^*,\ \tilde{\alpha} \in (Act \setminus \{\tau\})^*.$$

Theorem 7.3.5 *$\approx$ is strictly finer than $=_{ste}$.*

Proof: In Section 7.5.2, Theorem 7.5.19 (page 188), we show that $\approx$ is finer than $=_{ste}$. To see that $=_{ste}$ and $\approx$ do not coincide consider the fully probabilistic system of Figure 7.4 (page 173). Then, $s =_{ste} s'$ as, for instance,

$$Q(s, \{\alpha, \beta\}, \alpha) = \tfrac{3}{4} = \tfrac{1}{2} + \tfrac{1}{2} \cdot \tfrac{1}{2} = Q(s', \{\alpha, \beta\}, \alpha)$$

and $Q(s, \{\alpha\}, \alpha) = 1 = Q(s', \{\alpha\}, \alpha)$. On the other hand,

$$Prob(s', \tau^*\alpha, S) = 3/4 > 1/2 = Prob(w', \tau^*\alpha, S).$$

Hence, $s' \not\approx w'$. Thus, $Prob(s', \tau^*, W) = 1/2 > 0 = Prob(s, \tau^*, W)$ where W is the weak bisimulation equivalence class of w'. Thus, $s \not\approx s'$. ■

[ChCh91] presents algorithms for deciding the three kinds of equivalences which are based on solving linear equation systems and run in time $\mathcal{O}(n^4)$ where n is the number of states of the underlying system. In contrast to this, the use of weak (or branching) bisimulation has the advantage that it allows the use of the conditional probabilities $\mathbf{P}_{\mathcal{X}}(\cdot)$ which can be computed by simple arithmetic operations (rather than solving linear equation systems).

Testing equivalences à la Cleaveland et al [CSZ92]: [CSZ92] (see also [YCDS94] present quantitative extensions of the non-probabilistic testing preorders by de Nicola & Hennessy [dNHe83, Henn88]. Given a test $\mathcal{T}$ – which is represented by a fully probabilistic system equipped with a set of *success* states – the probability for a fully probabilistic process $\mathcal{P}$ to *pass* the test $\mathcal{T}$ is defined as the probability measure of the set of "interaction sequences" leading to a success state. Intuitively, given a class of tests, two fully probabilistic processes $\mathcal{P}$, $\mathcal{P}'$ are testing equivalent with respect to a certain class of tests iff $\mathcal{P}$ and $\mathcal{P}'$ pass all tests $\mathcal{T}$ of that class with the same probability. [CSZ92] consider two classes of tests:

- The class $Tests_0$ of τ-free tests which yields the testing equivalence denoted by $\equiv_0$.
- The class *Tests* of all tests that do not contain "τ-loops" which yields the testing equivalence denoted by $\equiv$.

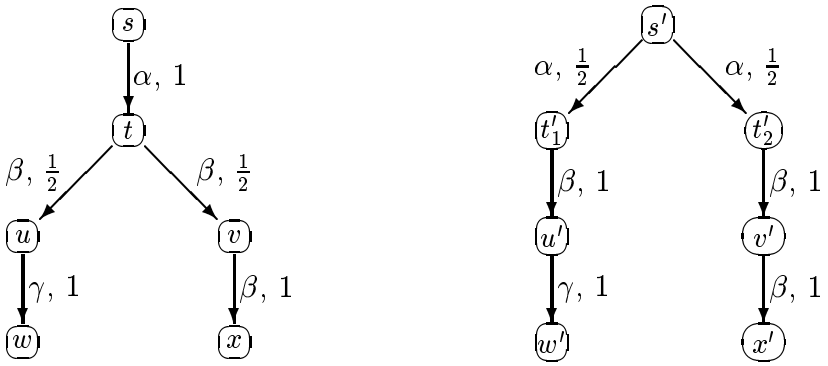


Figure 7.5: $s \equiv s'$ but $s \not\approx s'$

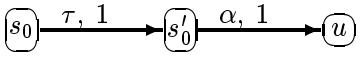
The exact definition (more precisely, an alternative characterization) of $\equiv_0$ is given in Section 7.5.2 (page 188 ff) where we prove that $\equiv_0$ is coarser than $\approx$. For the precise definition of $\equiv$ see [CSZ92] or [YCDS94].

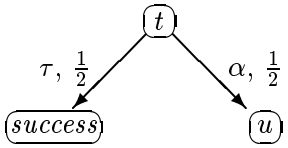
Theorem 7.3.6

(a) $\approx$ is strictly finer than $\equiv_0$.

(b) $\approx$ and $\equiv$ are not comparable.

Proof: In Section 7.5.2, Theorem 7.5.27 (page 191), we show that $\approx$ is finer than $\equiv_0$. As shown in [YCDS94], the states s and s' of the system shown in Figure 7.5 (page 174) are testing equivalent with respect to $\equiv$ (and hence, testing equivalent with respect to $\equiv_0$). On the other hand, s and s' are not weakly bisimilar as $Prob(s, \tau^*\alpha\tau^*, T) = 1$ while $Prob(s', \tau^*\alpha\tau^*, T) = 0$ where T denotes the weak bisimulation equivalence class of t. (Note that neither t'_1 nor t'_2 is weakly bisimilar to t.) The states s_0 and s'_0 of the system shown below are weakly bisimilar while $s_0 \not\equiv s'_0$.





For instance, the test $\mathcal{T}$ shown on the right distinguishes the states s_0 and s'_0. The probability for s_0 to pass the test $\mathcal{T}$ is $3/4$ while the probability for s'_0 to pass $\mathcal{T}$ is $1/2$. ■

7.4 Compositionality

We establish the congruence result (Theorem 7.4.2, page 175) stating the compositionality of weak bisimulation equivalence with respect to the operators of *PLSCCS*.[10] More precisely, we show that weak bisimulation equivalence $\approx$ is a congruence with respect to the *PLSCCS* operators action prefixing, restriction, relabelling, lazy product and guarded probabilistic choice.

[10] Recall the syntax and semantics of the lazy synchronous calculus *PLSCCS* which was introduced in Section 4.3 (page 83 ff).

In what follows, we shrink our attention to *finitary PLSCCS* programs, i.e. programs $\mathcal{P}$ where the associated fully probabilistic process $\mathcal{O}[\![\mathcal{P}]\!]$ is finite (or can be identified with a finite process).

Notation 7.4.1 [Finitary *PLSCCS* programs] *A PLSCCS program $\langle decl, s\rangle$ is called* finitary *iff there are only finitely many statements $t \in Stmt_{\mathbf{0}}$ that are reachable from s in $(Stmt_{\mathbf{0}}, Act_0, \mathbf{P}^{decl})$. A declaration $decl : ProcVar \to Stmt_{PLSCCS}$ is called finitary iff, for each $Z \in ProcVar$, $\langle decl, Z\rangle$ is finitary.*

If $\mathcal{P}$ is finitary then $\mathcal{O}[\![\mathcal{P}]\!]$ can be identified with the *finite* fully probabilistic process that arises from $\mathcal{O}[\![\mathcal{P}]\!]$ by removing all statements $t \in Stmt_{\mathbf{0}}$ that are not reachable from the initial state. Clearly, if *decl* is finitary then, for each statement s, $\langle decl, s\rangle$ is finitary.[11] For *PLSCCS* programs $\mathcal{P}$, $\mathcal{P}'$, we define $\mathcal{P} \approx \mathcal{P}'$ iff $\mathcal{O}[\![\mathcal{P}]\!] \approx \mathcal{O}[\![\mathcal{P}']\!]$. For fixed declaration *decl*, we define the relations $\approx^{decl}$ for *PLSCCS* statements by $s \approx^{decl} s'$ iff $\langle decl, s\rangle \approx \langle decl, s'\rangle$.

Theorem 7.4.2 *Weak bisimulation equivalence is preserved by the PLSCCS operators action prefixing, restriction, relabelling and lazy product. More precisely, if decl is a finitary declaration, then for all PLSCCS statements s, s', s_i, s_i':*

(a) If $s \approx^{decl} s'$ then $a.s \approx^{decl} a.s'$, $s \setminus L \approx^{decl} s' \setminus L$ and $s[\ell] \approx^{decl} s'[\ell]$.

(b) If $s_i \approx^{decl} s_i'$, $i = 1, 2$, then $s_1 \otimes s_2 \sim^{decl} s_1' \otimes s_2'$; and thus,

$$s_1 \otimes s_2 \approx^{decl} s_1' \otimes s_2'.$$

(c) Weak bisimulation equivalence is a congruence with respect to guarded probabilistic choice, i.e. if $s_i \approx^{decl} s_i'$, $i \in I$, then

$$\sum_{i\in I} [p_i]a_i.s_i \approx^{decl} \sum_{i\in I} [p_i]a_i.s_i'.$$

Proof: Part (a) is an easy verification. We show (b) and (c). More precisely, we fix a finitary declaration *decl*, some finite subsets S_1, S_2, of $Stmt_{\mathbf{0}}$ that contain $\mathbf{0}$ and that are closed with respect to the transition relation induced by $\mathbf{P}^{decl}$ (i.e. if $t \in S_i$ and $\mathbf{P}^{decl}(t, a, u) > 0$ then $u \in S_i$). We show that

$$R = \Big\{(s_1 \otimes s_2, s_1' \otimes s_2') : s_i, s_i' \in S_i, s_i \approx^{decl} s_i', i = 1, 2\Big\}$$

is a bisimulation (in the sense of Definition 3.4.3, page 54). Here, we put $t \otimes \mathbf{0} = \mathbf{0} \otimes t = \mathbf{0}$ and $t \approx^{decl} \mathbf{0}$ iff $\mathcal{O}[\![\langle decl, t\rangle]\!] \approx (Stmt_{\mathbf{0}}, Act_0, \mathbf{P}^{decl}, \mathbf{0})$. For subsets C_1 of S_1 and C_2 of S_2, we define

$$C_1 \otimes C_2 = \{s_1 \otimes s_2 : s_1 \in C_1, s_2 \in C_2\}.$$

Clearly, R is an equivalence relation on $S_1 \otimes S_2$. Each equivalence class $C \in S/R$ is of the form $C = C_1 \otimes C_2$ where $C_i \in S_i/\approx^{decl}$, $i = 1, 2$. Let $a \in Act$, $C = C_1 \otimes C_2 \in S/R$

[11] A sufficient condition which guarantees that *decl* is finitary is the "simplicity" of *decl* in the sense that, for all process variables Z, $Z' \in ProcVar$, there is no occurrence of Z' in $decl(Z)$ that is contained in a substatement of the form $t[\ell]$, $t \setminus L$ or $t_1 \otimes t_2$.

and $(s, s') \in R$ where $s = s_1 \otimes s_2$, $s' = s'_1 \otimes s'_2$, $s_i \approx^{decl} s'_i$, $i = 1, 2$. Then, by Theorem 7.1.4 (page 162) and Corollary 4.3.2 (page 84):

$$\mathbf{P}^{decl}(s, a, C) = \sum_{(\alpha_1,\alpha_2)\in Syn_a} Prob^{decl}(s_1, \tau^*\alpha_1, C_1) \cdot Prob^{decl}(s_2, \tau^*\alpha_2, C_2)$$

$$= \sum_{(\alpha_1,\alpha_2)\in Syn_a} Prob^{decl}(s'_1, \tau^*\alpha_1, C_1) \cdot Prob^{decl}(s'_2, \tau^*\alpha_2, C_2) = \mathbf{P}^{decl}(s', a, C).$$

Similarly, Corollary 4.3.3 (page 84) yields that $\mathbf{P}^{decl}(s, 0, C_{\mathbf{0}}) = \mathbf{P}^{decl}(s', 0, C_{\mathbf{0}})$ where $C_{\mathbf{0}} = [\mathbf{0}]_R$ is the equivalence class of $\mathbf{0}$ with respect to R. We conclude $\mathbf{P}^{decl}(s, a, C) = \mathbf{P}^{decl}(s', a, C)$ for all $a \in Act_0$ and $C \in S/R$. Hence, R is a bisimulation. In particular, whenever $(s, s') \in R$ then $s \approx^{decl} s'$. This yields the claim of part (b). Part (c) can be derived from Theorem 7.1.4 (page 162) and the fact that, for $C \subseteq Stmt_{\mathbf{0}}$ and $s = \sum_{i\in I} [p_i]a_i.s_i$,

$$Prob^{decl}(s, \tau^*a, C) = \sum_{i\in I_\tau} p_i \cdot Prob^{decl}(s_i, \tau^*a, C) + \sum_{j\in J} p_j$$

where $I_\tau = \{i \in I : a_i = \tau\}$ and $J = \{i \in I : a_i = a, s_i \in C\}$. ■

Example 7.4.3 We consider the *PLSCCS* program $Sender \otimes Receiver$ of Example 4.3.6 on page 86 which we verify against the specification

$$Spec \stackrel{\text{def}}{=} produce.consume.Spec.$$

Clearly, the operational semantics of $Sender \otimes Receiver$ (shown in Figure 4.14 on page 88) and the operational semantics of *Spec* are weakly bisimilar. On the other hand, our congruence result (part (b) of Theorem 7.4.2) allows us to use "modular verification" (i.e. to verify the components *Sender* and *Receiver* separately) avoiding the construction of $\mathcal{O}[\![Sender \otimes Receiver]\!]$ and using

$$Sender\_Spec \stackrel{\text{def}}{=} produce.deliver!wait.ack?.Sender\_Spec$$

as the specification for the sender. Clearly, $\mathcal{O}[\![Sender]\!]$ (shown in Figure 4.13 on page 87) and $\mathcal{O}[\![Sender\_Spec]\!]$ are weakly bisimilar. Thus, by Theorem 7.4.2 (page 175):

$$Sender \otimes Receiver \approx Sender\_Spec \otimes Receiver.$$

It is easy to see that $Spec \approx Sender\_Spec \otimes Receiver$ which yields that $Sender \otimes Receiver$ and *Spec* are weakly bisimilar (by the transitivity of $\approx$). ■

Of course, we cannot expect weak bisimulation equivalence to be a congruence for the synchronous parallel composition of *PSCCS* as *PSCCS* does not treat the internal action τ in any distinguished way. For example, if $s_1 = \alpha.nil$, $s'_1 = \tau.\alpha.nil$ and $s_2 = \beta.nil$ then $s_1 \times s_2$ can make a $\alpha * \beta$-move while $s'_1 \times s_2$ preforms $\tau * \beta$. Thus, if $\alpha * \beta \neq \tau * \beta$ then $s_1 \times s_2$ and $s'_1 \times s_2$ are not weakly bisimilar while s_1 and s'_1 are. The counterexample that demonstrates that weak bisimulation equivalence is not a congruence for the probabilistic choice operator is almost the same as the counterexample in the non-probabilistic case which shows that weak bisimulation equivalence is not preserved by non-deterministic choice. Consider the *PLSCCS* statements $s_1 = \alpha.nil$, $s'_1 = \tau.\alpha.nil$, $s_2 = \beta.nil$ and

$$s \;=\; \left[\tfrac{1}{2}\right] s_1 \;\oplus\; \left[\tfrac{1}{2}\right] s_2,\; s' \;=\; \left[\tfrac{1}{2}\right] s_1' \;\oplus\; \left[\tfrac{1}{2}\right] s_2.$$

Then, $s_1 \approx^{decl} s_1'$ but $s \not\approx^{decl} s'$ as s reachs via internal actions the weak bisimulation equivalence class C of $s_1 = \alpha.nil$ with probability $1/2$ while s' cannot move to a state that is weakly bisimilar to s_1. Formally, we have

$$Prob^{decl}(s, \tau^*, C) \;=\; \tfrac{1}{2} \;>\; 0 \;=\; Prob^{decl}(s', \tau^*, C)$$

where C is the weak bisimulation equivalence class of s_1 and *decl* an arbitrary declaration.

7.5 Proofs

In this section we give the proofs of the main results of that chapter. In what follows, we fix a finite action-labelled fully probabilistic system $(S, Act, \mathbf{P})$. We use the following notations: If R is an equivalence relation on S and Ω a regular expression such that $Prob(s, \Omega, C) = Prob(s', \Omega, C)$ for all $(s, s') \in R$ and $C \in S/R$ then we define for all $A \in S/R$: $Prob(A, \Omega, C) = Prob(s, \Omega, C)$ where $s \in A$. We simply write $[s]$ to denote the weak bisimulation equivalence class of s (i.e. $[s] = [s]_{\approx}$).

7.5.1 Weak and branching bisimulation equivalence

In this section we give the proof of Theorem 7.1.10 (page 163) which states that $\approx = \approx_{br}$ and the proof of Theorem 7.1.4 (page 162).

Definition 7.5.1 [Completeness of a weak bisimulation] *Let R be a weak bisimulation. R is called* complete *iff*

- *Whenever $s \in S$, $C \in S/R$ and $Prob(s, \tau^*, C) = 1$ then $s \in C$.*
- *If $Div \neq \emptyset$ then $Div \in S/R$.*

Note that, if R is a complete weak bisimulation and $A \in S/R$, $A \neq Div$, then $A \cap Div = \emptyset$ (in particular, A does not contain terminal states) and there is a state $s \in A$ with $\mathbf{P}(s, \tau, A) < 1$. Thus, $A \cap S_{\mathcal{X}} \neq \emptyset$ where $\mathcal{X} = S/R$ is the induced partition.[12]

Lemma 7.5.2 *$s \approx s'$ iff $(s, s') \in R$ for some complete weak bisimulation R.*

Proof: It suffices to show that each weak bisimulation is contained in some complete weak bisimulation. For R to be a weak bisimulation, we define $\mathcal{J}(R)$ to be the smallest equivalence relation on S which contains R and such that:

- If $Prob(s, \tau^*, [s']_R) = 1$ then $(s, s') \in \mathcal{J}(R)$
- If $Div \neq \emptyset$ then $(s, s') \in \mathcal{J}(R)$ for all s, $s' \in Div$.

Let $R_0 = R$, $R_{i+1} = \mathcal{J}(R_i)$ and $R' = \bigcup_i R_i$. It is easy to see that R' is a complete weak bisimulation. ■

[12] Here, $S_{\mathcal{X}}$ is as in Notation 7.2.2, page 165.

Notation 7.5.3 [The matrices $\mathbf{A}_R$ and $\mathbf{A}_R^0$] *For R to be a complete weak bisimulation, we define matrices $\mathbf{A}_R$ and $\mathbf{A}_R^0$ as follows: Let $A_1, \ldots, A_k$ be an enumeration of those equivalence classes $A_i \in S/R$ which contain some state $s_i \in S \setminus S_{term}$ with $\mathbf{P}(s_i, \tau, A_i) < 1$. (Then, $\{A_1, \ldots, A_k\} = S/R \setminus \{Div\}$.) Let $\mathbf{A}_R$ be the following matrix:*

$$\mathbf{A}_R \;=\; (\; Prob(A_i, \tau^*, A_j)\;)_{1 \leq i,j \leq k}$$

We define $A_0 = Div$. In the case where $A_0 = \emptyset$ we define $Prob(A_0, \tau^*, A_0) = 1$ and

$$Prob(A_0, \tau^*, A_j) = Prob(A_j, \tau^*, A_0) = 0$$

if $j \geq 1$. Let $\mathbf{A}_R^0 \;=\; (\; Prob(A_i, \tau^*, A_j)\;)_{0 \leq i,j \leq k}$.

Independent on whether or not $Div = \emptyset$, the matrix $\mathbf{A}_R^0$ is of the form

$$\mathbf{A}_R^0 \;=\; \left(\begin{array}{c|c} 1 & * \\ \hline 0 & \\ \vdots & \mathbf{A}_R \\ 0 & \end{array}\right).$$

Note that $S_{term} \subseteq Div = A_0$. Hence, $A_i \cap S_{term} = \emptyset$, $i = 1, \ldots, k$.

Lemma 7.5.4 *If R is a complete weak bisimulation then $\mathbf{A}_R$ and $\mathbf{A}_R^0$ are regular. Moreover: For all $l \in \{1, \ldots, k\}$ and all $b_0, \ldots, b_{l-1}, b_{l+1}, \ldots, b_k \in [0, 1]$, the equation system*

$$x_l = 0, \quad \sum_{i=0}^{k} x_i \cdot Prob(A_i, \tau^*, A_j) \;=\; b_j, \quad j = 0, \ldots, k, j \neq l$$

has at most one solution.

Proof: Let $A_0, \ldots, A_k$ be as in Notation 7.5.3 (page 178). For each $h \in \{1, \ldots, k\}$ we fix some state $s_h \in A_h$ with $\mathbf{P}(s_h, \tau, A_h) < 1$. Let

$$\mathbf{C} \;=\; (\; \mathbf{P}(s_h, \tau, A_i)\;)_{1 \leq h,i \leq k} \quad \text{and} \quad \mathbf{E} \;=\; \left(\begin{array}{ccccc} 1 - e_1 & 0 & 0 & \ldots & 0 \\ 0 & 1 - e_2 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \ldots & 1 - e_k \end{array}\right)$$

where

$$e_j \;=\; \sum_{i=1}^{k} \mathbf{P}(s_j, \tau, A_i) \cdot Prob(A_i, \tau^*, A_j).$$

We show that $\mathbf{C} \cdot \mathbf{A}_R \;+\; \mathbf{E} \;=\; \mathbf{A}_R$. Let $d_{h,j}$ be the element of $\mathbf{C} \cdot \mathbf{A}_R \;+\; \mathbf{E}$ in the h-th row and j-th column.

- For $j = 1, \ldots, k$ we have:

$$d_{j,j} \;=\; \sum_{i=1}^{k} \mathbf{P}(s_j, \tau, A_i) \cdot Prob(A_i, \tau^*, A_j) \;+\; 1 - e_j \;=\; 1 \;=\; Prob(A_j, \tau^*, A_j).$$

- For $h,\ j = 1, \ldots, k$ and $h \neq j$:

$$Prob(A_h, \tau^*, A_j) \ = \ Prob(s_h, \tau^*, A_j) \ = \ \sum_{i=1}^{k} \mathbf{P}(s_h, \tau, A_i) \cdot Prob(A_i, \tau^*, A_j) \ = \ d_{h,j}.$$

  Note that $Prob(A_0, \tau^*, A_j) = 0$ for all $j \geq 1$.

This yields $\mathbf{C} \cdot \mathbf{A}_R + \mathbf{E} \ = \ \mathbf{A}_R$. Thus, $\mathbf{E} \ = \ (\mathbf{I} - \mathbf{C}) \cdot \mathbf{A}_R$ where $\mathbf{I}$ denotes the $k \times k$-identity matrix. Next we show that $e_j > 0,\ j = 1, \ldots, k$.

- If $Prob(A_{i_0}, \tau^*, A_j) \neq 0$ for some $i_0 \in \{1, \ldots, k\} \setminus \{j\}$ with $\mathbf{P}(s_j, \tau, A_{i_0}) > 0$ then

$$e_j \ \leq \ \sum_{\substack{i=1 \\ i \neq i_0}}^{k} \mathbf{P}(s_j, \tau, A_i) \ + \ \mathbf{P}(s_j, \tau, A_{i_0}) \cdot Prob(A_{i_0}, \tau^*, A_j) \ < \ \mathbf{P}(s_j, \tau) \ \leq \ 1$$

  since $Prob(A_{i_0}, \tau^*, A_j) < 1$ because R is complete.
- If $Prob(A_i, \tau^*, A_j) = 0$ for all $i \in \{1, \ldots, k\} \setminus \{j\}$ with $\mathbf{P}(s_j, \tau, A_i) > 0$ then

$$e_j \ = \ \mathbf{P}(s_j, \tau, A_j) \cdot Prob(A_j, \tau^*, A_j) \ = \ \mathbf{P}(s_j, \tau, A_j) < 1.$$

Thus, in both cases, $e_j < 1$. Hence, $\mathbf{E}$ is regular which yields the regularity of $\mathbf{A}_R$. It is clear that the regularity of $\mathbf{A}_R$ implies the regularity of the "full" matrix $\mathbf{A}_R^0$. Note that $\mathbf{A}_R^0$ (and thus the inverse matrix $(\mathbf{A}_R{}^0)^{-1}$ of $\mathbf{A}_R^0$) are of the form:

$$\mathbf{A}_R^0 \ = \ \left(\begin{array}{c|c} 1 & * \\ \hline 0 & \\ \vdots & \mathbf{A}_R \\ 0 & \end{array}\right) \qquad (\mathbf{A}_R{}^0)^{-1} \ = \ \left(\begin{array}{c|c} 1 & * \\ \hline 0 & \\ \vdots & \mathbf{A}_R^{-1} \\ 0 & \end{array}\right)$$

Moreover, $\mathbf{E} \cdot \mathbf{A}_R^{-1} = \mathbf{I} - \mathbf{C}$. Thus, $(1 - e_j) \cdot a_{j,j} \ = \ 1 - \mathbf{P}(s_j, \tau, A_j) > 0$ where $(\mathbf{A}_R{}^0)^{-1} = (a_{i,j})_{0 \leq i,j \leq 1}$. Hence,

(*) $a_{j,j} > 0,\ j = 1, \ldots, k.$

Next we show that the equation system of above has at most one solution. Let

$$L = \{(b_0, \ldots, b_{l-1}, t, b_{l+1}, \ldots, b_k) : t \in I\!R\}$$

where $I\!R$ denotes the set of real numbers. Let $H = \{(x_0, \ldots, x_k) \in I\!R^{k+1} : x_l = 0\}$ and $L' = \{\mathbf{y} \cdot (\mathbf{A}_R{}^0)^{-1} : \mathbf{y} \in L\}$. Then, $H \cap L'$ is the set of solution of the equation system under consideration. We show that either $H \cap L' = \emptyset$ or $H \cap L'$ consists of a single point. First we observe that H and L (and thus L' and $H \cap L'$) are affine spaces with $dim(H) = k$ and $dim(L) = dim(L') = 1$ where $dim(X)$ denotes the dimension of X. Hence, if $H \cap L' \neq \emptyset$ then

- either $dim(H \cap L') = 0$ (iff $H \cap L'$ consists of a single point)
- or $dim(H \cap L') = 1$ (iff $L' \subseteq H$).

Therefore, it suffices to show that $L' \not\subseteq H$. We suppose that $L' \subseteq H$. Then, there are real vectors $\mathbf{a}$, $\mathbf{c}$ such that $L' = \{\mathbf{a} + t \cdot \mathbf{c} : t \in I\!R\}$ where $\mathbf{a} = (a_0, \ldots, a_k)$ and $\mathbf{c} = (c_0, \ldots, c_k)$ with $a_l = c_l = 0$ and $\mathbf{c} \neq 0$. By definition of L' we have $L = \{\mathbf{x} \cdot \mathbf{A}_R^0 : \mathbf{x} \in L'\}$. Hence,

$$b_j \;=\; \sum_{i=0}^{k} a_i \cdot \mathit{Prob}(A_i, \tau^*, A_j) \;+\; t \cdot \sum_{i=0}^{k} c_i \cdot \mathit{Prob}(A_i, \tau^*, A_j)$$

for all $j = 0, \ldots, k$, $j \neq l$ and $t \in I\!R$. Thus,

$$\sum_{i=0}^{k} c_i \cdot \mathit{Prob}(A_i, \tau^*, A_j) = 0 \;\text{ if } j \neq l.$$

Hence,

$$c \;\stackrel{\text{def}}{=}\; \sum_{i=0}^{k} c_i \cdot \mathit{Prob}(A_i, \tau^*, A_l) \;\neq\; 0$$

(since, otherwise the rows of $\mathbf{A}_R^0$ would be linear dependent in contradiction to the regularity of $\mathbf{A}_R^0$). W.l.o.g. $c = 1$. Then, $\mathbf{c}$ is the l-th row of $(\mathbf{A}_R{}^0)^{-1}$. In particular, $0 = c_l = a_{l,l}$. But this contradicts the constraint $a_{l,l} > 0$ from (*). ■

We show that $\approx$ coincides with another kind of bisimulation equivalence that we call right-branching bisimulation.

Notation 7.5.5 [Right-branching bisimulation] *A* right-branching bisimulation *is an equivalence relation R on S such that $\mathit{Prob}(s, \tau^*\widehat{a}, C) = \mathit{Prob}(s', \tau^*\widehat{a}, C)$ for all $(s, s') \in R$, $a \in \mathit{Act}$ and all equivalence classes $C \in S/R$. $s \approx_{rbr} s'$ iff $(s, s') \in R$ for some right-branching bisimulation R.*

Lemma 7.5.6 *$s \approx_{rbr} s'$ implies $s \approx s'$. More precisely: Each right-branching bisimulation is a weak bisimulation.*

Proof: Let R be a right-branching bisimulation. We show that R is a weak bisimulation. Let $\alpha \in \mathit{Act} \setminus \{\tau\}$, $s \in S$ and $C \in S/R$. Then, for all $B \in S/R$ and $s \in B$:

$$\begin{aligned} \mathit{Prob}(s, \tau^*\alpha\tau^*, C) \;&=\; \sum_{t \in S} \mathit{Prob}(s, \tau^*\alpha, t) \cdot \mathit{Prob}(t, \tau^*, C) \\ &= \sum_{A \in S/R} \mathit{Prob}(B, \tau^*\alpha, A) \cdot \mathit{Prob}(A, \tau^*, C). \end{aligned}$$

Hence, if $(s, s') \in R$ then $\mathit{Prob}(s, \tau^*\alpha\tau^*, C) \;=\; \mathit{Prob}(s', \tau^*\alpha\tau^*, C)$ for all $C \in S/R$ and $\alpha \in \mathit{Act} \setminus \{\tau\}$. ■

Lemma 7.5.7 *$s \approx s'$ implies $s \approx_{rbr} s'$. More precisely: Each complete weak bisimulation is a right-branching bisimulation.*

Proof: Let R be a complete weak bisimulation. We fix some $\alpha \in \mathit{Act} \setminus \{\tau\}$ and show that $\mathit{Prob}(s, \tau^*\alpha, A) \;=\; \mathit{Prob}(s', \tau^*\alpha, A)$ for all $s \approx s'$ and all $A \in S/R$. (Ranging over all α, we obtain that R is a right-branching bisimulation. Thus, $\approx \subseteq \approx_{rbr}$.)

By the regularity of $\mathbf{A}^0_R$ (Lemma 7.5.4, page 178): Whenever we fix a real vector $\mathbf{a}$ (of length $k+1$) then the linear equation system $\mathbf{x} \cdot \mathbf{A}^0_R = \mathbf{a}$ has a unique solution. For $s \in S$ and $j = 0, 1, \ldots, k$ we have:

$$\begin{aligned} Prob(s, \tau^*\alpha\tau^*, A_j) &= \sum_{t \in S} Prob(s, \tau^*\alpha, t) \cdot Prob(t, \tau^*, A_j) \\ &= \sum_{i=0}^{k} Prob(s, \tau^*\alpha, A_i) \cdot Prob(A_i, \tau^*, A_j). \end{aligned}$$

Thus, for fixed l: For all states $s \in A_l$, the vector $(Prob(s, \tau^*\alpha, A_i))_{0 \leq i \leq k}$ is a solution of the linear equation system $\mathbf{x} \cdot \mathbf{A}^0_R = \mathbf{a}$ where $\mathbf{a} = (a_j)_{0 \leq j \leq k}$ and $a_j = Prob(A_l, \tau^*\alpha\tau^*, A_j)$. By the regularity of $\mathbf{A}^0_R$: If $(s, s') \in R$ (i.e. s, $s' \in A_l$ for some l) then $Prob(s, \tau^*\alpha, A_i) = Prob(s', \tau^*\alpha, A_i)$, $i = 0, \ldots, k$. ■

Lemma 7.5.8 *$s \approx_{br} s'$ implies $s \approx s'$. More precisely: Each branching bisimulation is a weak bisimulation.*

Proof: Let R be a branching bisimulation. It suffices to show that R is a right-branching bisimulation (Lemma 7.5.6, page 180). For $r \geq 1$ and $A, C \in S/R$, $A \neq C$, let Γ_r be the set of tuples $(C_0, \ldots, C_r)$ such that

- $C_i \in S/R$, $i = 0, 1, \ldots, r$,
- $C_0 = A$, $C_r = C$,
- $C_i \neq C_{i+1}$, $i = 0, \ldots, r-1$.

Then, for all $s \in A$:

$$Prob(s, \tau^*, C) = \sum_{r=1}^{\infty} \sum_{(C_1, \ldots, C_r) \in \Gamma_r} \prod_{i=0}^{r-1} Prob_R(C_i, \tau^*, C_{i+1})$$

Hence, $Prob(s, \tau^*, C) = Prob(s', \tau^*, C)$ for all s, $s' \in A$. Similarly,

$$Prob(s, \tau^*\alpha, C) = Prob(s', \tau^*\alpha, C)$$

for all s, $s' \in A$, $\alpha \in Act \setminus \{\tau\}$ and $C \in S/R$. ■

Lemma 7.5.9 (cf. Lemma 7.2.4, page 165) *Let R be an equivalence relation on S. Then, R is a branching bisimulation if and only if for all $C \in S/R$, $a \in Act$ and $(s, s') \in R$:*

(1) *If $\mathbf{P}(s, \tau, [s]_R)$, $\mathbf{P}(s', \tau, [s']_R) < 1$ and $(a, C) \neq (\tau, [s]_R)$ then*

$$\frac{\mathbf{P}(s, a, C)}{1 - \mathbf{P}(s, \tau, [s]_R)} = \frac{\mathbf{P}(s', a, C)}{1 - \mathbf{P}(s', \tau, [s']_R)}.$$

(2) *If $\mathbf{P}(s, \tau, [s]_R) = 1$ then*

- *either $\mathbf{P}(t, \tau, [t]_R) = 1$ for all $t \in [s]_R$*
- *or there exists a finite path σ starting in s with $\sigma(i) \in [s]_R$, $i = 1, \ldots, |\sigma|$ and $\mathbf{P}(last(\sigma), \tau, [s]_R) < 1$.*

In this case: If $s \in S$ *with* $\mathbf{P}(s,\tau,[s]_R) < 1$ *then*

$$Prob_R(s,\tau^*\widehat{a},C) \;=\; \frac{\mathbf{P}(s,a,C)}{1-\mathbf{P}(s,\tau,C)}$$

for all $a \in Act$, $C \in S/R$ *with* $(a,C) \neq (\tau,[s]_R)$.

Proof: Let $T = S_{term} \cup \{t \in S \setminus S_{term} : \mathbf{P}(t,\tau,[t]_R) = 1\}$ (i.e. $T = S \setminus S_{\mathcal{X}}$ where $\mathcal{X} = S/R$).

"if": Let $A \in S/R$ with $A \subseteq T$. Then, for all $s \in A$:

$$Prob_R(s,\tau^*\widehat{a},C) \;=\; \begin{cases} 1 & : \text{ if } a=\tau \text{ and } C=A \\ 0 & : \text{ otherwise} \end{cases}$$

Then, $Prob_R(t,\tau^*\widehat{a},C) = Prob_R(t',\tau^*\widehat{a},C)$ for all t, $t' \in A$. Let A, $C \in S/R$ such that $A \not\subseteq T$. Then, the matrix $\mathbf{I}-\mathbf{C}_A$ is regular where $\mathbf{I}$ is the $|A| \times |A|$-identity matrix and $\mathbf{C}_A = (\mathbf{P}(s,\tau,t))_{s,t\in A}$. This can be seen as follows: If $(\mathbf{I}-\mathbf{C}_A)\cdot\mathbf{x} = 0$, $\mathbf{x} = (x_s)_{s\in A}$, then $x_s = \sum_{t\in A}\mathbf{P}(s,\tau,t)\cdot x_t$ for all $s \in A$. If we suppose that $\mathbf{x} \neq 0$ then we may assume w.l.o.g. that $x_s > 0$ for some $s \in A$ (otherwise we deal with $-\mathbf{x}$ rather than $\mathbf{x}$). Let W be the set of states $s \in A$ where x_s is maximal. Then, for all $s \in W$, $\mathbf{P}(s,\tau,A) = 1$ and $x_s = x_t$ if $\mathbf{P}(s,\tau,t) > 0$. Thus, $W \subseteq T$ and, if $s \in W$ then $\mathbf{P}(s,\tau,t) > 0$ implies $t \in W$. Hence, there is no finite path σ starting in s with $\sigma(i) \in A$, $i = 0,1,\ldots,|\sigma|-1$, and $\mathbf{P}(last(\sigma),\tau,A) < 1$. This contradicts (2).

Let $a \in Act$ such that $(a,C) \neq (\tau,A)$. Then,

$$Prob_R(s,\tau^*\widehat{a},C) = \sum_{t\in A}\mathbf{P}(s,\tau,t)\cdot Prob_R(t,\tau^*\widehat{a},C) \;+\; \mathbf{P}(s,a,C).$$

Hence, the vector $(Prob_R(s,\tau^*\widehat{a},C))_{s\in A}$ solves the equation system $(\mathbf{I}-\mathbf{C}_A)\cdot\mathbf{x} = \mathbf{b}$ where $\mathbf{b} = (b_s)_{s\in A}$ and $b_s = \mathbf{P}(s,a,C)$. On the other hand, for all $s \in A$:

$$(*)\quad \mathbf{P}(s,a,C) \;=\; = \sum_{t\in A}\mathbf{P}(s,\tau,t)\cdot\mathbf{P}(s,a,C) \;+\; \mathbf{P}(s,a,C)(1-\mathbf{P}(s,\tau,A)).$$

Let $x = \mathbf{P}(s,a,C)/(1-\mathbf{P}(s,\tau,A))$ where $s \in A \setminus T$. (*) yields:

$$x \;=\; \sum_{t\in A}\mathbf{P}(s,\tau,t)\cdot x \;+\; \mathbf{P}(s,a,C) \quad \text{for all } s \in A.$$

(Note that $\mathbf{P}(s,a,C) = 0$ and $\mathbf{P}(s,\tau,A) = 1$ for $s \in A \cap T$.) Hence, the vector $\mathbf{x} = (x_s)_{s\in A}$ where $x_s = x$ for all $s \in A$ solves the equation system $(\mathbf{I}-\mathbf{C}_A)\cdot\mathbf{x} = \mathbf{b}$. By the regularity of $\mathbf{I}-\mathbf{C}_A$ we get $Prob_R(s,\tau^*\widehat{a},C) \;=\; x$ for all $s \in A$. This yields that R is a branching bisimulation and

$$Prob_R(s,\tau^*\widehat{a},C) \;=\; x \;=\; \frac{\mathbf{P}(s,a,C)}{1-\mathbf{P}(s,\tau,A)}$$

for all $s \in A \setminus T$.

"only if": Let R be a branching bisimulation, A, $C \in S/R$ and $a \in Act$ such that $(a,C) \neq (\tau,A)$. Let

$$x_{a,C} \;=\; Prob_R(A,\tau^*\widehat{a},C).$$

Then, $x_{a,C} = \mathbf{P}(s,\tau,A)\cdot x_{a,C} + \mathbf{P}(s,a,C)$ for all $s \in A$. Hence, if $s \in A \cap T$ then

$$x_{a,C} = \frac{\mathbf{P}(s,a,C)}{1-\mathbf{P}(s,\tau,A)}.$$

If $s \in A \cap T$ and $A \not\subseteq T$ then $x_{a,C} \neq 0$ for some pair (a,C) as above. Thus,

$$Prob_R(s,\tau^*\widehat{a},C) = x_{a,C} > 0.$$

Then, there exists a finite path σ' starting in s of length $r \geq 1$ with $trace(\sigma') \in \tau^*a$, $\sigma'(i) \in A$, $i = 0,1,\ldots,r-1$ and $last(\sigma') \in C$. Let σ be the $(r-1)$-th prefix of σ'. Then, $\sigma(i) \in A$, $i = 1,\ldots,r-1$ and $\mathbf{P}(last(\sigma),\tau,A) < 1$ (since $\mathbf{P}(last(\sigma),a,C) > 0$). ■

Proposition 7.5.10 *$s \approx s'$ iff $s \approx_{rbr} s'$*

Proof: follows by Lemma 7.5.6 (page 180) and Lemma 7.5.7 (page 180). ■

Notation 7.5.11 [The conditional probabilities $\mathbf{P}_R(\cdot)$] *Let R be an equivalence relation on S.*

$$\mathbf{P}_R : S \times Act \times S \to [0,1]$$

is given by: $\mathbf{P}_R(s,a,t) = 0$ if $\mathbf{P}(s,\tau,[s]_R) = 1$ or $s \in S_{term}$. For $s \in S \setminus S_{term}$ with $\mathbf{P}(s,\tau,[s]_R) < 1$,

$$\mathbf{P}_R(s,a,t) = \frac{\mathbf{P}(s,a,t)}{1-\mathbf{P}(s,\tau,[s]_R)} \quad \text{if } a \neq \tau \text{ or } t \notin [s]_R.$$

and $\mathbf{P}_R(s,\tau,t) = 0$ if $t \in [s]_R$. For $C \subseteq S$, $a \in Act$ and $s \in S$, let

$$\mathbf{P}_R(s,a,C) = \sum_{t\in C} \mathbf{P}_R(s,a,t).$$

Clearly, $\mathbf{P}_R(\cdot) = \mathbf{P}_{\mathcal{X}}(\cdot)$ (defined as in Notation 7.2.3, page 165) for the induced partition $\mathcal{X} = S/R$.

Lemma 7.5.12 *$s \approx s'$ implies $s \approx_{br} s'$. More precisely: Each complete weak bisimulation is a branching bisimulation.*

Proof: Let R be a complete weak bisimulation. It suffices to show that R fulfills the conditions (1), (2) of Lemma 7.5.9 (page 181). Let $T = S_{term} \cup \{t \in S : \mathbf{P}(t,\tau,[t]_R) = 1\}$.

Condition (2): Let $A \in S/R$, $A \neq Div$. There exists $a \in Act$ and $C \in S/R$ with $(\tau,A) \neq (a,C)$ such that $Prob(A,\tau^*\widehat{a},C) \neq 0$. Hence, for each $s \in T$ there is a finite path σ with $trace(\sigma) \in \tau^*\widehat{a}$ and $last(\sigma) \in C$. Let i be the smallest index such that $\sigma(i) \notin T$. (Such an index i exists by definition of T.) Then, $i \geq 1$ and $\sigma(i) \in A \setminus T$. (Note that $\sigma(i-1) \in T$ implies $\sigma(i) \in A$.)

Condition (1): Let $A_0,\ldots,A_k$ be as in Notation 7.5.3 (page 178). Let $C \in S/R$, $A_j \neq C$, $s \in A_j$. Then,

$$Prob(s,\tau^*,C) = \sum_{i=0}^{k} \mathbf{P}(s,\tau,A_i)\cdot Prob(A_i,\tau^*,C).$$

Note that $Prob(C, \tau^*, C) = 1$. Now we suppose that $s \in A_j \setminus T$. Then,

$$\frac{Prob(s,\tau^*,C)}{1-\mathbf{P}(s,\tau,A_j)} \;=\; \sum_{i=0}^{k} \frac{\mathbf{P}(s,\tau,A_i)}{1-\mathbf{P}(s,\tau,A_j)} \cdot Prob(A_i,\tau^*,C)$$

$$= \sum_{\substack{i=0\\ i\neq j}}^{k} \frac{\mathbf{P}(s,\tau,A_i)}{1-\mathbf{P}(s,\tau,A_j)} \cdot Prob(A_i,\tau^*,C) \;+\; \frac{\mathbf{P}(s,\tau,A_j)}{1-\mathbf{P}(s,\tau,A_j)} \cdot Prob(A_j,\tau^*,C)$$

$$= \sum_{\substack{i=0\\ i\neq j}}^{k} \mathbf{P}_R(s,\tau,A_i) \cdot Prob(A_i,\tau^*,C) \;+\; \frac{\mathbf{P}(s,\tau,A_j)}{1-\mathbf{P}(s,\tau,A_j)} \cdot Prob(s,\tau^*,C)$$

Here, we use the fact that $Prob(s, \tau^*, C) = Prob(A_j, \tau^*, C)$. We obtain:

$$Prob(s,\tau^*,C) \;=\; Prob(s,\tau^*,C) \cdot \left(\frac{1}{1-\mathbf{P}(s,\tau,A_j)} - \frac{\mathbf{P}(s,\tau,A_j)}{1-\mathbf{P}(s,\tau,A_j)} \right)$$

$$= \sum_{\substack{i=0\\ i\neq j}}^{k} \mathbf{P}_R(s,\tau,A_i) \cdot Prob(A_i,\tau^*,C)$$

Thus, for each $s \in A_j \setminus T$, the vector $(\mathbf{P}_R(s,\tau,A_i))_{0\leq i\leq k}$ solves the equation system

$$x_j = 0, \quad \sum_{i=0}^{k} x_i \cdot Prob(A_i,\tau^*,C) \;=\; Prob(A_j,\tau^*,C).$$

Lemma 7.5.4 (page 178) yields $\mathbf{P}_R(s,\tau,C) = \mathbf{P}_R(s',\tau,C)$ for all $s, s' \in A_j$ and $C \in S/R$, $C \neq A_j$. Let

$$\mathbf{P}_R(A_j,\tau,C) = \mathbf{P}_R(s,\tau,C) \;\text{ where } s \in A_j \cap T.$$

For all $\alpha \in Act \setminus \{\tau\}$ and $s \in A_j$:

$$Prob(s,\tau^*\alpha,C) \;=\; \sum_{i=0}^{k} \mathbf{P}(s,\tau,A_i) \cdot Prob(A_i,\tau^*\alpha,C) \;+\; \mathbf{P}(s,\alpha,C)$$

As before, we obtain for $s \in A_j \setminus T$:

$$Prob(s,\tau^*\alpha,C) \;=\; \sum_{\substack{i=0\\ i\neq j}}^{k} \mathbf{P}_R(s,\tau,A_i) \cdot Prob(A_i,\tau^*\alpha,C) \;+\; \mathbf{P}_R(s,\alpha,C).$$

Then, for all $s \in A_j \setminus T$, $\alpha \in Act$ and $C \in S/R$:

$$Prob(A_j,\tau^*\alpha,C) \;=\; \sum_{\substack{i=0\\ i\neq j}}^{k} \mathbf{P}_R(A_j,\tau,A_i) \cdot Prob(A_i,\tau^*\alpha,C) \;+\; \mathbf{P}_R(s,\alpha,C)$$

We obtain $\mathbf{P}_R(s,\alpha,C) = \mathbf{P}_R(s',\alpha,C)$ for all $s,\ s' \in A_j \setminus T$. ■

Corollary 7.5.13 (cf. Theorem 7.1.10, page 163) *$s \approx s'$ iff $s \approx_{br} s'$.*

Lemma 7.5.14 *Let R_1, R_2 be branching bisimulations. Then, $R = (R_1 \cup R_2)^*$ is a branching bisimulation.*

Proof: We show that R fulfills the conditions (1) and (2) of Lemma 7.5.9 (page 181). First we observe that for $j \in \{1,2\}$, each equivalence class $C' \in S/R$ can be written as $C' = C_0 \cup \ldots \cup C_r$ where $C_i \in S/R_j$.

Condition (1): Let $B, C_0, \ldots, C_r \in S/R_j$ such that $C_i \cap C_h = \emptyset$ if $i \neq h$. Let $s, s' \in C_0$ with $\mathbf{P}(s,\tau,C_0), \mathbf{P}(s',\tau,C_0) < 1$ and $a \in Act$ with $a \neq \tau$ if $C_0 = B$ and $C' = C_0 \cup \ldots \cup C_r$. Then:

$$\begin{aligned}
\mathbf{P}(s,a,B) \cdot (1 - \mathbf{P}(s',\tau,C')) &= \mathbf{P}(s,a,B) \cdot \left(1 - \sum_{i=0}^{r} \mathbf{P}(s',\tau,C_i)\right) \\
&= \mathbf{P}(s,a,B) \cdot \left(1 - \mathbf{P}(s',\tau,C_0) - \sum_{i=1}^{r} \frac{\mathbf{P}(s,\tau,C_i)}{1-\mathbf{P}(s,\tau,C_0)} \cdot (1 - \mathbf{P}(s',\tau,C_0))\right) \\
&= (1 - \mathbf{P}(s',\tau,C_0)) \cdot \mathbf{P}(s,a,B) \cdot \left(1 - \sum_{i=1}^{r} \frac{\mathbf{P}(s,\tau,C_i)}{1-\mathbf{P}(s,\tau,C_0)}\right) \\
&= \mathbf{P}(s',a,B) \cdot (1 - \mathbf{P}(s,\tau,C_0)) \cdot \left(1 - \sum_{i=1}^{r} \frac{\mathbf{P}(s,\tau,C_i)}{1-\mathbf{P}(s,\tau,C_0)}\right) \\
&= \mathbf{P}(s',a,B) \cdot \left(1 - \sum_{i=0}^{r} \mathbf{P}(s,\tau,C_i)\right) = \mathbf{P}(s',a,B) \cdot (1 - \mathbf{P}(s,\tau,C'))
\end{aligned}$$

Now we assume that $C' \in S/R$. Hence, for all $B' \in S/R$, $a \in Act$ such that $a \neq \tau$ if $C_0 \subseteq B'$: If $\mathbf{P}(s,\tau,C'), \mathbf{P}(s',\tau,C') < 1$. then:

$$\frac{\mathbf{P}(s,a,B')}{1-\mathbf{P}(s,\tau,C')} = \sum_{i=0}^{r} \frac{\mathbf{P}(s,a,B_i)}{1-\mathbf{P}(s,\tau,C')} = \sum_{i=0}^{r} \frac{\mathbf{P}(s',a,B_i)}{1-\mathbf{P}(s',\tau,C')} = \frac{\mathbf{P}(s',a,B')}{1-\mathbf{P}(s',\tau,C')}$$

if $B' = B_0 \cup \ldots \cup B_m$ where $B_i \in S/R_j$ and $B_i \cap B_h = \emptyset$ if $i \neq h$.

Condition (2): Let $\mathbf{P}(s,\tau,[s]_R) = 1$. We may assume that $\mathbf{P}(s,\tau,[s]_{R_j}) < 1$, $j = 1,2$, and that $\mathbf{P}(t,\tau,C') < 1$ for some $t \in C'$ where $C' = [s]_R$. (In the case where $\mathbf{P}(s,\tau,[s]_{R_j}) = 1$ for some j we apply Lemma 7.5.9 (page 181) to R_j and obtain the claim.) There is some $a \in Act$, $B' \in S/R$ with $B' \neq C'$ if $a = \tau$ and $\mathbf{P}(t,a,B') > 0$. By definition of R there is a sequence $s = s_0, s_1, \ldots, s_l = t$ with $(s_i, s_{i+1}) \in R_1 \cup R_2$, $i = 1, \ldots, l$. It can easily shown by induction on i that $Prob(s_i, \tau^* \widehat{a}, B') > 0$, $i = 0, \ldots, l$, Hence, there is a finite path σ with $first(\sigma) = s$, $\sigma(i) \in C'$, $i = 0, 1, \ldots, |\sigma|$ and $\mathbf{P}(last(\sigma),\tau,C') < 1$. ■

Lemma 7.5.15 $\approx_{br}$ *is a branching bisimulation.*

Proof: Let $R_1, \ldots, R_r$ be an enumeration of all branching bisimulations (on the fixed fully probabilistic system $(S, Act, \mathbf{P})$). Let $R = (\bigcup_i R_i)^*$. By induction on r and using Lemma 7.5.14 (page 185), it can be shown that R is a branching bisimulation. Thus, $R \subseteq \approx_{br}$. On the other hand, $\approx_{br} = \bigcup_i R_i$ (by definition of $\approx_{br}$). Hence, $\approx_{br} \subseteq R$. Thus, $\approx_{br} = R$ is a branching bisimulation. ■

Lemma 7.5.16 $\approx$ *is a weak bisimulation.*

Proof: Lemma 7.5.8 (page 181), Lemma 7.5.15 (page 185) and Corollary 7.5.13 (page 184) yield that $\approx = \approx_{br}$ is a weak bisimulation. ■

Theorem 7.5.17 (cf. Theorem 7.1.4, page 162) *If $s \approx s'$. then, for all $C \in S/\approx$, $k \geq 1$ and $\alpha_1, \ldots, \alpha_k \in Act \setminus \{\tau\}$:*

(a) $Prob(s, \tau^*\alpha_1\tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k, C) = Prob(s', \tau^*\alpha_1\tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k, C)$

(b) $Prob(s, \tau^*\alpha_1\tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k\tau^*, C) = Prob(s', \tau^*\alpha_1\tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k\tau^*, C)$

Proof: We prove part (a) by induction on k. In the basis of induction ($k = 1$) we have to show that $Prob(s, \tau^*\alpha, C) = Prob(s', \tau^*\alpha, C)$ for all visible actions α and all weak bisimulation equivalence classes C. This follows immediately by Proposition 7.5.10 (page 183) and Lemma 7.5.16 (page 185). In the induction step $k - 1 \Longrightarrow k$ we assume that $k \geq 2$, $\alpha_1, \ldots, \alpha_k \in Act \setminus \{\tau\}$ and $\Omega = \tau^*\alpha_2\tau^* \ldots \tau^*\alpha_k$. Then,

- $Prob(t, \Omega, C) = Prob(t', \Omega, C)$ for all $t \approx t'$ and $C \in S/\approx$
- $Prob(s, \tau^*\alpha_1, A) = Prob(s', \tau^*\alpha_1, A)$ for all $A \in S/\approx$

(induction hypothesis). Thus:

$$Prob(s, \tau^*\alpha_1\Omega, C) = \sum_{A \in S/\approx} Prob(s, \tau^*\alpha_1, A) \cdot Prob(A, \Omega, C)$$

$$= \sum_{A \in S/\approx} Prob(s', \tau^*\alpha_1, A) \cdot Prob(A, \Omega, C) = Prob(s', \tau^*\alpha_1\Omega, C).$$

Here, we use the fact that $Path_{ful}(u, \tau^*\alpha_1\Omega, C)$ can be written as disjoint union of the sets $\Pi_A(u)$, $A \in S/\approx$, where Π_A is the set of fulpaths π such that

- $trace(\pi^{(k)}) \in \tau^*\alpha_1$,
- $\pi(k) \in A$,
- $\pi = \pi^{(k)} \circ \gamma$ where $\gamma \in Path_{ful}(\pi(k), \Omega, C)$

for some $k \geq 0$. Part (b) can be derived from (a):

$$Prob(s, \Omega, C) = \sum_{A \in S/\approx} Prob(s, \tau^*\alpha_1\tau^* \ldots \tau^*\alpha_k, A) \cdot Prob(A, \tau^*, C)$$

$$= \sum_{A \in S/\approx} Prob(s', \tau^*\alpha_1\tau^* \ldots \tau^*\alpha_k, A) \cdot Prob(A, \tau^*, C) = Prob(s', \Omega, C).$$

where $\Omega = \tau^*\alpha_1\tau^* \ldots \tau^*\alpha_k\tau^*$. ■

7.5.2 $\approx$ and the testing equivalences $=_{ste}$ and $\equiv_0$

We complete the proofs of Theorem 7.3.5 (page 173) and Theorem 7.3.6 (page 174) by showing that $\approx$ is finer than the testing equivalences $=_{ste}$ and $\equiv_0$.

Lemma 7.5.18 *If A, $C \in S/\approx$, s, $s' \in A$, $L \subseteq Act \setminus \{\tau\}$ and $\alpha \in L$ then*

$$Q(s, L, \alpha, C) = Q(s', L, \alpha, C).$$

Proof: First we observe that, for all $A, B \in S/\approx = S/\approx_{br}$, $s, s' \in A$ with $\mathbf{P}(s,\tau,A)$, $\mathbf{P}(s',\tau,A) < 1$ and $\alpha \in Act \setminus \{\tau\}$:

- $$\frac{\mathbf{P}(s,\tau,S\setminus A) \;+\; \mathbf{P}(s,L)}{1-\mathbf{P}(s,\tau,A)} \;=\; \frac{\mathbf{P}(s',\tau,S\setminus A) \;+\; \mathbf{P}(s',L)}{1-\mathbf{P}(s',\tau,A)}$$
- $$\frac{\mathbf{P}(s,\alpha,B)}{1-\mathbf{P}(s,\tau,A)} \;=\; \frac{\mathbf{P}(s',\alpha,B)}{1-\mathbf{P}(s',\tau,A)}$$

In particular, $\mathbf{P}(s,\tau,S\setminus A) \;+\; \mathbf{P}(s,L) = 0$ iff $\mathbf{P}(s',\tau,S\setminus A) \;+\; \mathbf{P}(s',L) = 0$. If $A = Div$ then we put $\mathbf{P}'(A,\alpha,B) = 0$ and $r_A = 1$. For $A \in S/\approx$, $A \neq Div$, we define

$$\mathbf{P}'(A,\alpha,B) \;=\; \frac{\mathbf{P}(s,\alpha,B)}{1-\mathbf{P}(s,\tau,A)}, \quad r_A \;=\; \frac{\mathbf{P}(s,\tau,S\setminus A)+\mathbf{P}(s,L)}{1-\mathbf{P}(s,\tau,A)}$$

where $s \in A$ such that $\mathbf{P}(s,\tau,A) < 1$. Let $(q_A)_{A\in S/\approx}$ be the unique solution of the following equation system.

1. $q_A = 0$ if $r_A = 0$ or $Prob(A,\tau^*\alpha,C) = 0$.
2. If $r_A > 0$ and $Prob(A,\tau^*\alpha,C) > 0$ then

$$q_A = \;\frac{1}{r_A}\cdot\left(\mathbf{P}'(A,\alpha,C) \;+\; \sum_{\substack{B\in S/\approx\\ B\neq A}} \mathbf{P}'(A,\tau,B)\cdot q_B\right).$$

The uniqueness of the equation system above is an easy verification. For all $A \in S/\approx$ such that $r_A > 0$ and $Prob(A,\tau^*\alpha,C) > 0$ and $s \in A$ with $\mathbf{P}(s,\tau,A) > 0$ we have:

$$\mathbf{P}(s,\tau,S\setminus A)+\mathbf{P}(s,L) > 0$$

and

$$\begin{aligned}
\left(1-\frac{\mathbf{P}(s,\tau,A)}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)}\right)\cdot q_A \;&=\; \frac{\mathbf{P}(s,\tau,S\setminus A)+\mathbf{P}(s,L)}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)}\cdot q_A\\
&=\; \frac{\mathbf{P}(s,\tau,S\setminus A)+\mathbf{P}(s,L)}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)}\cdot\frac{1}{r_A}\cdot\left(\mathbf{P}'(A,\alpha,C)+\sum_{\substack{B\in S/\approx\\ B\neq A}}\mathbf{P}'(A,\tau,B)\cdot q_B\right)\\
&=\; \frac{1-\mathbf{P}(s,\tau,A)}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)}\cdot\left(\mathbf{P}'(A,\alpha,C)+\sum_{\substack{B\in S/\approx\\ B\neq A}}\mathbf{P}'(A,\tau,B)\cdot q_B\right)\\
&=\; \frac{1}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)}\cdot\left(\mathbf{P}(s,\alpha,C)+\sum_{\substack{B\in S/\approx\\ B\neq A}}\mathbf{P}(A,\tau,B)\cdot q_B\right)\\
&=\; \frac{\mathbf{P}(s,\alpha,C)}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)} \;+\; \sum_{\substack{B\in S/\approx\\ B\neq A}}\frac{\mathbf{P}(s,\tau,B)}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)}\cdot q_B.
\end{aligned}$$

Thus,

$$q_A \;=\; \frac{\mathbf{P}(s,\alpha,C)}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)} \;+\; \sum_{B\in S/\approx} \frac{\mathbf{P}(s,\tau,B)}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)} \cdot q_B.$$

For $A \in S/\approx$ and $s \in A$, let $q_s = q_A$ and $r_s = r_A$. Then, the vector $(q_s)_{s\in S}$ solves the following regular linear equation system. If $Prob(s,\tau^*\alpha,C) = 0$ or $r_s = 0$ then $q_s = 0$. Otherwise,

$$q_s \;=\; \frac{\mathbf{P}(s,\alpha,C)}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)} \;+\; \sum_{u\in S} \frac{\mathbf{P}(s,\tau,u)}{\mathbf{P}(s,\tau)+\mathbf{P}(s,L)} \cdot q_u.$$

It is easy to see that, if $r_s = 0$ then $Q(s,L,\alpha,C) = 0$. Thus, the vector $(Q(s,L,\alpha,C))_{s\in S}$ is also a solution of the equation system above. Hence,

$$q_s = Q(s,L,\alpha,C) \text{ for all } s \in S.$$

We conclude: $Q(s,L,\alpha,C) = q_A = Q(s',L,\alpha,C)$ for all s, $s' \in A$, $A \in S/\approx$. ■

Theorem 7.5.19 (cf. Theorem 7.3.5, page 173) *$\approx$ is finer than $=_{ste}$.*

Proof: As observed in [Chri90b], $s =_{ste} s'$ iff

$$Q(s,L_1\ldots L_k,\alpha_1\ldots\alpha_k) = Q(s',L_1\ldots L_k,\alpha_1\ldots\alpha_k)$$

for all $L_1,\ldots,L_k \in$ *Offerings* and $\alpha_1,\ldots,\alpha_k \in Act\setminus\{\tau\}$. By induction on k and using Lemma 7.5.18 (page 186) we obtain that, if $s \approx s'$ then

$$Q(s,L_1\ldots L_k,\alpha_1\ldots\alpha_k,C) = Q(s',L_1\ldots L_k,\alpha_1\ldots\alpha_k,C)$$

for all $C \in S/\approx$. Summing up over all $C \in S/\approx$ we obtain

$$Q(s,L_1\ldots L_k,\alpha_1\ldots\alpha_k) = Q(s',L_1\ldots L_k,\alpha_1\ldots\alpha_k).$$

Hence, $s =_{ste} s'$. ■

Notation 7.5.20 [The set *$Distr_0(X)$***]** *For X to be a set, let $Distr_0(X)$ be the set consisting of all distributions on X and the function $\rho : X \to [0,1]$ with $\rho(x) = 0$ for all $x \in X$.*

Notation 7.5.21 [Probabilistic traces] *A* probabilistic trace *is a finite sequence*

$$\theta \;=\; \langle\rho_1,\alpha_1\rangle\langle\rho_2,\alpha_2\rangle \;\ldots\langle\rho_k,\alpha_k\rangle$$

over $Distr_0(Act\setminus\{\tau\})\times(Act\setminus\{\tau\})$. ε_{PrTr} denotes the empty probabilistic trace, ProbTraces the collection of all probabilistic traces.

Notation 7.5.22 [The normalizator *$norm(s,\rho)$***]** *For $\rho \in Distr_0(Act\setminus\{\tau\})$ and $s \in S$, the* normalizator *of s and ρ is defined by*

$$norm(s,\rho) \;=\; \sum_{\alpha\in Act\setminus\{\tau\}} \mathbf{P}(s,\alpha)\cdot\rho(\alpha) \;+\; \mathbf{P}(s,\tau).$$

[CSZ92, YCDS94] define the probabilities $N(s,\alpha,\rho,C)$ that from state s the state t is reached via a finite path labelled by $\tau^*\alpha$ given that the environment is enabling actions accordance with ρ. Here, we use a slightly different way to define $N(\cdot)$ (which yields the same values).

Notation 7.5.23 [The values $N(s,\alpha,\rho,C)$] *Let*

$$N : S \times (Act \setminus \{\tau\}) \times Distr_0(Act \setminus \{\tau\}) \times 2^S \to [0,1]$$

be defined as follows. The vector $(N(s,\alpha,\rho,C))_{s\in S}$ is the unique solution of the following linear equation system:

1. *If $Prob(s,\tau^*\alpha,C) = 0$ or $norm(s,\rho) = 0$ then $N(s,\alpha,\rho,C) = 0$.*
2. *If $Prob(s,\tau^*\alpha,C) > 0$ and $norm(s,\rho) > 0$ then*

$$N(s,\alpha,\rho,C) \;=\; \frac{\rho(\alpha)}{norm(s,\rho)} \cdot \mathbf{P}(s,\alpha,C) \;+\; \sum_{t\in S} \frac{\mathbf{P}(s,\tau,t)}{norm(s,\rho)} \cdot N(t,\alpha,\rho,C).$$

Clearly, if $\rho(\alpha) = 0$ for all α then $N(s,\alpha,\rho,C) = 0$ for all states s and $C \subseteq S$.

Notation 7.5.24 [The values $M(s,\theta)$] *Let $M : S \times ProbTraces \to [0,1]$ be given by: $M(s,\varepsilon_{PrTr}) = 1$ and*

$$M(s,\langle\rho,\alpha\rangle\theta) \;=\; \sum_{t\in S} N(s,\alpha,\rho,t) \cdot M(t,\theta)$$

Definition 7.5.25 [The testing equivalence $\equiv_0$, cf. [CSZ92, YCDS94]]

$$s \;\equiv_0\; s' \text{ iff } M(s,\theta) = M(s',\theta) \text{ for all } \theta \in ProbTraces.$$

Lemma 7.5.26 *If $s \approx s'$ then $N(s,\alpha,\rho,C) \;=\; N(s',\alpha,\rho,C)$ for all $\alpha \in Act \setminus \{\tau\}$, $\rho \in Distr_0(Act \setminus \{\tau\}$ and $C \in S/\approx$.*

Proof: If $\rho(\alpha) = 0$ for all α then $N(s,\alpha,\rho,C) = N(s',\alpha,\rho,C) = 0$. Now we assume that $\rho \in Distr(Act \setminus \{\tau\})$. For $A \in S/\approx$, $A \neq Div$, we choose some $s \in A$ with $\mathbf{P}(s,\tau,A) < 1$ and define

$$\mathbf{P}'(A,a,C) \;=\; \frac{\mathbf{P}(s,\alpha,C)}{1-\mathbf{P}(s,\tau,A)} \qquad \text{for } (a,C) \neq (\tau,A),$$

$$\mathbf{P}'(A,\tau) \;=\; \sum_{\substack{C\in S/\approx \\ C\neq A}} \mathbf{P}'(A,\tau,C), \qquad \mathbf{P}'(A,\alpha) \;=\; \sum_{C\in S/\approx} \mathbf{P}'(A,\alpha,C).$$

We define $\mathbf{P}'(Div,\tau) = 0$ and $\mathbf{P}'(Div,a,C) = 0$ if $(a,C) \neq (\tau,Div)$. For all $A \in S/\approx$ we define:

$$norm(A,\rho) \;=\; \sum_{\alpha\in Act\setminus\{\tau\}} \mathbf{P}'(A,\alpha) \cdot \rho(\alpha) \;+\; \mathbf{P}'(A,\tau).$$

First we show that

(1) $norm(A,\rho) = 0$ implies $Prob(s,\tau^*\alpha) = 0$ for all $s \in A$ and $\alpha \in Act \setminus \{\tau\}$ with $\rho(\alpha) > 0$.

Let $norm(A,\rho) = 0$ and $\alpha \in Act \setminus \{\tau\}$ with $\rho(\alpha) > 0$. Then:

- $\mathbf{P}'(A,\alpha,C) = 0$ for all $C \in S/\approx$,
- $\mathbf{P}'(A,\tau,C) = 0$ for all $C \in S/\approx$, $C \neq A$.

Hence, $\mathbf{P}(s,\alpha) = 0$ and $\mathbf{P}(s,\tau,S\setminus A) = 0$ for all $s \in A$. In particular, $Prob(s,\tau^*,S\setminus A) = 0$ for all $s \in A$. This yields $Prob(s,\tau^*\alpha) = 0$.

For A, $C \in S/\approx$, $\alpha \in Act \setminus \{\tau\}$ and $\rho \in Distr_0(Act \setminus \{\tau\})$, we define $N(A,\alpha,\rho,C)$ as follows. The vector $(N(A,\alpha,\rho,C))_{A\in S/\approx}$ is the unique solution of the linear equation system:

1. If $norm(A,\rho) = 0$ then $N(A,\alpha,\rho,C) = 0$.
2. If $norm(A,\rho) > 0$ then

$$N(A,\alpha,\rho,C) \;=\; \frac{\rho(\alpha)}{norm(A,\rho)} \cdot \mathbf{P}'(A,\alpha,C) \;+\; \sum_{\substack{B\in S/\approx \\ B\neq A}} \frac{\mathbf{P}'(A,\tau,t)}{norm(A,\rho)} \cdot N(B,\alpha,\rho,C).$$

In what follows, we suppose α, ρ and C to be fixed. It suffices to show that

(*) $N(s,\alpha,\rho,C) = N(A,\alpha,\rho,C)$ for all $s \in A$ and $A \in S/\approx$.

For all $s \in S$ we define $x_s = N([s],\alpha,\rho,C)$. (Recall that $[s]$ denotes the weak bisimulation equivalence class of s.) Clearly, (*) holds if $norm(A,\rho) = 0$. Now we assume $A \in S/\approx$ and $norm(A,\rho) > 0$. (In particular, $A \neq Div$.) Then:

(2) If $s \in A$ with $\mathbf{P}(s,\tau,A) < 1$ then

$$norm(s,\rho) - \mathbf{P}(s,\tau,A) = norm(A,\rho) \cdot (1 - \mathbf{P}(s,\tau,A)).$$

(3) If $s \in A$ with $\mathbf{P}(s,\tau,A) < 1$ then $norm(s,\rho) = \mathbf{P}(s,\tau,A)$ iff $norm(A,\rho) = 0$.

First we assume that $norm(A,\rho) = 0$. By (1), $Prob(s,\tau^*\alpha) = 0$ for all $s \in A$. Thus, by the definition of $N(\cdot)$, $N(s,\alpha,\rho,C) = 0 = N(A,\alpha,\rho,C) = x_s$.

Next we assume that $norm(A,\rho) > 0$. It is easy to see that, if $Prob(A,\tau^*\alpha,C) = 0$ then $x_s = 0 = N(A,\alpha,\rho,C)$ for all $s \in A$. In what follows, we suppose $Prob(A,\tau^*\alpha,C) > 0$. By (2) and (3),

- $norm(s,\rho) > 0$ for all $s \in A$,
- $norm(s,\rho) > \mathbf{P}(s,\tau,A)$ for all $s \in A$ with $\mathbf{P}(s,\tau,A) < 1$.

Let $s \in A$ with $\mathbf{P}(s,\tau,A) < 1$. Then:

$$\begin{aligned} x_s \;&=\; N(A,\alpha,\rho,C) \\ &=\; \frac{\rho(\alpha)}{norm(A,\rho)} \cdot \sum_{t\in C} \frac{\mathbf{P}(s,\alpha,t)}{1-\mathbf{P}(s,\tau,A)} \;+\; \frac{1}{norm(A,\rho)} \cdot \sum_{t\in S\setminus A} \frac{\mathbf{P}(s,\tau,t)}{1-\mathbf{P}(s,\tau,A)} \cdot x_t \\ &=\; \rho(\alpha) \cdot \frac{1-\mathbf{P}(s,\tau,A)}{norm(s,\rho)-\mathbf{P}(s,\tau,A)} \cdot \sum_{t\in C} \frac{\mathbf{P}(s,\alpha,C)}{1-\mathbf{P}(s,\tau,A)} \end{aligned}$$

$$+ \quad \frac{1-\mathbf{P}(s,\tau,A)}{norm(s,\rho)-\mathbf{P}(s,\tau,A)} \cdot \sum_{t \in S \setminus A} \frac{\mathbf{P}(s,\tau,t)}{1-\mathbf{P}(s,\tau,A)} \cdot x_t$$

$$= \quad \frac{\rho(\alpha)}{norm(s,\rho)-\mathbf{P}(s,\tau,A)} \cdot \mathbf{P}(s,\alpha,C)$$

$$+ \quad \frac{1}{norm(s,\rho)-\mathbf{P}(s,\tau,A)} \cdot \sum_{t \in S} \mathbf{P}(s,\tau,t) \cdot x_t \; - \; \frac{\mathbf{P}(s,\tau,A)}{norm(s,\rho)-\mathbf{P}(s,\tau,A)} \cdot x_s$$

Thus,

$$x_s \cdot \frac{norm(s,\rho)}{norm(s,\rho)-\mathbf{P}(s,\tau,A)} \; = \; x_s \cdot \left(1 + \frac{\mathbf{P}(s,\tau,A)}{norm(s,\rho)-\mathbf{P}(s,\tau,A)}\right)$$

$$= \; \frac{1}{norm(s,\rho)-\mathbf{P}(s,\tau,A)} \cdot \left(\rho(\alpha) \cdot \mathbf{P}(s,\alpha,C) \; + \; \sum_{t \in S} \mathbf{P}(s,\tau,t) \cdot x_t\right).$$

Hence,

$$x_s \; = \; \frac{\rho(\alpha)}{norm(s,\rho)} \cdot \mathbf{P}(s,\alpha,C) \; + \; \frac{1}{norm(s,\rho)} \cdot \sum_{t \in S} \; \mathbf{P}(s,\tau,t) \cdot x_t.$$

This yields $x_s = N(s,\alpha,\rho,C)$ for all $s \in A$. ■

Theorem 7.5.27 (cf. part (a) of Theorem 7.3.6, page 174) $\approx$ *is finer than* $\equiv_0$.

Proof: follows by Lemma 7.5.26 (page 189) and induction on the length of the probabilistic traces. ■

Chapter 8

Fairness of probabilistic choice

In Section 3.2.3 (page 45 ff) we argued that, for concurrent probabilistic systems, certain liveness properties cannot be established unless fairness assumptions are made about the way in which the non-deterministic choices are resolved. For probabilistic systems, one might also consider fairness with respect to the probabilistic choices. Clearly, the probabilities for the transitions can be viewed as conditions on the frequencies with which a certain transition is chosen. Thus, fairness assumptions about the probabilistic choices seem to be superfluous as they are expressed implicitly by the transition probabilities. Nevertheless, the probabilistic choices might be resolved unfair, and hence – as in the non-probabilistic (or concurrent probabilistic) case – it is possible that certain liveness probabilities are violated in some executions while they hold in all executions that are fair with respect to the probabilistic choices. For instance, if we flip a coin infinitely often then the property that eventually the outcome is "head" does *not* hold *for all* executions as it is possible that we always obtain "tail". But the probability for such an unfair behaviour is zero; i.e. the event "eventually head" holds *for almost all* executions. Thus, from a purely descriptive point of view, fairness with respect to the probabilistic choices is irrelevant because the probability measure of all executions that satisfy a certain linear time property does not depend on whether or not we shrink the attention to those executions where the probabilistic choices are resolved in a fair manner. However, by the results of Pnueli & Zuck [Pnue83, PnZu86a, PnZu93] fairness with respect to the probabilistic choices might be helpful for verifying qualitative properties for probabilistic systems. [Pnue83, PnZu86a, PnZu93] introduces two kinds of fairness with respect to the probabilistic choices (called *extremely fairness* and α*-fairness*) for (a variant of) concurrent probabilistic systems. Extreme and α-fairness are shown to be *sound* for the verification of qualitative linear time properties in the following sense: Whenever a linear time property φ holds for all execution sequences that are extremely fair (or α-fair) then φ holds with probability 1 (independent on the adversary).

The main goal of that chapter (whose results are published in [BaKw98a], a joint work with Marta Kwiatkowska) is to present a general notion of fairness with respect to the probabilistic choices (shortly called *p-fairness*) that subsumes extremely and α-fairness à la [Pnue83, PnZu86a, PnZu93] and the above mentioned soundness result. More precisely, we show that in order to demonstrate the validity of a qualitative linear time property φ for probabilistic processes it suffices to show – for some instance of our general p-fairness

notion – that φ holds for all p-fair execution sequences. This allows one, given an instance of our p-fairness notion, to reduce the verification of qualitative linear time properties of probabilistic processes to the non-probabilistic case: rather than compute the exact probabilities of the set of paths fulfilling φ, it is sufficient to establish that φ holds for all p-fair execution sequences by means of well-known *non-probabilistic* methods (deductive methods or model checking, see e.g. [LiPn85, MaPn92, CGH94, Lamp94, GPV$^+$95, MaPn95]).

P-fairness might also be useful for computing the probability measure of certain events. Given a set Π of fulpaths for which we want to compute $Prob(\Pi(s))$, one might define a "simpler" set Π' of fulpaths and show that, for any p-fair fulpath π, $\pi \in \Pi$ iff $\pi \in \Pi'$ (which yields $Prob(\Pi(s)) = Prob(\Pi'(s))$). Thus, the more "complicate" set Π might be replaced by the "simpler" set Π'.[1]

Organization of this chapter: The notion of p-fairness is introduced for both fully probabilistic and concurrent probabilistic systems. In Section 8.1, we introduce p-fairness for fully probabilistic systems and present our main result stating that, for every instance of our general notion of p-fairness, the set of p-fair execution sequences in bounded systems has probability 1 (Theorem 8.1.5, page 196). Section 8.2 deals with p-fairness concurrent probabilistic systems and shows that extreme and α-fairness and the above mentioned soundness result à la Pnueli & Zuck can be obtained from our general p-fairness notion.

8.1 P-fairness for fully probabilistic systems

We introduce a general notion of (strong) fairness with respect to the probabilistic choices in fully probabilitsic systems. For this, we suppose that the alternatives of the probabilistic choices are associated with "labels", with each label denoting e.g. a process name or an action. We say an execution sequence is *p-fair* if whenever a label is enabled infinitely many times then it is taken infinitely many times.

Definition 8.1.1 [p-fairness for fully probabilistic systems] *Let $(S, \mathbf{P})$ be a fully probabilistic system. A* p-fairness condition *for $(S, \mathbf{P})$ is a pair (L, l) where L is a non-empty countable set of* labels *and $\mathsf{l} : S \times S \to 2^{\mathsf{L}}$ a function with $\mathsf{l}(s,t) = \emptyset$ if $\mathbf{P}(s,t) = 0$. Let $\ell \in \mathsf{L}$. ℓ is called*

- enabled *in a state s iff $\ell \in \mathsf{l}(s,t)$ for some $t \in S$,*
- taken *in the i-th step of a fulpath π iff $\ell \in \mathsf{l}(\pi(i), \pi(i+1))$.*

A fulpath π is called p-fair with respect to $(\mathsf{L}, \mathsf{l}, \ell)$ *iff either ℓ is enabled only finitely many times in π or ℓ is taken infinitely often in π. π is called* p-fair with respect to (L, l) *(or* (L, l)-fair*) iff, for each label $\ell \in \mathsf{L}$, π is p-fair with respect to $(\mathsf{L}, \mathsf{l}, \ell)$.*

Note that all finite fulpaths are (L, l)-fair since each label ℓ is enabled only finitely many times. If (L, l) are understood from the context then we briefly speak about p-fairness

[1] For instance, in the correctness proofs of the model checking algorithms in Chapter 9 we make use of this technique and introduce *state* and *total fairness* as special instances of p-fairness (resp. a combination of p-fairness and fairness of non-deterministic choice in the case of total fairness) that we use to give simple characterizations for the wanted probability measures.

with respect to a label $\ell \in \mathsf{L}$ (rather than p-fairness with respect to $(\mathsf{L}, \mathsf{l}, \ell)$) and p-fairness (rather than (L, l)-fairness).

The set L of labels should be thought of as an abstraction which allows to express different kinds of fairness. Clearly, whether or not a fair probabilistic transition system yields a reasonable notion of fairness depends on the choice of L and ℓ.

Example 8.1.2 [**Process fairness**] To see why we need sets of labels we show how to define *process fairness*. We consider a fully probabilistic system which is obtained from a *probabilistic merge* of sequential randomizes processes $\mathcal{P}_1, \ldots, \mathcal{P}_n$. For this, we consider the parallel (interleaved) execution of $\mathcal{P}_1, \ldots, \mathcal{P}_n$ on a single processor where we assume a scheduler that decides randomly which of the processes $\mathcal{P}_i$ performs the next step. Given a global state s of the composed system, the scheduler decides – according to a certain distribution μ_s – which step has to be performed next. The transition probabilities of the composed system are given by these distributions μ_s in the sense that $\mathbf{P}(s,t) = \mu_s(t)$.[2] To define process fairness, let L to be the set of process names (i.e. $\mathsf{L} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$) and $\mathsf{l}(s,t)$ the set of processes that take part in the transition from the global state s to the global state t. Thus, $\mathsf{l}(s,t)$ might consist of a single process $\mathcal{P}_i$ (if the global transition $s \to t$ arises from an autonomous move by $\mathcal{P}_i$ while the other processes are idle) or of a set consisting of two or more processes (if a communication occurs). Then, (L, l)-fairness is process fairness in the following sense. For s to be a global state of the system, we say that process $\mathcal{P}_i$ is enabled in s iff there is some global t with $\mathbf{P}(s,t) > 0$ such that $\mathcal{P}_i$ takes part in the transition $s \to t$. Let $\pi = s_0 \to s_1 \to \ldots$ be an infinite fulpath. Then, π is (L, l)-fair iff whenever $\mathcal{P}_i$ is enabled in infinitely many states s_i then there are infinitely many indices i where $\mathcal{P}_i$ is activated in the transition $s_i \to s_{i+1}$.

Similarly, we can define *interaction fairness* which ensures that whenever the synchronization of certain processes $\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_k}$ is possible in infinitely many global states then there are infinitely many steps where (exactly) the processes $\mathcal{P}_{i_1}, \ldots, \mathcal{P}_{i_k}$ perform a synchronized step. For this, we deal with L to be the powerset of $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and the labelling function l that assigns to each transition $s \to t$ the singleton set $\mathsf{l}(s,t)$ of all processes that are activated in the step $s \to t$. Then, (L, l)-fairness is interaction fairness. ■

Remark 8.1.3 [**Action fairness**] To define *action (event) fairness* in action-labelled probabilistic systems we have to deal with a slight modification of p-fairness. For an action-labelled fully probabilistic systems $(S, Act, \mathbf{P})$ we use a labelling function $\mathsf{l} : S \times Act \times S \to 2^{\mathsf{L}}$ that assigns to each (action-labelled) step a set of labels. To define action fairness, we deal with $\mathsf{L} = Act$ and define $\mathsf{l}(s,a,t) = \{a\}$. Then, (L, l)-fairness ensures that whenever an action $a \in Act$ is enabled infinitely often then a is taken infinitely often. More precisely, if $\pi = s_0 \stackrel{a_1}{\to} s_1 \stackrel{a_2}{\to} \ldots$ is an infinite fulpath then π is (L, l)-fair iff whenever $\mathbf{P}(s_i, a) > 0$ for infinitely many i then $a = a_i$ for infinitely many i. ■

[2] This can be viewed as a generalization of the probabilistic merge operator proposed by Baeten, Bergstra & Smolka [BBS92]. [BBS92] deal with a (binary) probabilistic merge operator $\mathcal{P}_1 \|_{p,q} \mathcal{P}_2$ parametrized by probabilities p, $q \in [0,1]$ (with $p + q \leq 1$) which are interpreted as follows. A communication between $\mathcal{P}_1$ and $\mathcal{P}_2$ occurs with probability $1 - q$. $p \cdot q$ is the probability for $\mathcal{P}_1$ to make an autonomous move while $\mathcal{P}_2$ is idle; similarly, $(1-p) \cdot q$ is the probability for $\mathcal{P}_2$ to make an autonomous move while $\mathcal{P}_1$ is idle.

Notation 8.1.4 [**The sets** $pFair_\ell$ **and** $pFair_{(\mathsf{L},\mathsf{I})}$] *Let* (L,I) *be a p-fairness condition for a fully probabilistic system* $(S,\mathbf{P})$. *For* $\ell \in \mathsf{L}$, *we define* $pFair_{(\mathsf{L},\mathsf{I},\ell)}$ *to be the set of fulpaths that are p-fair with respect to* ℓ.

$$pFair_{(\mathsf{L},\mathsf{I})} = \bigcap_{\ell\in\mathsf{L}} pFair_{(\mathsf{L},\mathsf{I},\ell)}$$

denotes the set of all fulpaths that are (L,I)*-fair.*

If (L,I) are understood from the context then we briefly write $pFair$ rather than $pFair_{(\mathsf{L},\mathsf{I})}$ and $pFair_\ell$ rather than $pFair_{(\mathsf{L},\mathsf{I},\ell)}$. To see that $pFair(s)$ is measurable we first express p-fairness as a linear time formula, and then use a well-known result [Vard85, PnZu93] stating that the set of paths fulfilling a given linear time formula is measurable. The underlying linear time logic is a slight modification of *LTL* (see Section 9.1.3, page 212) which uses labelled next step operators X_ℓ rather than the usual (unlabelled) next step operator X. Formulas are built from: the truth values *tt* and *ff*, the atomic propositions $enabled(\ell)$ for each label $\ell \in \mathsf{L}$, the usual boolean connectives $\wedge$, $\vee$, $\neg$, $\rightarrow$, and the temporal operators $\Box$ ("always"), $\Diamond$ ("eventually") and a next-step operator X_ℓ for each label $\ell \in \mathsf{L}$. The formulas are interpreted over the fulpaths of fully probabilistic system $(S,\mathbf{P})$ with a p-fairness condition (L,I). We define the satisfaction relation $\models$ as follows. Let π be a fulpath in $(S,\mathbf{P})$. Then,

$(\pi,j) \models enabled(\ell)$ iff ℓ is enabled in $\pi(j)$
$(\pi,j) \models X_\ell\ \varphi$ iff $|\pi| \geq j+1$, $\ell \in \mathsf{I}(\pi(j),\pi(j+1))$ and $(\pi,j+1) \models \varphi$.

The other operators are interpreted in the usual way (see e.g. [MaPn92]). We write $\pi \models \varphi$ iff $(\pi,0) \models \varphi$. As shown e.g. in [Vard85, PnZu93], for a given formula φ, the set of fulpaths π starting in a fixed state $s \in S$ such that $\pi \models \varphi$ is measurable. We define

$$\varphi_\ell \ = \ \Box\Diamond enabled(\ell) \ \rightarrow \ \Box\Diamond taken(\ell) \quad \text{where} \quad taken(\ell) \ = \ X_\ell\ tt.$$

Clearly, $\pi \models \varphi_\ell$ iff π is p-fair with respect to ℓ. Thus,

$$pFair_\ell(s) \ = \ \{\pi \in Path_{ful}(s) : \pi \ \models \ \varphi_\ell\ \}$$

and $pFair_{(\mathsf{L},\mathsf{I})}(s) = \bigcap_{\ell\in\mathsf{L}} pFair_\ell(s)$ are measurable.

Our results rely on the *boundedness* of (possibly infinite-state) fully probabilistic systems (see Definition 3.1.12, page 38). We now state our main result which shows that for each instance of p-fairness in a bounded system, the measure for the p-fair fulpaths is 1.

Theorem 8.1.5 *Let* (L,I) *be a p-fairness condition for a bounded fully probabilistic system* $(S,\mathbf{P})$ *and* $s \in S$. *Then,*

$$Prob(pFair_{(\mathsf{L},\mathsf{I})}(s)) = 1.$$

Proof: Let $c > 0$ be a real number such that $\mathbf{P}(s,t) > 0$ implies $\mathbf{P}(s,t) \geq c$. It suffices to show that $pFair_\ell(s) = 1$ for all $\ell \in \mathsf{L}$.[3] Let ℓ be a fixed label and Π be the set of all fulpaths where ℓ is enabled infinitely often and which totally ignore ℓ-steps, i.e.

$$\Pi \ = \ \{\pi \in Path_{ful} : \pi \models \ \Box\Diamond enabled(\ell) \wedge \Box\neg taken(\ell)\}$$

[3] Note that $Prob(\Pi_i) = 1$ implies $Prob(\bigcap_i \Pi_i) = 1$ which holds in each probabilistic space.

If σ is a finite path with $last(\sigma) = s$ then we briefly write $\sigma \circ \Pi(s)$ to denote the set $\{\sigma \circ \pi : \pi \in \Pi(s)\}$. Let $Path^t_{fin}$ be the set of all finite paths ending in t and

$$T \;=\; \{t \in S : t \models enabled(\ell)\}.$$

We show that, for all $s \in S$, $Prob(\Pi(s)) = 0$ and that $Path_{ful}(s) \setminus pFair_\ell(s)$ can be written as a countable union of sets of the form $\sigma \circ \Pi(t)$ where $\sigma \in Path^t_{fin}$ and $t \in T$.

Claim 1: $Prob(\Pi(s)) = 0$ for all $s \in S$.

Proof: We define Σ to be the set of finite paths $\sigma \in Path_{fin}$ such that $|\sigma| \geq 1$, $\ell \notin \mathsf{l}(s, \sigma(1))$, $\sigma(i) \notin T$, $i = 1, \ldots, |\sigma| - 1$, and $last(\sigma) \in T$. For $t \in T$, let $\Sigma^t = \Sigma \cap Path^t_{fin}$. Then, for $t \in T$ and $s \in S$,

$$\Sigma^t(s) = \{\sigma \in \Sigma : first(\sigma) = s,\ last(\sigma) = t\}.$$

$\Sigma(s)$ is countable and $\Sigma(s) \;=\; \bigcup_{t \in T} \Sigma^t(s)$ where $\Sigma^t(s) \cap \Sigma^{t'}(s) = \emptyset$ if $t \neq t'$. Thus,

$$(1) \qquad \sum_{\sigma \in \Sigma(s)} \mathbf{P}(\sigma) \;=\; \sum_{t \in T} \sum_{\sigma \in \Sigma^t(s)} \mathbf{P}(\sigma).$$

We have

$$\Pi(s) \;=\; \bigcup_{t \in T} \bigcup_{\sigma \in \Sigma^t(s)} \sigma \circ \Pi(t) \;\text{ for all } s \in S.$$

As $\sigma \circ \Pi(t) \;\cap\; \sigma' \circ \Pi(t') = \emptyset$ if $(\sigma, t) \neq (\sigma', t')$, and as $\sigma \circ \Pi(t)$ is a measurable set with $Prob(\sigma \circ \Pi(t)) = \mathbf{P}(\sigma) \cdot Prob(\Pi(t))$ we obtain:

$$(2) \qquad Prob(\Pi(s)) \;=\; \sum_{t \in T} \sum_{\sigma \in \Sigma^t(s)} \mathbf{P}(\sigma) \cdot Prob(\Pi(t)) \qquad \text{for all } s \in S.$$

Let $t \in T$. As $t \models enabled(\ell)$ there is some $s_t \in S$ with $\ell \in \mathsf{l}(t, s_t)$. Since $\mathbf{P}(t, s_t) \geq c$ we obtain:

$$(3) \qquad \sum_{\sigma \in \Sigma(t)} \mathbf{P}(\sigma) \;\leq\; \sum_{s \neq s_t} \mathbf{P}(t, s) \;\leq\; 1 \;-\; \mathbf{P}(t, s_t) \;\leq\; 1 - c \;\text{ for all } t \in T.$$

We show by induction on k that $Prob(\Pi(t)) \leq (1 - c)^k$ for all $t \in T$. In the basis of induction ($k = 0$) there is nothing to show. In the induction step ($k \Longrightarrow k + 1$) we suppose that $Prob(\Pi(t)) \leq (1 - c)^k$ for all $t \in T$. By (1), (2), (3) and the induction hypothesis we get for all $t \in T$:

$$Prob(\Pi(t)) \;=\; \sum_{u \in T} \sum_{\sigma \in \Sigma^u(t)} \mathbf{P}(\sigma) \cdot Prob(\Pi(u)) \;\leq\; (1-c)^k \sum_{u \in T} \sum_{\sigma \in \Sigma^u(t)} \mathbf{P}(\sigma)$$

$$= \;\; (1-c)^k \sum_{\sigma \in \Sigma(t)} \mathbf{P}(\sigma) \;\leq\; (1-c)^{k+1}.$$

We conclude $Prob(\Pi(t)) = 0$ for all $t \in T$. Thus, $Prob(\Pi(s)) = 0$ for all $s \in S$ (by (2)). ⌋

Claim 2: $Prob(pFair_\ell(s)) = 1$ for all $s \in S$.

Proof: It is clear that

$$Path_{ful}(s) \setminus pFair_\ell(s) \;=\; \bigcup_{t \in T} \bigcup_{\sigma \in Path^t_{fin}(s)} \sigma \circ \Pi(t).$$

Note that $Path^t_{fin}$ is countable. Claim 1 yields

$$Prob(\sigma \circ \Pi(t)) \;=\; \mathbf{P}(\sigma) \cdot Prob(\Pi(t)) = 0$$

for all $t \in T$. Hence,

$$Prob(Path_{ful}(s) \setminus pFair_\ell(s)) \;\leq\; \sum_{t \in T} \; \sum_{\sigma \in Path^t_{fin}(s)} Prob(\sigma \circ \Pi(t)) \;=\; 0$$

and $Prob(pFair_\ell(s)) = 1$. ⌋■

Remark 8.1.6 If we drop the assumption that $(S, \mathbf{P})$ is bounded then the probability of the fair paths might be less than 1. As a counter-example consider the system of Figure 8.1, i.e. the system $(S, \mathbf{P}, \mathsf{L}, \mathsf{l})$ where $S = \{t\} \cup \{s_0, s_1, \ldots\}$, $\mathsf{L} = \{\ell\}$ and

$$\mathbf{P}(s_i, v) \;=\; \begin{cases} 2^{-r_i} & : \text{ if } v = s_{i+1} \\ 1 - 2^{-r_i} & : \text{ if } v = t \\ 0 & : \text{ otherwise} \end{cases} \qquad \mathsf{l}(s_i, v) \;=\; \begin{cases} \emptyset & : \text{ if } v = s_{i+1} \\ \{\ell\} & : \text{ if } v = t \end{cases}$$

and $\mathbf{P}(t, s_i) = 0$, $\mathbf{P}(t, t) = 1$, $\mathsf{l}(t, t) = \emptyset$. Here, $(r_i)_{i \geq 0}$ is a sequence of positive reals where $\sum_{i \geq 0} \; r_i$ is convergent. $\pi = s_0 \to s_1 \to s_2 \to \ldots$ is not p-fair as ℓ is continuously enabled

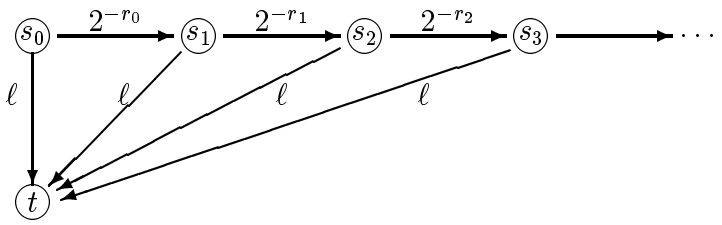


Figure 8.1:

but never taken in π. Any other fulpath is finite and hence p-fair. Hence,

$$Prob(pFair(s_0)) \;=\; 1 \;-\; \lim_{k \to \infty} \; \mathbf{P}(s_0 \to s_1 \to \ldots \to s_k)$$

$$=\; 1 \;-\; \lim_{k \to \infty} \; 2^{-(r_0 + r_1 + \ldots + r_{k-1})} \;=\; 1 - 2^{-r} < 1$$

where $r = \sum_{i \geq 0} \; r_i$. ■

Soundness of p-fairness: Theorem 8.1.5 (page 196) yields the soundness of proving the validity of qualitative linear time properties under p-fairness constraints in the following sense. We suppose a (linear time) logic $\mathbf{L}$ and, for a fixed bounded probabilistic system $(S, \mathbf{P})$, a satisfaction relation $\models \subseteq \; Path_{ful} \times \mathbf{L}$ such that, for each $s \in S$ and each formula φ of $\mathbf{L}$, the set $\{\pi \in Path(s) : \pi \models \varphi\}$ is measurable.[4] s is called *φ-valid* iff $Prob\{\pi \in Path(s) : \pi \models \varphi\} \;=\; 1$. By Theorem 8.1.5 (page 196) we obtain the following.

Corollary 8.1.7 *Let $(S, \mathbf{P})$ be a bounded fully probabilistic system and (L, l) a p-fairness condition on $(S, \mathbf{P})$ Then, for all $s \in S$:*

[4]For example, $\mathbf{L}$ might be the standard propositional linear time logic *LTL* (see Section 9.1.3, page 212 ff).

If $\pi \models \varphi$ *for all* $\pi \in pFair_{(\mathsf{L},\mathsf{I})}(s)$ *then* s *is* φ*-valid.*

Proof: follows immediately from Theorem 8.1.5 (page 196). ■

Hence, in order to establish a (qualitative) linear time property φ for a probabilistic process, it suffices to show that all p-fair fulpaths satisfy φ for *some* instance of our general p-fairness notion (which can be achieved with well-known non-probabilistic methods).

Corollary 8.1.8 *Let* $(S, \mathbf{P})$ *be a bounded fully probabilistic system,* (L,I) *a p-fairness condition on* $(S, \mathbf{P})$*,* $s \in S$ *and* Π *a subset of* $Path_{ful}$ *such that* $\Pi(s)$ *is measurable. Then:*

$$Prob\left(\Pi(s) \cap pFair_{(\mathsf{L},\mathsf{I})}\right) \;=\; Prob(\Pi(s)).$$

Proof: follows immediately from Theorem 8.1.5 (page 196). ■

In particular, whenever φ is a linear time formula then the probability of the set of fulpaths fulfilling φ *equals* the probability of the set of p-fair fulpaths fulfilling φ. In other words, whether or not a (qualitative or quantitative) linear time property holds for a probabilistic process does not depend on whether fairness with respect to probabilistic choice is required. Hence, from a purely descriptive point of view, fairness with respect to probabilistic choice is irrelevant.[5]

8.2 P-fairness for concurrent probabilistic systems

In this section, we define p-fairness for concurrent probabilistic systems and show that the soundness result of the p-fairness approach for establishing qualitative linear time properties carries over to the concurrent case. Moreover, we show that the extreme and α-fairness of Pnueli & Zuck [Pnue83, PnZu86a, PnZu93] are special instances of our general p-fairnes notion. Hence, the soundness results established in [Pnue83, PnZu93] are special cases of the results presented here. We cannot expect a general *completeness* result (in the sense that, if a linear time property φ holds with probability 1 in all adversaries then φ holds on all p-fair execution sequences) as in [Pnue83] extreme fairness is shown to be incomplete. However, we are able to show that – in some sense – α-fairness (shown to be complete in [PnZu93]) is *the only* p-fairness notion that is complete for proving validity of qualitative linear time properties (Lemma 8.2.11, page 203).

As in fully probabilistic case, we assume that the probabilistic alternatives are associated with certain labels. P-fairness in concurrent systems ensures that whenever a label ℓ is enabled infinitely often then ℓ is taken infinitely often where the underlying definition of "enabled" in a state s only depends on the chosen non-deterministic alternatives and the associated distribution $\mu \in Steps(s)$ (but not on the other non-deterministic alternatives $\nu \in Steps(s) \setminus \{\mu\}$).

[5] One might wonder why such a result is possible, since in the non-probabilistic case it is folklore knowledge that certain liveness properties cannot be established without suitable fairness assumptions. It is worth noting that φ-validity of a state s in a probabilistic transition system is weaker than φ-validity in the corresponding non-probabilistic transition system. Recall that, in the non-probabilistic case, a state s of a transition system is said to be φ-valid iff *all* fulpaths starting in s satisfy φ, whereas in the probabilistic case, φ-validity requires that φ holds for *almost all* fulpaths starting in s.

Definition 8.2.1 [p-fairness for concurrent systems] *Let $\mathcal{S} = (S, Steps)$ be a concurrent probabilistic system. A* p-fairness condition *on $\mathcal{S}$ is a pair (L, I) consisting of a nonempty countable set L of labels and a function*

$$\mathsf{I} : \{(\sigma, \mu, t) : \sigma \in Path_{fin}, \mu \in Steps(last(\sigma)), t \in Supp(\mu)\} \longrightarrow 2^{\mathsf{L}}.$$

If π is a fulpath in $\mathcal{S}$ and $\ell \in \mathsf{L}$ then we say

- *ℓ is* enabled *in the i-th step of π iff*

$$\ell \in \mathsf{I}(\pi^{(i)}, step(\pi, i), s)$$

  for some $s \in Supp(step(\pi, i))$,
- *ℓ is* taken *in the i-th step of π iff $\ell \in \mathsf{I}(\pi^{(i)}, step(\pi, i), \pi(i+1))$.*

If $\ell \in \mathsf{L}$ then π is called p-fair with respect to $(\mathsf{L}, \mathsf{I}, \ell)$ *iff either ℓ is enabled only finitely many times in π or ℓ is taken infinitely many times in π. π is called* p-fair with respect to (L, I) *(or* (L, I)-fair*) iff, for each label $\ell \in \mathsf{L}$, π is p-fair with respect to $(\mathsf{L}, \mathsf{I}, \ell)$.*

If (L, I) are understood from the context then we briefly speak about p-fairness with respect to a label $\ell \in \mathsf{L}$ (rather than p-fairness with respect to $(\mathsf{L}, \mathsf{I}, \ell)$) and p-fairness (rather than (L, I)-fairness).

Notation 8.2.2 [The sets $pFair_\ell$ and $pFair_{(\mathsf{L},\mathsf{I})}$] *Let $(\mathsf{L}, \mathsf{I}, \ell)$ be a p-fairness condition for a concurrent probabilistic system $(S, Steps)$. $pFair_{(\mathsf{L},\mathsf{I},\ell)}$ (or briefly $pFair_\ell$) denotes the set of fulpaths π which are (L, I)-fair with respect to ℓ, and $pFair_{(\mathsf{L},\mathsf{I})}$ the set of fulpaths π which are (L, I)-fair.*

Theorem 8.2.3 *Let $(S, Steps)$ be a finite concurrent probabilistic system, (L, I) a p-fairness condition on $(S, Steps)$ and A an adversary of $(S, Steps)$. Then,*

$$Prob(pFair^A_{(\mathsf{L},\mathsf{I})}(s)) \;=\; 1$$

for all $s \in S$. Moreover, if Π is a subset of $Path^A_{ful}$ where $\Pi(s)$ is measurable then

$$Prob(\Pi(s) \cap pFair_{(\mathsf{L},\mathsf{I})}) = Prob(\Pi(s)).$$

Proof: It is easy to see that, for A to be an adversary of $(S, Steps)$, a fulpath $\pi \in Path^A$ is (L, I)-fair if and only if π is (L, I)-fair as a path in the fully probabilistic system $\mathcal{S}^A$.[6] As $(S, Steps)$ is finite, for each adversary A, the associated fully probabilistic system $\mathcal{S}^A$ is bounded. Thus, the claim follows by Theorem 8.1.5 (page 196) and Corollary 8.1.8 (page 199). ■

As before, we suppose a (linear time) logic $\mathbf{L}$ and a satisfaction relation $\models \subseteq Path_{ful} \times \mathbf{L}$ such that, for each $s \in S$, each adversary A and each formula φ of $\mathbf{L}$, the set $\{\pi \in Path^A_{ful}(s) : \pi \models \varphi\}$ is measurable. We then obtain:

[6] Recall that we identify each path γ in $(S, Steps)$ with the path $\gamma = \gamma^{(0)} \to \gamma^{(1)} \to \gamma^{(2)} \to \ldots$ in $\mathcal{S}^A$. See Chapter 3, page 42.

Corollary 8.2.4 *If* (L,I) *is a p-fairness condition for a finite concurrent probabilistic system* $(S, Steps)$ *and* φ *a formula of* $\mathbf{L}$ *then*

$$Prob\{\pi \in pFair^A_{(\mathsf{L},\mathsf{I})}(s) : \pi \models \varphi\} \;=\; Prob\{\pi \in Path^A_{ful}(s) : \pi \models \varphi\}$$

for all adversaries A.

Proof: follows immediately from Theorem 8.2.3 (page 200). ■

We call a state s *φ-valid* iff $Prob\{\pi \in Path^A_{ful}(s) : \pi \models \varphi\} = 1$ for all adversaries A. Furthermore, the soundness of proving the validity of linear time formulas under (L,I)-fairness follows.

Corollary 8.2.5 *Whenever* (L,I) *is a p-fairness condition for a finite concurrent probabilistic system* $(S, Steps)$, φ *is a formula of* $\mathbf{L}$ *and* $s \in S$. *Then:*

$$\textit{If } \pi \models \varphi \textit{ for all fulpaths } \pi \in pFair_{(\mathsf{L},\mathsf{I})}(s) \textit{ then } s \textit{ is } \varphi\textit{-valid.}$$

Proof: follows immediately from Corollary 8.2.4 (page 201). ■

Extreme and α-fairness à la Pnueli & Zuck: In [Pnue83] and [PnZu93] notions of extreme fairness and α-fairness are introduced for (a variant of) concurrent probabilistic systems. The definition of extreme fairness [Pnue83] employs a collection state predicates (described by first order formulas), whereas α-fairness [PnZu93] uses some kind of linear time logic with past operators. We now adapt the notions extreme and α-fairness for our model of concurrent probabilistic systems and show that extreme and α-fairness (adapted for our less general model of concurrent probabilistic systems) are instances of p-fairness conditions as defined above.[7]

For the definition of extreme fairness we suppose a set $StatePred \subseteq 2^S$ (where each element $\chi \in StatePred$ represents a state predicate).

Definition 8.2.6 [Extreme fairness, cf. [Pnue83, PnZu86a]] *Let* π *be a fulpath in* $\mathcal{S}$. π *is called* extremely fair *iff, for each* $\chi \in StatePred$, *each* $s \in S$ *and* $\mu \in Steps(s)$, *whenever* $step(\pi, i) = \mu$ *for infinitely many* $i \geq 0$ *with* $\pi(i) \in \chi$ *then there are infinitely many indices* $i \geq 0$ *with* $\pi(i) \in \chi$, $step(\pi, i) = \mu$ *and* $\pi(i+1) = s$.

To define α-fairness we suppose $PastForm$ to be a set consisting of subsets of $Path_{fin}$ where we assume that each element χ of $PastForm$ represents a past formulas of some linear time logic [LPZ85]. Note that, for χ to be a past formula, χ can be identified with the set of all finite paths σ such that each fulpath $\pi \in \sigma\uparrow$ fulfills χ.

Definition 8.2.7 [α-fairness, cf. [PnZu93]] *A fulpath* π *is called* α-fair *iff, for each* $\chi \in PastForm$, $s \in S$ *and* $\mu \in Steps(s)$, *whenever there are infinitely many indices* i *with* $\pi^{(i)} \in \chi$ *and* $step(\pi, i) = \mu$ *then there are infinitely many indices* j *with* $\pi^{(j)} \in \chi$, $step(\pi, j) = \mu$ *and* $\pi(j+1) = s$.

The next lemma shows that extreme and α-fairness are instances of p-fairness conditions in our sense:

[7] See Section 3.6, page 63 for the precise connection between the model à la Pnueli & Zuck and ours.

Lemma 8.2.8 *Let* (L,I) *be a p-fairness condition for a concurrent probabilistic system* $(S, Steps)$ *and* π *be a fulpath in* $(S, Steps)$*. Then:*

(a) π *is extremely fair if and only if* π *is* $(\mathsf{L}_{efair},\mathsf{I}_{efair})$*-fair where*

$$\mathsf{L}_{efair} \;=\; \{(\chi,\mu,s) : \chi \in StatePred, s \in S, \mu \in Steps(s)\},$$
$$\mathsf{I}_{efair}(\sigma,\mu,s) \;=\; \{(\chi,\mu,s) \in \mathsf{L}_{efair} : last(\sigma) \in \chi\ \}.$$

(b) π *is* α*-fair if and only if* π *is* $(\mathsf{L}_{\alpha fair},\mathsf{I}_{\alpha fair})$*-fair where*

$$\mathsf{L}_{\alpha fair} \;=\; \{(\chi,\mu,s) : \chi \in PastForm, s \in S, \mu \in Steps(s)\},$$
$$\mathsf{I}_{\alpha fair}(\sigma,\mu,s) \;=\; \{(\chi,\mu,s) \in \mathsf{L}_{\alpha fair} : \sigma \in \chi\}.$$

Proof: We only show (a) as (b) can be shown similarly. For simplicity, we write L and I instead of L_{efair} and I_{efair} respectively. Let π be a fulpath in $(S, Steps)$.

"only if": Let π be extremely fair and let $\ell = (\chi,\mu,s) \in \mathsf{L}$ such that ℓ is infinitely often enabled in π. Let I be the set of indices $i \geq 0$ such that ℓ is enabled in the i-th state of π. Then, $\pi(i) \in \chi$ and $step(\pi,i) = \mu$ for all $i \in I$. As π is extremely fair there exists an infinite subset J of I such that $\pi(j+1) = s$ for all $j \in J$. Hence, $\ell \in \mathsf{I}(\pi^{(j)},\mu,s)$ for all $j \in J$, i.e. ℓ is taken infinitely often in π.

"if": We suppose π to be (L,I)-fair and $step(\pi,i) = \mu$ for infinitely many indices i with $\pi(i) \in \chi$. Let s be a mode of μ and let $\ell = (\chi,\mu,s)$. Then, ℓ is enabled infinitely often in π. Hence, ℓ is taken infinitely often in π, i.e. there are infinitely many indices j with $\ell \in \mathsf{I}(\pi^{(j)},\pi^{(j+1)})$. For each such index j, $\pi(j) \in \chi$, $\mu = step(\pi,j)$ and $\pi(j+1) = s$. Thus, π is extremely fair. ■

From part (a) of Lemma 8.2.8 we can deduce that our soundness result (Corollary 8.2.5, page 201) is a generalization of the result of [Pnue83] which states the soundness of proving qualitative properties under extreme fairness. In [PnZu93] it is shown that, for each state s and each linear time formula φ,

$$s \text{ is } \varphi\text{-valid iff } \pi \models \varphi \text{ holds for all } \alpha\text{-fair fulpaths } \pi \in Path_{ful}(s).$$

The "if"-part is an instance of Corollary 8.2.5 (page 201), whereas the "only-if"-part (the completeness of the α-fairness approach) is not. The reason for this is that a general completeness result cannot be established, as it is shown in [Pnue83] that extreme fairness is not a necessary condition for the validity of linear time formulas.

In the remainder of this section, we show that α-fairness is the only p-fairness notion which is complete for verifying qualitative properties expressed by linear time formulas with past operators. We suppose that formulas of the linear time logic $\mathbf{L}$ are built from the truth values *tt* and *ff*, atomic propositions, the usual boolean connectives, and the temporal operators $\mathcal{U}$ ("until"), $\mathcal{U}^{-1}$ ("since"), X^{-1} ("previous step") and labelled next-step operators X_μ, $\mu \in \bigcup_s Steps(s)$. The usual next-step operator X can be derived from the labelled next-step operators by putting $X\varphi = \bigvee_\mu X_\mu\varphi$. $\mathbf{L}_{past}$ denotes the set of *past formulas* of $\mathbf{L}$, i.e. formulas which are built from atomic propositions, the boolean connectives and the operators $\mathcal{U}^{-1}$ and X^{-1}.

We fix a concurrent probabilistic system $(S, Steps)$ together with a satisfaction relation $\models \subseteq Path_{ful} \times I\!N \times \mathbf{L}$ (where $I\!N$ is the set of non-negative integers) with $(\pi,j) \models X_\mu\varphi$ iff $step(\pi,j) = \mu$ and $(\pi,j+1) \models \varphi$. The remaining operators are interpreted in the usual way (see e.g. [MaPn92]). The satisfaction relation $\models \subseteq Path_{ful} \times \mathbf{L}$, as used earlier, is

given by $\pi \models \varphi$ iff $(\pi, 0) \models \varphi$. Let

$$\Pi_\varphi \;=\; \{\pi \in Path_{ful} : \pi \models \varphi\}.$$

Then, s is called *φ-valid* iff $Prob(\Pi_\varphi^A(s)) \;=\; 1$ for all adversaries A. For a past formula ψ and a finite path σ with $|\sigma| = j$, we define

$$\sigma \models \psi \text{ iff } (\pi, j) \models \psi \text{ for all fulpaths } \pi \text{ with } \pi^{(j)} = \sigma$$

(or equivalently, iff $(\pi, j) \models \psi$ for some fulpath π with $\pi^{(j)} = \sigma$). Let χ_ψ be the set of finite paths σ with $\sigma \models \psi$ and $PastForm = \{\chi_\psi : \psi \in \mathbf{L}_{past}\}$. Let $(\mathsf{L}_{\alpha fair}, \mathsf{l}_{\alpha fair})$ be as in part (b) of Lemma 8.2.8 (page 202). We write $\alpha Fair$ instead of $pFair_{(\mathsf{L}_{\alpha fair}, \mathsf{l}_{\alpha fair})}$.

Definition 8.2.9 [Completeness of p-fairness conditions] *Let (L, l) be a p-fairness condition for a concurrent probabilistic system $(S, Steps)$ and $\mathbf{L}$ a linear time logic as before.* (L, l) *is called* complete *(for verifying qualitative properties expressed as formulas of $\mathbf{L}$) iff the following holds:*

$$\textit{if } s \textit{ is } \varphi\textit{-valid then } \pi \models \varphi \textit{ for all } \varphi \in pFair_{(\mathsf{L},\mathsf{l})}$$

for all formulas φ of $\mathbf{L}$ and all states $s \in S$.

It is easy to see that the completeness result of [PnZu93] (where labelled next-step operators are not used) carries over to $\mathbf{L}$, i.e. if s is φ-valid then $\alpha Fair(s) \subseteq \Pi_\varphi(s)$. Thus, $(\mathsf{L}_{\alpha fair}, \mathsf{l}_{\alpha fair})$ is complete.

Definition 8.2.10 [Expressiveness of L for a p-fairness condition] *Let (L, l) be a p-fairness condition for a concurrent probabilistic system $(S, Steps)$ and $\mathbf{L}$ a linear time logic as before.* $\mathbf{L}$ *is called* expressive for (L, l) *iff for each $\ell \in L$ there exists a formula φ of $\mathbf{L}$ with $\Pi_\varphi = pFair_{(\mathsf{L},\mathsf{l},\ell)}$.*

We may assume that for each state $s \in S$ there is an atomic proposition a_s with $(\pi, j) \models a_s$ iff $\pi(j) = s$. Then, $\mathbf{L}$ is expressive for $(\mathsf{L}_{\alpha fair}, \mathsf{l}_{\alpha fair})$ as, for $\ell = (\chi_\psi, \mu, s)$ and

$$\varphi_\ell \;=\; \Box\Diamond enabled(\ell) \;\rightarrow\; \Box\Diamond(\psi \wedge X_\mu a_s)$$

where $enabled(\ell) \;=\; \psi \wedge X_\mu tt$, we have that

$$\pi \models \varphi_\ell \text{ iff } \pi \text{ is } (\mathsf{L}_{\alpha fair}, \mathsf{l}_{\alpha fair})\text{-fair with respect to } \ell.$$

The next lemma shows that – in some sense – α-fairness is the only p-fairness notion which is complete for verifying qualitative linear time properties.

Lemma 8.2.11 *Let $(S, Steps)$, $\mathbf{L}$ and (L, l) be as before. If $\mathbf{L}$ is expressive for (L, l) then (L, l) is complete iff $pFair_{(\mathsf{L},\mathsf{l})} = \alpha Fair$.*

Proof: It suffices to show that if (L, l), $(\mathsf{L}', \mathsf{l}')$ are p-fairness conditions such that

- $\mathbf{L}$ is expressive for (L, l),
- $(\mathsf{L}', \mathsf{l}')$ is complete

then $pFair_{(\mathsf{L}',\mathsf{l}')}(s) \subseteq pFair_{(\mathsf{L},\mathsf{l},\ell)}(s)$ for all $s \in S$ and $\ell \in \mathsf{L}$. Since $\mathbf{L}$ is expressive for (L, l) there is a formula φ with $\Pi_\varphi = pFair_{(\mathsf{L},\mathsf{l},\ell)}$ where $\Pi_\varphi = \{\pi \in Path_{ful} : \pi \models \varphi\}$. Since

$$Prob(\Pi_\varphi^A(s)) \;=\; Prob(pFair^A_{(\mathsf{L},\mathsf{l},\ell)}(s)) \;= 1$$

for all adversaries A we obtain φ-validity of s. Hence, $pFair_{(\mathsf{L}',\mathsf{l}')}(s) \subseteq pFair_{(\mathsf{L},\mathsf{l},\ell)}(s)$ by the completeness of $(\mathsf{L}', \mathsf{l}')$. Thus, $pFair_{(\mathsf{L}',\mathsf{l}')}(s) \subseteq pFair_{(\mathsf{L},\mathsf{l})}(s)$. ■

Chapter 9

Verifying quantitative temporal properties

The main goal of this chapter is to present the basic concepts of the algorithmic methods for verifying quantitative properties specified in the temporal logical framework.[1] We consider both fully probabilistic and concurrent probabilistic systems. For the handling of fully probabilistic systems, we recall techniques proposed in the literature (mainly the methods of Hansson & Jonsson [HaJo94]).[2] For the latter (concurrent) case, we mainly concentrate on methods that involve *fairness*. The underlying fairness notions are those of [HSP83, Vard85] (see Section 3.2.3, page 45).[3] For this, we first recall the approach of Bianco & deAlfaro [BidAl95, dAlf97a, dAlf97b] and then show how to handle concurrent probabilistic systems when fairness assumptions about the environment are made.

Probabilistic computation tree logic: Combining several aspects of the logics considered in [HaJo89, Hans91, HaJo94, SeLy94, BidAl95, IyNa96, dAlf97a, dAlf97b] we introduce a logic, called $PCTL^*$, for specifying quantitative properties for probabilistic systems such as "the system terminates with probability at least 0.75" or "the message will be delivered within in next three steps with probability at least $\frac{2}{3}$". $PCTL^*$ can be viewed as the probabilistic counterpart to the logic CTL^* [EmHa86] that combines computation tree logic CTL [ClEm81] and (propositional) linear time logic LTL. As in CTL^*, $PCTL^*$ distinguishes between state and path formulas where the path formulas stand for linear time properties that make statements about the executions (fulpaths) whereas the state formulas express branching time properties that assert something about the possible behaviours in the states. To reason about the possible behaviours in the states, CTL^* uses the quantifiers $\forall$ ("for all executions", also often denoted by the letter A) and $\exists$ ("there exists an execution", also often denoted by the letter E) combined with a path formula φ.[4] In a probabilistic scenario, we also want to reason about the "quantity" of the executions

[1] The basic ideas behind the use of temporal logic as specification formalism for probabilistic systems are sketched in the introduction. See Section 1.1.3 (page 16) and Section 1.2.3, (page 24).

[2] The reason why we recall the results here are twofolds. First, the underlying basic ideas are also used in our algorithm. Second, our symbolic model checker of Chapter 10 make use of them.

[3] In his thesis, Luca deAlfaro [dAlf97a] proposes a different notion of fairness and presents corresponding verification methods. The relation to our approach is also discussed in [dAlf97a].

[4] The CTL^* formula $\forall\varphi$ asserts that the linear time property φ holds for all executions while $\exists\varphi$ states the existence of a computation that fulfills φ.

that satisfy a certain linear time formula φ. For this, *PCTL** replaces the quantifiers $\forall$ and $\exists$ by a probabilistic operator and uses state formulas of the form $\mathsf{Prob}_{\bowtie p}(\varphi)$ rather than $\forall\varphi$ or $\exists\varphi$. Here, the subscript $\bowtie p$ (where $\bowtie$ is a comparison operator, e.g. $\geq$ or $<$) specifies an interval of "acceptable" probabilities. For example, $\mathsf{Prob}_{\geq p}(\varphi)$ asserts that the probability that φ holds is at least p.

PCTL* formulas can be interpreted over the states of a fully probabilistic or concurrent probabilistic system. In the former (fully probabilistic) case, in the *PCTL** state formula $\mathsf{Prob}_{\bowtie p}(\varphi)$, the probabilistic operator refers to the probability measure of the fulpaths where φ holds. In the latter (concurrent) case – where it makes no sense to speak about probabilities of certain events unless the non-determinism is resolved – $\mathsf{Prob}_{\bowtie p}(\varphi)$ is viewed to be correct for a state s if the probability for the fulpaths satisfying φ and starting in s lies in the interval $\{q \in [0,1] : q \bowtie p\}$ under all possible environments (adversaries). The assumptions (e.g. some kind of fairness assumptions) that we make about the environment are formalized by an appropriate type $\mathcal{A}$ of adversaries.[5] For the truth value of the formulas involving the probabilistic operator, we range over all adversaries $A \in \mathcal{A}$; that is, $\mathsf{Prob}_{\bowtie p}(\varphi)$ holds in a state s if, under all adversaries $A \in \mathcal{A}$, the probability measure of the fulpaths starting in s and satisfying φ is $\bowtie p$.

Model checking: The basic idea behind *PCTL** model checking (i.e. computing the set of states where a given state formula holds) is the reduction to the verification of quantitative *LTL* specifications [BidAl95, ASB$^+$95, IyNa96] while the latter can be reduced (via several tricky intermediate steps) to a probabilistic reachability analysis which can be done by solving *linear equation systems* or *linear optimization problems* [CoYa88, CoYa90, HaJo94, BidAl95]. For the reduction of calculating the probabilities for *LTL* formulas to a *probabilistic reachability analysis* (by which we mean the computation of the probabilities to reach a certain set of states), one can use of the *ω-automaton* approach à la Sistla, Vardi & Wolper [WVS83, Vard85, VaWo86] for both fully probabilistic [CoYa95, IyNa96] or concurrent probabilistic [dAlf97a, dAlf97b] systems. Using suitable adaptions of the methods presented in [BidAl95, dAlf97a, dAlf97b], we present a model checking algorithm for *PCTL** with respect to the interpretation over concurrent probabilistic systems where fairness assumptions are made (i.e. where the chosen type $\mathcal{A}$ consists of some kind of fair adversaries). The time complexity of our method is double exponential in the size of the system and linear in the formula. By the results of [CoYa95], this is optimal as it meets the lower bound for verifying concurrent probabilistic systems against (qualitative) linear time properties. The underlying *PCTL* model checker uses similar techniques as the one proposed by [BidAl95] for concurrent probabilistic systems with respect to the standard interpretation that does not involves fairness (and deals with the whole class of adversaries). For the handling of the probabilistic operator (i.e. formulas of the form $\mathsf{Prob}_{\bowtie p}(\varphi)$), techniques are needed to calculate the minimal or maximal probabilities for φ under all adversaries of the chosen type. As in the approach of [BidAl95], the minimal and maximal probabilities for the until operator (i.e. path formulas of the form $\varphi = \Phi_1 \mathcal{U} \Phi_2$) can be computed by solving linear optimization problems. Our *PCTL* model checker for the interpretations with fairness and the one of [BidAl95] for the standard interpretation run in polynomial time.

[5] The standard interpretation where fairness in not taken into account is obtained by $\mathcal{A} = \mathcal{A}dv$ while the use of e.g. $\mathcal{A} = \mathcal{A}dv_{fair}$ leads to an interpretation where fairness is integrated.

$$\Phi \;::=\; tt \;\mid\; a \;\mid\; \Phi_1 \wedge \Phi_2 \;\mid\; \neg\Phi \;\mid\; \mathsf{Prob}_{\bowtie p}(\varphi)$$

$$\varphi \;::=\; \Phi \;\mid\; X\varphi \;\mid\; \varphi_1 \mathcal{U} \varphi_2 \;\mid\; \varphi_1 \mathcal{U}^{\leq k} \varphi_2 \;\mid\; \varphi_1 \wedge \varphi_2 \;\mid\; \neg\varphi$$

Figure 9.1: Syntax of $PCTL^*$

Organization of that chapter: Section 9.1 explains the syntax of $PCTL^*$ (and the sublogics $PCTL$ and LTL) and the interpretations over fully probabilistic and concurrent probabilistic systems. Model checking algorithms for $PCTL^*$, $PCTL$ and LTL are presented Sections 9.2, 9.3 and 9.4 where we briefly sketch how the methods of the literature work and show how to deal with satisfaction relations where fairness is involved. The theoretical foundations of the model checking algorithms for $PCTL$ and LTL are formulated in theorems whose proofs are given in Section 9.5.

The results of this chapter are mainly based on the joint work with Marta Kwiatkowska [BaKw98].[6] In this chapter, we assume some familiarity with temporal logics and ω-automaton and the connection between them. See e.g. the survey papers [Thom90, Thom96, Vard96] for the ω-automaton approach and [Emer90, MaPn92, MaPn95] for temporal logics.

9.1 The logic $PCTL^*$

In this section, we explain the syntax of $PCTL^*$ (and the sublogics $PCTL$ and LTL) and present several semantics for $PCTL^*$. The interpretation over fully probabilistic systems is in the style of [HaJo94, ASB+95, IyNa96]. For the interpretation over concurrent probabilistic systems, we introduce satisfaction relations $\models_{\mathcal{A}}$ that are parametrized by a class $\mathcal{A}$ of adversaries. Intuitively, the chosen type $\mathcal{A}$ of adversaries formalizes the assumptions that are made about the "environment" (the instance that resolves the nondeterministic choices). In the case where $\mathcal{A} = \mathcal{A}dv$ (the collection of all adversaries) we obtain the standard interpretation of [BidAl95].

Syntax of $PCTL^*$: We fix a finite set AP of atomic propositions. $PCTL^*$ *state formulas* (denoted by capital greek letters $\Phi, \Psi, \ldots$) and $PCTL^*$ *path formulas* (denoted by greek letters $\varphi, \psi, \ldots$) over AP are given by the grammar shown in Figure 9.1 (page 207). Here, $a \in AP$, $p \in [0,1]$, $\bowtie \in \{\leq, <, \geq, >\}$ and k is a natural number. The usual derived constants and boolean operators (for both state and path formulas) are $ff = \neg tt$, $\phi_1 \vee \phi_2 = \neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi_1 \rightarrow \varphi_2 = \neg\phi_1 \vee \phi_2$. The path formulas are built from the boolean connectives and the standard temporal operators X, $\mathcal{U}$ and $\mathcal{U}^{\leq k}$. The meanings of the temporal operators X ("next step"), $\mathcal{U}$ ("until") and $\mathcal{U}^{\leq k}$ ("bounded until" or "within the next k steps") are as in the non-probabilistic case. As usual, operators for

[6] When writing down this thesis the author detected that the $PCTL$ model checker of [BaKw98] can be reformulated resulting in a simpler algorithm. This simplification is presented in Section 9.3.

modelling "eventually" $\Diamond$, "sometimes within the next k steps" $\Diamond^{\leq k}$, "always" $\Box$ and "always within the next k steps" $\Box^{\leq k}$ can be derived.

$$\Diamond\varphi \;=\; tt\;\mathcal{U}\;\varphi,\;\; \Diamond^{\leq k}\varphi \;=\; tt\;\mathcal{U}^{\leq k}\;\varphi,\;\; \Box\varphi \;=\; \neg\Diamond\neg\varphi,\;\; \Box^{\leq k}\varphi \;=\; \neg\Diamond^{\leq k}\neg\varphi.$$

For instance, if *error* is an atomic proposition that characterizes all states where a system error has happened then $\mathsf{Prob}_{<0.001}(\Diamond^{\leq 4} error)$ asserts that the probability for a system error within the next four steps is less than 0.001. If $crit_1$ and $crit_2$ are atomic propositions stating that certain processes $\mathcal{P}_1$ or $\mathcal{P}_2$ are in their critical sections then $\mathsf{Prob}_{\geq 1}\,(\Box(\neg crit_1 \vee \neg crit_2))$ stands for the qualitative safety property stating mutual exclusion. The formula $\mathsf{Prob}_{\geq 0.99}\Big(\;\Box\,\Big(request \rightarrow \Diamond^{\leq 5} response\Big)\;\Big)$ stands for the quantitative progress property stating that there is a 99% chance that every request will be answered within the next 5 steps.

Remark 9.1.1 [Quantification over the fulpaths] Formulas of the form $\forall\varphi$ and $\exists\varphi$ (where φ is a path formula and where $\forall$ and $\exists$ are the usual CTL^* path quantifiers that range over all fulpaths) could be added to our logic $PCTL^*$, but we omit them for the sake of simplicity.[7] ■

Semantics of $PCTL^*$: The state formulas are interpreted over the states of a probabilistic system, the paths formulas over the fulpaths. We fix a probabilistic (fully probabilistic or concurrent probabilistic) system $\mathcal{S}$ with state space S and proposition labels in AP. $\mathcal{L}$ denotes the labelling function $S \to 2^{AP}$. Dealing with a fully probabilistic system, the satisfaction relation for the state and path formulas is denoted by $\models$. In the concurrent case, we fix a certain class $\mathcal{A}$ of adversaries and use the symbol $\models_{\mathcal{A}}$ for the induced satisfaction relation. In what follows, we write $\models_*$ to denote the satisfaction relation $\models$ in the fully probabilistic case; in the concurrent case, $\models_*$ denotes $\models_{\mathcal{A}}$ for some adversary type $\mathcal{A} \subseteq \mathcal{A}dv$. The satisfaction relations

$$\models_* \subseteq Path_{ful} \times I\!N \times PathFormulas, \quad \models_* \subseteq\; S \times StateFormulas$$

are defined as follows. As usual, we write $(\pi, j) \models_* \varphi$ rather than $(\pi, j, \varphi) \in \models_*$ and $s \models_* \Phi$ rather than $(s, \Phi) \in \models_*$. The definition of $\models_*$ for the path formulas is as in the non-probabilistic case; see Figure 9.2 (page 209). From this, we obtain

- $(\pi, j) \models_* \Diamond\varphi$ iff there exists an integer l with $j \leq l \leq |\pi|$ and $(\pi, l) \models_* \varphi$
- $(\pi, j) \models_* \Diamond^{\leq k}\varphi$ iff there exists an integer l with $j \leq l \leq \min\{|\pi|, j+k\}$ and $(\pi, l) \models_* \varphi$
- $(\pi, j) \models_* \Box\varphi$ iff $(\pi, l) \models_* \varphi$ for all integers l with $j \leq l \leq |\pi|$
- $(\pi, j) \models_* \Box^{\leq k}\varphi$ iff $(\pi, l) \models_* \varphi$ for all integers l with $j \leq l \leq \min\{|\pi|, k\}$.

For the state formulas, $\models_*$ is defined as follows.

- $s \models_* tt$ for all $s \in S$
- $s \models_* a$ iff $a \in \mathcal{L}(s)$
- $s \models_* \Phi_1 \wedge \Phi_2$ iff $s \models_* \Phi_i$, $i = 1, 2$
- $s \models_* \neg\Phi$ iff $s \not\models \Phi$

[7] When adding state formulas $\forall\varphi$ and $\exists\varphi$ to our syntax, for the semantics of $\forall\varphi$ and $\exists\varphi$ we can either use the standard interpretation (where $\forall\varphi$ asserts that φ holds for all fulpaths) or an interpretation that requires "path fairness". In this case, our model checker of Section 9.3 would have to be extended, e.g. by the method proposed in [CES83] for the standard interpretation or the method of [EmLei85] for checking whether a path formula holds for all (some) fair paths.

$(\pi, j) \models_* \Phi$ iff $j \leq |\pi|$ and $\pi \models \Phi$

$(\pi, j) \models_* \varphi_1 \wedge \varphi_2$ iff $(\pi, j) \models_* \varphi_i$, $i = 1, 2$

$(\pi, j) \models_* \neg\varphi$ iff $(\pi, j) \not\models_* \varphi$

$(\pi, j) \models_* X\varphi$ iff $j < |\pi|$ and $(\pi, j+1) \models_* \varphi$

$(\pi, j) \models_* \varphi_1 \mathcal{U} \varphi_2$ iff there exists an integer l with $j \leq l \leq |\pi|$ and
$(\pi, i) \models_* \varphi_1$, $i = j, j+1, \ldots, l-1$ and $(\pi, l) \models_* \varphi_2$

$(\pi, j) \models_* \varphi_1 \mathcal{U}^{\leq k} \varphi_2$ iff there exists an integer l with $j \leq l \leq \min\{|\pi|, j+k\}$ and
$(\pi, i) \models_* \varphi_1$, $i = j, j+1, \ldots, l-1$ and $(\pi, l) \models_* \varphi_2$

Figure 9.2: The satisfaction relation $\models_*$ for $PCTL^*$ path formulas

The state formula $\Phi = \mathsf{Prob}_{\bowtie p}(\varphi)$ ensures that the probability measure for the fulpaths satisfying φ lies in the interval $I_{\bowtie p} = \{q \in [0,1] : q \bowtie p\}$. Here, the truth value of a path formula interpreted over a fulpath (rather than a pair (π, j)) is given by:

$$\pi \models_* \varphi \quad \text{iff} \quad (\pi, 0) \models_* \varphi.$$

In the fully probabilistic case, satisfaction of $\mathsf{Prob}_{\bowtie p}(\varphi)$ in a state s is derived from the probability measure of $\{\pi \in Path_{ful}(s) : \pi \models \varphi\}$ (see Section 9.1.1). In the concurrent case, satisfaction of $\mathsf{Prob}_{\bowtie p}(\varphi)$ depends on the chosen type $\mathcal{A}$ of adversaries and is defined in terms of the probabilty measures of $\{\pi \in Path^A_{ful}(s) : \pi \models_{\mathcal{A}} \varphi\}$ where A ranges over all adversaries in $\mathcal{A}$ (see Section 9.1.2).[8]

9.1.1 Interpretation over fully probabilistic systems

Let $\mathcal{S} = (S, \mathbf{P}, AP, \mathcal{L})$ be a fully probabilistic system. We define

$$s \models \mathsf{Prob}_{\bowtie p}(\varphi) \quad \text{iff} \quad Prob\,\{\pi \in Path_{ful}(s) : \pi \models \varphi\} \;\bowtie\; p.$$

Notation 9.1.2 [The set $Sat(\Phi)$] *Let* $Sat(\Phi) = \{s \in S : s \models \Phi\}$.

For a fully probabilistic process $\mathcal{P}$, we write $\mathcal{P} \models \Phi$ iff the initial state of $\mathcal{P}$ lies in $Sat(\Phi)$.

Example 9.1.3 [Sock selection problem] We consider the sock selection problem of [GSB94] and the associated fully probabilistic system (see Example 3.3.14 on page 52). The $PCTL^*$ formula $\Phi = \mathsf{Prob}_{\geq 1-1/2^{2n-2}}(\Diamond success)$ states that the probability for the algorithm to terminate in a successful state (i.e. a state where we got a matching pair of socks) is at least $1 - 1/2^{2n-2}$. Then, for the initial state

$$s_{init} = \langle color(sock_1), color(sock_2), 2n-2 \rangle,$$

we have $Prob\,\{\pi \in Path_{ful}(s_{init}) : \pi \models \Diamond success\} = 1 - 1/2^{2n-2}$. Thus, $s_{init} \models \Phi$. ■

[8] It is easy to see that, for each state s and path formula φ, the set $\{\pi \in Path_{ful}(s) : \pi \models \varphi\}$ (or $\{\pi \in Path^A_{ful}(s) : \pi \models_{\mathcal{A}} \varphi\}$ for $A \in \mathcal{A}dv$ in the concurrent case) is measurable.

Example 9.1.4 [Simple communication protocol] For the simple commmunication protocol of Example 1.2.1 (page 19) equipped with the atomic propositions *init* and *wait* and the labelling function $\mathcal{L}$ with $a \in \mathcal{L}(s_*)$ iff $a = *$ we have

$$s_{init} \models \mathsf{Prob}_{\geq 0.9999}(\Diamond^{\leq 4} wait)$$

which asserts that, with probability at least 0.9999, if the sender is in its initial state (where it produces a message), then the message will be eventually delivered within the next four steps. This can be seen as follows. Let $p_s(\varphi)$ denote the probability measure of all fulpaths $\pi \in Path_{ful}(s)$ where φ holds. Then, we have

$$\begin{array}{rcl} p_{s_{init}}(\Diamond^{\leq 4} wait) & = & p_{s_{del}}(\Diamond^{\leq 3} wait) \\ & = & \frac{99}{100} + \frac{1}{100} \cdot p_{s_{lost}}(\Diamond^{\leq 2} wait) \\ & = & \frac{99}{100} + \frac{1}{100} \cdot p_{s_{del}}(\Diamond^{\leq 1} wait) \\ & = & \frac{99}{100} + \frac{1}{100} \cdot \frac{99}{100} = \frac{9999}{10000}. \end{array}$$

Here, we use the fact that $p_s(\Diamond^{\leq k} a) = 1$ if $a \in \mathcal{L}(s)$ and, for $a \notin \mathcal{L}(s)$,

$$p_s(\Diamond^{\leq k+1} a) = \sum_{t \in S} \mathbf{P}(s,t) \cdot p_t(\Diamond^{\leq k} a)$$

and $p_s(\Diamond^{\leq 0} a) = 0$. (See Section 9.3, page 217). ■

9.1.2 Interpretation over concurrent probabilistic systems

If $\mathcal{S} = (S, Steps, AP, \mathcal{L})$ os a concurrent probabilistic system and $\mathcal{A} \subseteq \mathcal{A}dv$ then we define

$$s \models_{\mathcal{A}} \mathsf{Prob}_{\bowtie p}(\varphi) \text{ iff } Prob\left\{\pi \in Path^{\mathcal{A}}_{ful}(s) : \pi \models_{\mathcal{A}} \varphi\right\} \bowtie p \text{ for all } A \in \mathcal{A}.$$

Notation 9.1.5 [The set $Sat_{\mathcal{A}}(\Phi)$] *Let* $Sat_{\mathcal{A}}(\Phi) = \{s \in S : s \models_{\mathcal{A}} \Phi\}$.

For a concurrent probabilistic process $\mathcal{P}$, we write $\mathcal{P} \models_{\mathcal{A}} \Phi$ iff the initial state of $\mathcal{P}$ belongs to $Sat_{\mathcal{A}}(\Phi)$. In the remainder of that thesis, we shrink our attention to the following four classes of adversaries:

- $\mathcal{A}dv$ (the set of all adversaries)
- $\mathcal{A}dv_{fair}$ (the set of fair adversaries in the sense of Definition 3.2.17, page 46),
- $\mathcal{A}dv_{sfair}$ (the set of strictly fair adversaries in the sense of Definition 3.2.17, page 46),
- $\mathcal{A}dv_{Wfair}$ (the set of W-fair adversaries in the sense of Definition 3.2.20, page 47).

The satisfaction relation $\models_{\mathcal{A}dv}$ yields the standard interpretation à la [BidAl95]. We briefly write $\models$ instead of $\models_{\mathcal{A}dv}$. For the satisfaction relations $\models_{\mathcal{A}dv_{fair}}$, $\models_{\mathcal{A}dv_{sfair}}$ and $\models_{\mathcal{A}dv_{Wfair}}$, we also write $\models_{fair}$, $\models_{sfair}$ and $\models_{Wfair}$. Similarly, we often write $Sat(\Phi)$, $Sat_{fair}(\Phi)$, $Sat_{sfair}(\Phi)$ and $Sat_{Wfair}(\Phi)$ rather than $Sat_{\mathcal{A}dv}(\Phi)$, $Sat_{\mathcal{A}dv_{fair}}(\Phi)$, $Sat_{\mathcal{A}dv_{sfair}}(\Phi)$ and $Sat_{\mathcal{A}dv_{Wfair}}(\Phi)$ respectively.

The following example demonstrates that the satisfaction relation in the concurrent case depends on the chosen $\mathcal{A}$. In particular, the below example shows that – as in the non-probabilistic case – fairness assumptions (with respect to the non-deterministic choices) might be essential for establishing certain (quantitative or qualitative) properties.

Example 9.1.6 [Roulette player] We consider the roulette player of Example 1.2.3 on page 22 (see Figure 1.3 on page 22). We use the atomic propositions *play*, *happy* and *won* and the labelling function $\mathcal{L}$ where $a \in \mathcal{L}(s_*)$ iff $a = *$. First, we regard the formula

$$\varphi \;=\; \Box\,(play \rightarrow \Diamond won)\,.$$

with respect to the standard satisfaction relation $\models$ where we range over all adversaries. For each adversary A, $Prob\left\{\pi \in Path^A_{ful}(s_{init}) : \pi \models \varphi\right\} \;=\; 1$. Thus, $s_{init} \;\models\; \mathsf{Prob}_{\geq 1}(\varphi)$ which ensures that – independent on the environment (adversary) – whenever the roulette player starts playing then he will eventually win a game with probability 1. Next we regard the $PCTL^*$ state formula

$$\Psi \;=\; \mathsf{Prob}_{\geq 0.5}(\psi) \text{ where } \psi \;=\; \Diamond happy.$$

Intuitively, Ψ states that, there is at least a 50% chance for the roulette player to leave the casino while winning the last game. The truth value of the formula Ψ depends on the environment (the chosen adversary type $\mathcal{A}$). Let $p^A_s(\psi)$ be the probability measure of $\left\{\pi \in Path^A_{ful}(s) : \pi \models \psi\right\}$. For the simple adversary A with $A(s_{won}) = \mu^1_{s_{play}}$ we have $p^A_{s_{init}}(\psi) = 0$ because A forces the roulette player to stay forever in the casino. Thus,

$$s_{init} \;\not\models\; \Psi$$

when we deal with the standard satisfaction relation $\models$ that does not involve fairness. Dealing with a satisfaction relation where fairness in the state s_{won} is assumed, the formula Ψ holds in the initial state. This is because A behave unfair in the state s_{won} and, for each other adversary B, we have $p^B_{s_{init}}(\psi) = 1/2$ (cf. Example 3.2.13 on page 44). Hence,

$$s_{init} \;\models_{fair}\; \Psi, \quad s_{init} \;\models_{sfair}\; \Psi, \quad s_{init} \;\models_{Wfair}\; \Psi$$

provided that W contains s_{won}. When we use a set W that does not contain s_{won}, then the above adversary A is W-fair which yields $s_{init} \;\not\models_{Wfair}\; \Psi$. Thus, the quantitative property Ψ cannot be established unless appropriate fairness assumptions are made. ■

Remark 9.1.7 [The CTL^* quantifiers $\forall$ and $\exists$] As in [Hans91, SeLy94, BaKw98], for the use of $PCTL^*$ as specification language for concurrent systems, instead of the probabilistic operator $\mathsf{Prob}_{\bowtie p}(\varphi)$ we might use state formulas of the form $[\forall\varphi]_{\sqsupseteq p}$ and $[\exists\varphi]_{\sqsupseteq p}$.[9] Then, $[\forall\varphi]_{\sqsupseteq p}$ states that *under all adversaries* (of the chosen type) the probability for φ is $\sqsupseteq p$ which corresponds to the meaning of $\mathsf{Prob}_{\sqsupseteq p}(\varphi)$. $PCTL^*$ state formulas with an upper bound for the probabilities (i.e. formulas of the form $\mathsf{Prob}_{\sqsubseteq p}(\varphi)$) can be expressed either by $[\forall\neg\varphi]_{\sqsupseteq 1-p}$ (which is equivalent to $\mathsf{Prob}_{\sqsupseteq 1-p}(\neg\varphi)$) or with the help of existential quantification. For instance, $\mathsf{Prob}_{\leq p}(\varphi)$ corresponds to $\neg[\exists\varphi]_{>p}$. ■

9.1.3 The sublogics $PCTL$ and LTL

$PCTL^*$ is a combination of $PCTL$ (probabilistic computation tree logic) and LTL (propositional linear time logic). In $PCTL$, arbitrary combinations of state formulas are possible

[9] Here, we use $\sqsupseteq$ to denote one of the comparison operators $\geq$ or $>$. As in CTL^*, the quantifiers $\forall$ and $\exists$ range over all possible resolutions of the non-deterministic choices yielding *executions* in the non-probabilistic case and *execution trees* in the probabilistic case.

but only path formulas of the form $X\Phi$, $\Phi_1 \mathcal{U} \Phi_2$ and $\Phi_1 \mathcal{U}^{\leq k} \Phi_2$ (where Φ, Φ_1 and Φ_2 are state formulas) are allowed. Linear time logic *LTL* is the other "extreme" fragment of *PCTL** where arbitrary combinations of path formulas but only propositional state formulas are allowed.

Probabilistic computation tree logic *PCTL*: In *PCTL*, only "simple" path formulas built from the temporal operators X, $\mathcal{U}^{\leq k}$ or $\mathcal{U}$ and state formulas are allowed. Formally, *PCTL* is those sublogic of *PCTL** whose state and path formulas are built from the production system shown in Figure 9.3 (page 212). In what follows, we briefly speak

$$\Phi \ ::= \ tt \ \Big| \ a \ \Big| \ \Phi_1 \wedge \Phi_2 \ \Big| \ \neg\Phi \ \Big| \ \mathsf{Prob}_{\bowtie p}(\varphi)$$

$$\varphi \ ::= \ X\Phi \ \Big| \ \Phi_1 \mathcal{U} \Phi_2 \ \Big| \ \Phi_1 \mathcal{U}^{\leq k} \Phi_2$$

Figure 9.3: Syntax of *PCTL*

about *PCTL* formulas rather than *PCTL* state formulas. In *PCTL*, the temporal operators "eventually" or "sometimes within the next k steps" are obtained as in the case of *PCTL**:

$$\mathsf{Prob}_{\bowtie p}(\Diamond \Phi) \ = \ \mathsf{Prob}_{\bowtie p}(tt \ \mathcal{U} \ \Phi), \quad \mathsf{Prob}_{\bowtie p}(\Diamond^{\leq k} \Phi) \ = \ \mathsf{Prob}_{\bowtie p}(tt \ \mathcal{U}^{\leq k} \ \Phi).$$

For modelling the temporal operators "always" and "always within the next k steps" in *PCTL*, we use the fact that, for any *PCTL** path formula φ, the formulas $\mathsf{Prob}_{\bowtie p}(\varphi)$ and $\mathsf{Prob}_{\overline{\bowtie} 1-p}(\neg\varphi)$ are equivalent where $\overline{\leq} = \geq$, $\overline{<} = >$, $\overline{\geq} = \leq$ and $\overline{>} = <$.[10] Thus, "always" and "always within the next k steps" can be obtained in *PCTL* by:

$$\mathsf{Prob}_{\bowtie p}(\Box \Phi) \ = \ \mathsf{Prob}_{\overline{\bowtie} 1-p}(\Diamond \neg \Phi), \quad \mathsf{Prob}_{\bowtie p}(\Box^{\leq k} \Phi) \ = \ \mathsf{Prob}_{\overline{\bowtie} 1-p}(\Diamond^{\leq k} \neg \Phi)$$

Note that – because of the simplicity of the *PCTL* path formulas – the satisfaction relation $\models_*$ (where $\models_*$ stands for $\models$ in the fully probabilistic case and for $\models_{\mathcal{A}}$ in the concurrent case) for *PCTL* path formulas is given by:

$\pi \models_* X\Phi$ iff $|\pi| \geq 1$ and $\pi(1) \models_* \Phi$

$\pi \models_* \Phi_1 \mathcal{U} \Phi_2$ iff there exists an integer l with $0 \leq l \leq |\pi|$ and
$$\pi(i) \models_* \Phi_1, \ i = 0, 1, \ldots, l-1 \text{ and } \pi(l) \models_* \Phi_2$$

$\pi \models_* \Phi_1 \mathcal{U}^{\leq k} \Phi_2$ iff there exists an integer l with $0 \leq l \leq \min\{|\pi|, k\}$ and
$$\pi(i) \models_* \Phi_1, \ i = 0, 1, \ldots, l-1 \text{ and } \pi(l) \models_* \Phi_2.$$

Linear time logic *LTL*: In *LTL*, the probabilistic operator $\mathsf{Prob}_{\bowtie p}$ is removed. Thus, *LTL* formulas are path formulas built from atomic propositions, the boolean connectives and the temporal operators. Formally, *LTL* formulas are those *PCTL** path formulas that are built from the grammar shown in Figure 9.4 (page 213).

LTL itself can serve as specification formalism for probabilistic systems. In that case, a specification consists of a *LTL* formula and a lower or upper bound for the "acceptable" probabilities.

[10] Equivalence means semantic equality; that is, for any probabilistic system $\mathcal{S}$ and any state s of $\mathcal{S}$, the first formula holds in s if and only if the second formula holds in s.

$$\varphi ::= \quad tt \;\Big|\; a \;\Big|\; \neg\varphi \;\Big|\; \varphi_1 \wedge \varphi_2 \;\Big|\; X\varphi \;\Big|\; \varphi_1 \mathcal{U} \varphi_2 \;\Big|\; \varphi_1 \mathcal{U}^{\leq k} \varphi_2$$

Figure 9.4: Syntax of *LTL*

Notation 9.1.8 [Quantitative *LTL* specifications] *A* quantitative *LTL* specification *is a pair* $\langle \varphi, I \rangle$ *consisting of a LTL formula* φ *and an interval* I *of the form* $[0,p]$, $[0,p[$, $[p,1]$ *or* $]p,1]$ *for some* $p \in [0,1]$.

A state s of a fully probabilistic probabilistic system is viewed to be correct with respect to a *LTL* specification $\langle \varphi, I \rangle$ if the "truth value" of that formula φ lies in the interval I of "acceptable" probabilities. Here, the "truth value" is given by the probability measure of all fulpaths starting in s and satisfying φ.

Notation 9.1.9 [The set $Sat(\langle \varphi, I \rangle)$ (fully probabilistic case)] *Let* $Sat(\langle \varphi, I \rangle)$ *be the set of states* $s \in S$ *where* $Prob\,\{\pi \in Path_{ful}(s) : \pi \models \varphi\} \;\in\; I$.

In the concurrent case, a *LTL* specification $\langle \varphi, I \rangle$ asserts that, for any possible environment, the probability for φ lies in the interval I. As before, the possible environments are formalized by an adversary type $\mathcal{A} \subseteq \mathcal{A}dv$.

Notation 9.1.10 [The sets $Sat_{\mathcal{A}}(\langle \varphi, I \rangle)$ (concurrent case)] *Let* $Sat_{\mathcal{A}}(\langle \varphi, I \rangle)$ *be the set of states* $s \in S$ *such that, for all* $A \in \mathcal{A}$,

$$Prob\left\{\pi \in Path^{A}_{ful}(s) : \pi \models_{\mathcal{A}} \varphi\right\} \;\in\; I.$$

For $\mathcal{A} \in \{\mathcal{A}dv, \mathcal{A}dv_{fair}, \mathcal{A}dv_{sfair}, \mathcal{A}dv_{Wfair}\}$, we often write $Sat(\langle \varphi, I \rangle)$, $Sat_{fair}(\langle \varphi, I \rangle)$, $Sat_{sfair}(\langle \varphi, I \rangle)$ or $Sat_{Wfair}(\langle \varphi, I \rangle)$ rather than $Sat_{\mathcal{A}}(\langle \varphi, I \rangle)$.

Remark 9.1.11 [Qualitative *LTL* specifications] Various authors, for example [Vard85, VaWo86, CoYa95, PnZu93], deal with *LTL* (or similar logics) as a formalism for specifying *qualitative* (linear time) properties that assert that a certain *LTL* formula φ holds for almost all fulpaths (resp. almost all fulpaths of an adversary of the chosen type $\mathcal{A}$). Such qualitative properties can be formalized by quantitive *LTL* specifications of the form $\langle \varphi, [1,1] \rangle$. ■

Clearly, in the fully probabilistic and concurrent cases, we have the equivalence of the quantitative *LTL* specification $\langle \varphi, I_{\bowtie p} \rangle$ and the *PCTL** state formula $\mathsf{Prob}_{\bowtie p}(\varphi)$ in the sense that $Sat_*(\mathsf{Prob}_{\bowtie p}(\varphi)) \;=\; Sat_*(\langle \varphi, I \rangle)$ where $I_{\bowtie p} = \{q \in [0,1] : q \bowtie p\}$.

9.1.4 Related logics

We mentioned before that our logic *PCTL** is based on existing logics proposed in the literature. We briefly sketch the connections and differences.

Fully probabilistic case: Our logic $PCTL$ agrees with the logic (also called $PCTL$) introduced by Hansson & Jonsson [HaJo94]; the full logic $PCTL^*$ with the logic considered by Aziz et al [ASB+95] (and later considered e.g. by Iyer & Narasimha [IyNa96]).[11]

Concurrent case: Dealing with the standard interpretation $\models$ our logic $PCTL^*$ (resp. the sublogic $PCTL$) essentially agrees with the logics considered in [HaJo89, Hans91, SeLy94, BidAl95, dAlf97a, dAlf97b]. The main difference between our logic $PCTL$ and the logic (also called $PCTL$) of Hansson [Hans91] (and later considered by Segala & Lynch [SeLy94]) is that the latter deals with action labels while we label the states with atomic propositions.[12] The logic $pCTL^*$ of Bianco & deAlfaro [BidAl95] agrees with our logic $PCTL^*$. Luca de Alfaro [dAlf97a, dAlf97b] uses an extension of $pCTL^*$ that contains an operator to express bounds on the average time between events which does not have a counterpart in $PCTL^*$.[13]

9.1.5 $PCTL^*$ equivalence and bisimulation equivalence

As shown in [ASB+95], for fully probabilistic systems, $PCTL^*$ equivalence (and also $PCTL$ equivalence) is the same as bisimulation equivalence. The connection between $PCTL$ equivalence and bisimulation equivalence for action-labelled concurrent systems is discussed in [SeLy94]. We conjecture that these results carry over to the proposition-labelled case and claim that, for concurrent probabilistic systems, bisimulation equivalence implies $PCTL$ equivalence (with respect to the standard interpretation $\models$) while the converse does not hold.[14] To see why (in the concurrent case) $PCTL$ equivalence (or even $PCTL^*$ equivalence) does not imply bisimulation equivalence consider the system shown in Figure 9.5 (page 215). The states s and s' are $PCTL^*$ equivalent but not bisimulation equivalent.[15] Note that this stands in contrast to the non-probabilistic case where CTL^* equivalence and bisimulation equivalence coincide [BCG88].

9.2 Model checking algorithms for $PCTL^*$

In this section, we consider model checking algorithms for $PCTL^*$ and the sublogics $PCTL$ and LTL. A model checking algorithm for $PCTL^*$ means a method that takes as its input a $PCTL^*$ state formula Ψ over a certain set AP of atomic propositions

[11] The only difference is that [ASB+95, IyNa96] do not use the bounded until operator $\mathcal{U}^{\leq k}$. Moreover, [ASB+95] extends the interpretation to the states of a *generalized Markov chain* (which can be viewed as a fully probabilistic system with intervals of transition probabilities).

[12] Moreover, there are some minor differences. [Hans91, SeLy94] avoid the (explicit) use of the probabilistic operator $\mathsf{Prob}_{\bowtie}$ and deal with state formulas of the form $[\forall\varphi]_{\sqsupseteq p}$ and $[\exists\varphi]_{\sqsupseteq p}$ as explained in Remark 9.1.7 (page 211). [Hans91] mainly concentrates on the specification of *soft deadlines*. For these, the unbounded until operator $\mathcal{U}$ is not needed. However, unbounded until $\mathcal{U}$ could be added as well.

[13] More minor differences between $pCTL^*$ and $PCTL^*$ are that $PCTL^*$ contains the next step operator X and the bounded until operator $\mathcal{U}^{\leq k}$, whereas $pCTL^*$ does not (but these operators could easily be added). Vice versa, $pCTL^*$ contains the usual CTL^* quantifiers $\forall$ and $\exists$ (denoted A and E in the approach by Bianco & de Alfaro) that range over all paths: $\forall$ meaning "for all fulpaths" and $\exists$ "there is a fulpath".

[14] For this, we assume a suitable adaption of the definition of bisimulation equivalence for the proposition-labelled concurrent case (in the style of [JoLa91, ASB+95]).

[15] Here, we assume a labelling function $\mathcal{L}$ with $\mathcal{L}(s) = \mathcal{L}(s')$.

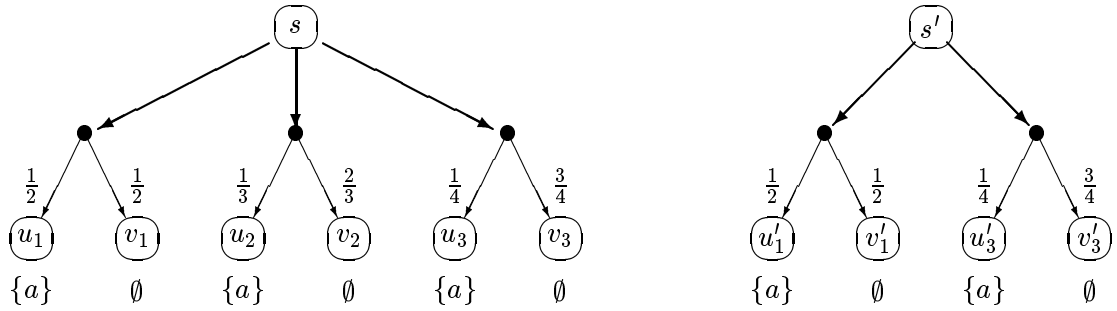


Figure 9.5: s and s' are $PCTL^*$ equivalent but $s \not\sim s'$

and a finite (fully or concurrent) probabilistic system $\mathcal{S}$ with proposition labels in AP and computes the set $Sat_*(\Psi)$ of states s in $\mathcal{S}$ where Ψ holds.[16] Similarly, $PCTL$ (or LTL) model checking means a procedure to compute $Sat_*(\Psi)$ (or $Sat_*(\langle \varphi, I \rangle)$) for a given finite probabilistic system and $PCTL$ formula Ψ (or quantitative LTL specification $\langle \varphi, I \rangle$). Clearly, any $PCTL^*$ model checking algorithm subsumes $PCTL$ and LTL model checking algorithms and yields an automatic procedure to verify probabilistic processes against quantitative temporal logical specifications, provided that the process can be described by a finite system $\mathcal{S}$ with initial state s_{init} and that the specification can be expressed by a $PCTL^*$ state formula Ψ. Model checking algorithms for $PCTL$, LTL and $PCTL^*$ are presented for fully probabilistic systems in [CoYa88, HaJo94, ASB+95, CoYa95, IyNa96] and concurrent probabilistic systems with respect to the standard satisfaction relation $\models$ in [BidAl95, dAlf97a, dAlf97b]. These methods are based on the following common ideas.

(1) The $PCTL^*$ model checking algorithm is based on a recursive procedure that successively computes the sets $Sat_*(\Phi)$ for all state subformulas Φ of the given $PCTL^*$ formula Ψ. For the handling of subformulas of the form $\Phi = \mathsf{Prob}_{\bowtie p}(\varphi)$, the $PCTL^*$ path formula φ is translated into a LTL formula φ' such that $Sat_*(\Phi)$ can be derived from $Sat_*(\langle \varphi', I_{\bowtie p} \rangle)$ where the latter is computed with a model checking algorithm for LTL.

(2) The method for LTL uses the $PCTL$ model checker for the handling of the until operator. For this, the underlying LTL formula $\varphi_1 \mathcal{U} \varphi_2$ is replaced by a $PCTL$ path formula of the form $a \mathcal{U} b$ for atomic propositions a and b, the system $\mathcal{S}$ by a more complex system $\mathcal{S}'$.

Thus, $PCTL^*$ model checking can be reduced to LTL model checking; LTL model checking to $PCTL$ model checking. It is worth noting that these reductions can be seen as the probabilistic counterparts to the results (for non-probabilistic systems) by Emerson & Lei [EmLei85] (where it is shown that any model checking algorithm for LTL can be modified for a CTL^* model checking algorithm with the same complexity) and Clarke, Grumberg & Hamaguchi [CGH94] (where it is shown that LTL model checking can be reduced to CTL model checking with fairness assumptions).

We now explain how a given model checking algorithm for LTL can be applied to obtain a $PCTL^*$ model checking algorithm. This method goes back to Bianco & de Alfaro [BidAl95] where concurrent systems and the satisfaction relation $\models$ are considered. It is

[16] As before, $Sat_*(\Psi)$ denotes $Sat(\Psi)$ for fully probabilistic systems. In the concurrent case, we assume a fixed class $\mathcal{A}$ of adversaries and deal with $Sat_*(\Psi) = Sat_{\mathcal{A}}(\Psi)$.

also applicable for fully probabilistic systems or concurrent systems with other satisfaction relations, e.g. $\models_{fair}$, $\models_{sfair}$ or $\models_{Wfair}$. Similar ideas are used in [ASB+95, IyNa96] that consider $PCTL^*$ with the interpretation over fully probabilistic systems.

Model checking for $PCTL^*$: The input is a $PCTL^*$ state formula Ψ over AP and a finite probabilistic system with state space S and labelling function $\mathcal{L} : S \to 2^{AP}$. The algorithm is based on a recursive procedure that successively computes the sets $Sat_*(\Phi)$ for all state subformulas Φ of Ψ. The cases where Φ is tt, an atomic proposition or of the form $\neg\Phi'$ or $\Phi_1 \wedge \Phi_2$ are clear since we have

$$Sat_*(tt) = S, \qquad Sat_*(\neg\Phi') = S \setminus Sat_*(\Phi'),$$
$$Sat_*(a) = \{s \in S : a \in \mathcal{L}(s)\}, \qquad Sat_*(\Phi_1 \wedge \Phi_2) = Sat_*(\Phi_1) \cap Sat_*(\Phi_2).$$

The interesting case is where the outermost operator of Φ is the probabilistic operator $\mathsf{Prob}_{\bowtie p}$. For this, we apply a model checking algorithm for LTL. Let $\Phi = \mathsf{Prob}_{\bowtie p}(\varphi)$ and let $\Phi_1, \ldots, \Phi_k$ be the maximal state subformulas of φ. We apply the described method recursively to $\Phi_1, \ldots, \Phi_k$ and obtain the sets $Sat_*(\Phi_i)$, $i = \ldots, k$. Then, we replace the subformulas $\Phi_1, \ldots, \Phi_k$ by "fresh" atomic propositions $a_1, \ldots, a_k$ and extend the labelling function $\mathcal{L}$ by inserting a_i into $\mathcal{L}(s)$ iff $s \in Sat_*(\Phi_i)$. The so obtained path formula φ' is a LTL formula over $AP \cup \{a_1, \ldots, a_k\}$. Thus, we may apply the given LTL model checking algorithm to the LTL specification $\langle \varphi', I_{\bowtie p} \rangle$ and obtain $Sat_*(\Phi) = Sat_*(\langle \varphi, I_{\bowtie p} \rangle)$ where $I_{\bowtie p} = \{q \in [0,1] : q \bowtie p\}$.

In the next two section, we consider model checking algorithms for $PCTL$ (Section 9.3) and LTL (Section 9.4). As described above, the method for LTL can be modified for a $PCTL^*$ model checker. We briefly recall the results of the literature (where the fully probabilistic case and the concurrent case with the standard interpretation $\models$ are considered) and present methods to deal with an interpretation that assumes fairness with respect to the non-deterministic choices. More precisely, we deal with the satisfaction relations $\models_{fair}$, $\models_{sfair}$ and $\models_{Wfair}$ that range over all fair, strictly fair and W-fair adversaries respectively. We will see that the result of [dAlf97b] stating that $PCTL^*$ model checking with respect to $\models$ can be done in time polynomial in the size of the system and double exponential in the size of the formula carries over to the above satisfaction relations where fairness is involved. Thus (by the results of the following two sections):

Theorem 9.2.1 *Let Ψ a $PCTL^*$ formula, $\mathcal{S}$ a finite concurrent probabilistic system and W a subset of the state space of $\mathcal{S}$. Then, $Sat_{fair}(\Psi)$, $Sat_{sfair}(\Psi)$ and $Sat_{Wfair}(\Psi)$ can be computed in time polynomial in the size of $\mathcal{S}$ and double exponential in the size of Ψ.*

By the results of [CoYa95], this time complexity is optimal. In the fully probabilistic case, the LTL model checking algorithm of [CoYa88, CoYa95] yields a $PCTL^*$ model checking that runs in time polynomial in the size of the system and single exponential in the size of the formula. An alternative algorithm with the same time complexity (based on the ω-automaton approach) is presented in [IyNa96].

9.3 Model checking for $PCTL$

Model checking algorithms for $PCTL$ are presented by [HaJo94] for fully probabilistic systems and by [BidAl95] for concurrent probabilistic systems with respect to the stan-

dard satisfaction relation $\models$. Both algorithms are based on a recursive procedure that successively computes the sets $Sat(\Phi)$ for all subformulas Φ of the given formula Ψ. For the handling of the until operator, [HaJo94] uses linear equation systems, [BidAl95] linear optimization problems (cf. Remark 3.1.8, page 36, and Remark 3.2.12, page 43). In both cases, the time complexity is polynomial in the size of the system and linear in the size of the formula.

In this section, we briefly sketch the methods of [HaJo94, BidAl95] and present model checking algorithms for *PCTL* with respect to the satisfaction relations $\models_{fair}$, $\models_{sfair}$ and $\models_{Wfair}$. As before, we write $Sat_*(\Phi)$ to denote the set $Sat(\Phi)$ in the fully probabilistic case and $Sat_{\mathcal{A}}(\Phi)$ in the concurrent case (where $\mathcal{A}$ is the chosen type of adversaries).

The *main procedure* is the same for fully probabilistic and concurrent probabilistic systems and uses the ideas of the model checking algorithm for *CTL* à la Clarke, Emerson & Sistla [CES83]. The starting point is a *PCTL* formula Ψ over AP and a finite probabilistic system $\mathcal{S}$ with state space S and a labelling function $\mathcal{L} : S \to 2^{AP}$. First, it builds the *parse tree* of Ψ whose nodes stand for subformulas of Ψ. The root represents the formula Ψ. The leaves are labelled by the boolean constant tt or an atomic proposition. The internal nodes are labelled by one of the operators $\wedge$, $\neg$, $\mathsf{Prob}_{\bowtie p}(X\_)$ $\mathsf{Prob}_{\bowtie p}(\mathcal{U}\_)$ or $\mathsf{Prob}_{\bowtie p}(\mathcal{U}^{\leq k}\_)$. Nodes labelled by $\neg$ or a next-step operator $\mathsf{Prob}_{\bowtie p}(X\_)$ have exactly one son, representing the argument of the negation, resp. next step operator, in the corresponding subformula. Nodes labelled by $\wedge$ or an until operator $\mathsf{Prob}_{\bowtie p}(\mathcal{U}\_)$ or $\mathsf{Prob}_{\bowtie p}(\mathcal{U}^{\leq k}\_)$ have exactly two sons (their arguments). If v is a node then let Φ_v denote the formula represented by v. In a *bottom-up* manner, we calculate the sets $Sat_*(\Phi_v)$ of states where the corresponding subformula Φ_v holds. For the handling of the leaves (nodes where the corresponding formula is tt or an atomic proposition) we use the fact that $Sat_*(tt) = S$ and $Sat_*(a) = \{s \in S : a \in \mathcal{L}(s)\}$. For the computation of Φ_v for an internal node v, we might assume that the sets $Sat_*(\Phi_w)$ for the sons w of v are already computed. Thus, we can treat the proper state subformulas of Φ_v as atomic propositions. The cases where the outermost operator of Φ_v is one of the boolean connectives $\neg$ or $\wedge$ is clear as we have: $Sat_*(\neg\Phi) = S \setminus Sat_*(\Phi)$ and $Sat_*(\Phi_1 \wedge \Phi_2) = Sat_*(\Phi_1) \cap Sat_*(\Phi_2)$. Now we consider the case where Φ_v is of the form $\mathsf{Prob}_{\bowtie p}(\varphi)$.

Fully probabilistic case: We briefly recall the results of Hansson & Jonsson [HaJo94] for the fully probabilistic case. As before, let $\mathbf{P} : S \times S \to [0,1]$ denote the transition probability matrix in $\mathcal{S}$ (i.e. $\mathcal{S} = (S, \mathbf{P}, AP, \mathcal{L})$). We compute

$$p_s(\varphi) \;=\; Prob\{\pi \in Path_{ful}(s) : \pi \models \varphi\}$$

for all state $s \in S$ and then put $Sat(\Phi_v) = \{s \in S : p_s(\varphi) \bowtie p\}$. The probabilities $p_s(\varphi)$ can be computed as follows. The handling with the next step operator is based on the observation that

$$p_s(X\Phi) \;=\; \mathbf{P}(s, Sat(\Phi)) \;=\; \sum_{t \in Sat(\Phi)} \mathbf{P}(s,t).$$

For the computation of $p_s(\Phi_1 \mathcal{U}^{\leq k} \Phi_2)$, [HaJo94] proposes two methods. One uses iterative matrix multiplication; the other is based on the fact that

$p_s(\Phi_1 \mathcal{U}^{\leq k} \Phi_2) = 1$ if $s \in Sat(\Phi_2)$,
$p_s(\Phi_1 \mathcal{U}^{\leq k} \Phi_2) = 0$ if $s \in S \setminus (Sat(\Phi_1) \cup Sat(\Phi_2))$,

$p_s(\Phi_1 \mathcal{U}^{\leq 0} \Phi_2) = 0$ if $s \in Sat(\Phi_1) \setminus Sat(\Phi_2)$

and, for $s \in Sat(\Phi_1) \setminus Sat(\Phi_2)$ and $k \geq 1$,

$$p_s(\Phi_1 \mathcal{U}^{\leq k} \Phi_2) \;=\; \sum_{t \in S} \mathbf{P}(s,t) \cdot p_t(\Phi_1 \mathcal{U}^{\leq k-1} \Phi_2).$$

For the until operator, the probabilities $p_s(\Phi_1 \mathcal{U} \Phi_2)$ can be obtained by solving a regular linear equation system (preceeded by a graph analysis which yields the set $\{s \in S : p_s(\Phi_1 \mathcal{U} \Phi_2) = 0\}$). Alternatively, one can use an iterative method that computes an approximation of the function $s \mapsto p_s(\Phi_1 \mathcal{U} \Phi_2)$ (viewed as the least fixed point of an operator on the function space $S \to [0,1]$).[17] See Remark 3.1.8 (page 36).

Concurrent case: In the remainder of that section, we deal with the concurrent case and show how to compute the set $Sat_{\mathcal{A}}(\mathsf{Prob}_{\bowtie p}(\varphi))$ where $\mathcal{A}$ is one of the adversary types $\mathcal{A}dv$, $\mathcal{A}dv_{fair}$, $\mathcal{A}dv_{sfair}$ or $\mathcal{A}dv_{Wfair}$. The method for $\mathcal{A} = \mathcal{A}dv$ is those of [BidAl95].

As before, let $\mathcal{S} = (S, Steps, AP, \mathcal{L})$ be the underlying system. Recall that we assume $\mathcal{S}$ to be finite, i.e. the state space S is finite and, for any state s, the set $Steps(s)$ is finite. For the satisfaction relation $\models_{Wfair}$, we deal with a fixed subset W of S. We consider the cases $\varphi = X\Phi$, $\varphi = \Phi_1 \mathcal{U} \Phi_2$ and $\varphi = \Phi_1 \mathcal{U}^{\leq k} \Phi_2$ and present criterias for $s \models_{\mathcal{A}} \mathsf{Prob}_{\bowtie p}(\varphi)$ by means of the sets $Sat_{\mathcal{A}}(\Phi)$ or $Sat_{\mathcal{A}}(\Phi_i)$, $i = 1, 2$. Since we assume that the sets $Sat_{\mathcal{A}}(\Phi)$, $Sat_{\mathcal{A}}(\Phi_i)$, $i = 1, 2$, are already computed these criterias yield a method for computing $Sat_{\mathcal{A}}(\mathsf{Prob}_{\bowtie p}(\varphi))$.

Notation 9.3.1 [The comparison operators $\sqsupseteq$ and $\sqsubseteq$] *We write $\sqsupseteq$ to denote one of the comparison operators $\geq$ or $>$. Similarly, $\sqsubseteq$ stands for $\leq$ or $<$.*

Clearly, for formulas of the form $\mathsf{Prob}_{\sqsupseteq p}(\varphi)$ we need the "minimal" probabilities for φ under all adversaries $A \in \mathcal{A}$, while the constraint $\sqsubseteq p$ requires to look for the "maximal" probabilities under all $A \in \mathcal{A}$.

The next-step and the bounded-until operator are dealt with in the same way for all four interpretations whereas the unbounded until operator requires special methods for the satisfaction relations with fairness.[18]

9.3.1 Next step

The following lemma shows that the set of states where $\mathsf{Prob}_{\bowtie p}(X\Phi)$ holds is obtained by computing the values $\mu\,[Sat_{\mathcal{A}}(\Phi)] \;=\; \sum_{t \in Sat_{\mathcal{A}}(\Phi)} \mu(t)$ for all distributions $\mu \in \bigcup_s Steps(s)$.

Lemma 9.3.2 *For all $s \in S$:*

$$s \;\models_{\mathcal{A}}\; \mathsf{Prob}_{\bowtie p}(X\Phi) \quad \textit{iff} \quad \sum_{t \in Sat_{\mathcal{A}}(\Phi)} \mu(t) \;\bowtie\; p \quad \textit{for all } \mu \in Steps(s)$$

[17] Our MTBDD-based symbolic model checker of Chapter 10 applies this iterative method.

[18] This observation is no surprise as fairness is a property that only concerns the infinite behaviours and the probabilities for the fulpaths satisfying *PCTL* path formulas built from next step X or bounded until $\mathcal{U}^{\leq k}$ only depend on the finite paths up to length k (where $k = 1$ in the case of the next step operator). On the other hand, unbounded until $\mathcal{U}$ asserts something about the infinite behaviour and, in general, an investigation of the finite paths up to a fixed length is not sufficient to reason about the fulpaths satisfying $\Phi_1 \mathcal{U} \Phi_2$.

Proof: easy verification. ■

E.g. $Sat_{\mathcal{A}}(\mathsf{Prob}_{\sqsupseteq p}(X\Phi))$ is the set of states $s \in S$ where $\min_{\mu \in Steps(s)} \mu\,[Sat_{\mathcal{A}}(\Phi)] \bowtie p$.

9.3.2 Bounded until

The below characterization induces the computation of e.g. $Sat_{\mathcal{A}}(\mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U}^{\leq k} \Phi_2))$ by recursively calculating the probabilities

$$\min_{A \in \mathcal{A}dv} \; p_s^A(\Phi_1 \mathcal{U}^{\leq l} \Phi_2), \quad l = 0, \ldots, k,$$

where $p_s^A(\Phi_1 \mathcal{U}^{\leq l} \Phi_2)$ is the probability measure of all fulpaths $\pi \in Path_{ful}^A(s)$ with $\pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}^{\leq l} \Phi_2$. This method is just an adaption of the method proposed by [HaJo94] sketched on page 217.

Lemma 9.3.3 *For all $s \in S$:*

$$\begin{aligned} s \;&\models_{\mathcal{A}}\; \mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U}^{\leq k} \Phi_2) \quad \textit{iff} \quad q_{s,k}^{min} \sqsupseteq p, \\ s \;&\models_{\mathcal{A}}\; \mathsf{Prob}_{\sqsubset p}(\Phi_1 \mathcal{U}^{\leq k} \Phi_2) \quad \textit{iff} \quad q_{s,k}^{max} \sqsubset p. \end{aligned}$$

Here, the values $q_{s,l}^{max}$ and $q_{s,l}^{min}$, $s \in S$, $l = 0, 1, \ldots, k$, are defined as follows.

- *If $s \models_{\mathcal{A}} \neg\Phi_1 \wedge \neg\Phi_2$ then $q_{s,l}^{max} = q_{s,l}^{min} = 0$ for all $l \geq 0$.*
- *If $s \models_{\mathcal{A}} \Phi_2$ then $q_{s,l}^{max} = q_{s,l}^{min} = 1$ for all $l \geq 0$.*
- *If $s \models_{\mathcal{A}} \Phi_1$ then $q_{s,0}^{max} = q_{s,0}^{min} = 0$ and*

$$q_{s,l+1}^{max} \;=\; \max_{\mu \in Steps(s)} \sum_{t \in S} \mu(t) \cdot q_{t,l}^{max}, \quad q_{s,l+1}^{min} \;=\; \min_{\mu \in Steps(s)} \sum_{t \in S} \mu(t) \cdot q_{t,l}^{min}.$$

Proof: For $A \in \mathcal{A}$, let $q_{s,l}^A = Prob\left\{\pi \in Path_{ful}^A(s) : \pi \models_{\mathcal{A}} \Phi_1 \mathcal{U}^{\leq l} \Phi_2\right\}$. By induction on l, we get $q_{s,l}^{max} = \max_{A \in \mathcal{A}} q_{s,l}^A$ and $q_{s,l}^{min} = \min_{A \in \mathcal{A}} q_{s,l}^A$ which yields the claim. ■

Example 9.3.4 We consider the system of Figure 9.6 (page 219) and the *PCTL* formula $\mathsf{Prob}_{\leq \frac{1}{3}}(a\mathcal{U}^{\leq 3} b)$. Using the notations of Lemma 9.3.3 (page 219), we have $q_{v,3}^{max} = q_{z,3}^{max} = 1$,

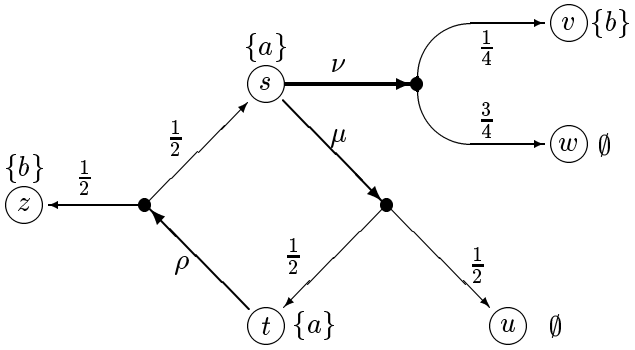


Figure 9.6: $t \not\models_{\mathcal{A}} \mathsf{Prob}_{\leq \frac{1}{3}}(a\mathcal{U}^{\leq 3} b)$ and $s \models_{\mathcal{A}} \mathsf{Prob}_{\leq \frac{1}{3}}(a\mathcal{U}^{\leq 3} b)$

$q_{w,3}^{max} = q_{u,3}^{max} = 0$ and the recursive formulas

$$q^{max}_{s,i+1} \;=\; \max\left\{\, \tfrac{1}{2}\cdot q^{max}_{t,i},\;\; \tfrac{1}{4}\,\right\}, \qquad q^{max}_{t,i+1} \;=\; \tfrac{1}{2} \;+\; \tfrac{1}{2}\cdot q^{max}_{s,i}$$

where $q^{max}_{s,0} = q^{max}_{t,0} = 0$. We obtain $q^{max}_{s,1} = 1/4,\;\; q^{max}_{t,1} = 1/2,\;\; q^{max}_{s,2} = 1/4,\;\; q^{max}_{t,2} = 5/8,$

$$q^{max}_{s,3} \;=\; \tfrac{5}{16} \quad \text{and} \quad q^{max}_{t,3} \;=\; \tfrac{21}{32}.$$

Hence, $t \not\models_{\mathcal{A}} \mathsf{Prob}_{\leq \frac{1}{3}}(a\mathcal{U}^{\leq 3}b)$ and $s \models_{\mathcal{A}} \mathsf{Prob}_{\leq \frac{1}{3}}(a\mathcal{U}^{\leq 3}b)$. ■

9.3.3 Unbounded until

This section is concerned with the unbounded until operator, i.e. formulas of the form $\mathsf{Prob}_{\bowtie p}(\Phi_1\mathcal{U}\Phi_2)$ and the satisfaction relations $\models$, $\models_{fair}$, $\models_{sfair}$ and $\models_{Wfair}$.

Simplified notations: For the rest of this section, we fix two *PCTL* formulas Φ_1, Φ_2 over AP. We suppose that the sets of states $s \in S$ with $s \models_{\mathcal{A}} \Phi_i$ are already computed. We may suppose that Φ_1, Φ_2 are atomic propositions with $\Phi_i \in \mathcal{L}(s_i)$ if and only if $s \models_{\mathcal{A}} \Phi_i$, $i = 1, 2$. This simplifying assumption allows us to use the same notation for all four interpretations (since $s \models_{\mathcal{A}} a$ iff $s \models a$ for all atomic propositions a), and is made for this reason alone. We simply write $Sat(\Phi_i)$ rather than $Sat_{\mathcal{A}}(\Phi_i)$, $i = 1, 2$.

Notation 9.3.5 [The probabilities $p^*_s(\Phi_1\mathcal{U}\Phi_2)$] *For $A \in \mathcal{A}dv$ and $s \in S$, let*

$$\begin{aligned}
p^A_s(\Phi_1\mathcal{U}\Phi_2) \;&=\; Prob\left\{\pi \in Path^A_{full}(s) : \pi \models\; \Phi_1\mathcal{U}\Phi_2\right\},\\
p^{max}_s(\Phi_1\mathcal{U}\Phi_2) \;&=\; \sup\left\{\; p^A_s(\Phi_1\mathcal{U}\Phi_2) \;:\; A \in \mathcal{A}dv \;\right\},\\
p^{min}_s(\Phi_1\mathcal{U}\Phi_2) \;&=\; \inf\left\{\; p^A_s(\Phi_1\mathcal{U}\Phi_2) \;:\; A \in \mathcal{A}dv \;\right\}.
\end{aligned}$$

By the results of [CoYa90, BidAl95] (more precisely, by Corollary 20 (part 1) of [BidAl95], which uses the results of [CoYa90]):

$$\begin{aligned}
p^{max}_s(\Phi_1\mathcal{U}\Phi_2) \;&=\; \max\left\{\; p^A_s(\Phi_1\mathcal{U}\Phi_2) \;:\; A \in \mathcal{A}dv_{simple} \;\right\},\\
p^{min}_s(\Phi_1\mathcal{U}\Phi_2) \;&=\; \min\left\{\; p^A_s(\Phi_1\mathcal{U}\Phi_2) \;:\; A \in \mathcal{A}dv_{simple} \;\right\}.
\end{aligned}$$

Observe that $\mathcal{A}dv_{simple}$ is finite (thus, $\min_{A\in\mathcal{A}dv_{simple}}$ and $\max_{A\in\mathcal{A}dv_{simple}}$ exist). In particular, this yields that $p^{max}_s(\Phi_1\mathcal{U}\Phi_2) = \max\left\{p^A_s(\Phi_1\mathcal{U}\Phi_2) : A \in \mathcal{A}dv\right\}$ and $p^{min}_s(\Phi_1\mathcal{U}\Phi_2) = \min\left\{p^A_s(\Phi_1\mathcal{U}\Phi_2) : A \in \mathcal{A}dv\right\}$; thus, also $\max_{A\in\mathcal{A}dv}$ and $\min_{A\in\mathcal{A}dv}$ exist. Immediately by the definition of $p^{min}_s(\cdot)$ and $p^{max}_s(\cdot)$ we get:

(i) $s \;\models\; \mathsf{Prob}_{\sqsubseteq p}(\Phi_1\mathcal{U}\Phi_2)$ iff $p^{max}_s(\Phi_1\mathcal{U}\Phi_2) \;\sqsubseteq\; p$.
(ii) $s \;\models\; \mathsf{Prob}_{\sqsupseteq p}(\Phi_1\mathcal{U}\Phi_2)$ iff $p^{min}_s(\Phi_1\mathcal{U}\Phi_2) \;\sqsupseteq\; p$.

This fact is used in the *PCTL* model checker of [BidAl95]. Having obtained the sets $Sat_{\mathcal{A}dv}(\Phi_i)$, $i = 1, 2$, one computes the values $p^{max}_s(\Phi_1\mathcal{U}\Phi_2)$ and $p^{min}_s(\Phi_1\mathcal{U}\Phi_2)$ which yield

$$\begin{aligned}
Sat_{\mathcal{A}dv}\left(\; \mathsf{Prob}_{\sqsubseteq p}(\Phi_1\mathcal{U}\Phi_2) \;\right) \;&=\; \left\{s \in S : p^{max}_s(\Phi_1\mathcal{U}\Phi_2) \sqsubseteq p\right\},\\
Sat_{\mathcal{A}dv}\left(\; \mathsf{Prob}_{\sqsupseteq p}(\Phi_1\mathcal{U}\Phi_2) \;\right) \;&=\; \left\{s \in S : p^{min}_s(\Phi_1\mathcal{U}\Phi_2) \sqsupseteq p\right\}.
\end{aligned}$$

[BidAl95] propose to compute the values $p^{max}_s(\Phi_1\mathcal{U}\Phi_2)$ and $p^{min}_s(\Phi_1\mathcal{U}\Phi_2)$ by solving certain linear optimization problems. Alternatively, one can use the characterization of the

functions $s \mapsto p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$ and $s \mapsto p_s^{min}(\Phi_1 \mathcal{U} \Phi_2)$ as least fixed points of certain operators on function space $S \to [0,1]$ and compute (approximations for) them with iterative methods.[19] See Remark 3.2.12 (page 43).

We now turn to the question how to deal with the satisfaction relation that involve fairness (namely, the satisfaction relations $\models_{fair}$, $\models_{sfair}$ and $\models_{Wfair}$). For this, we present a series of technical results that characterizes the states where $\mathsf{Prob}_{\bowtie p}(\Phi_1 \mathcal{U} \Phi_2)$ holds with respect to one of the above satisfaction relations. For readers' convenience we state the main theorems in this section without proof (those are included in Section 9.5, page 241 ff). Instead, we include justification for the technical results in the form of examples and informal explanations.

First, we observe that the results by Emerson & Lei [EmLei85] stating that *CTL* model checking under fairness assumptions can be reduced to CTL^* model checking cannot be adapted for the probabilistic case (for the logics *PCTL* and $PCTL^*$). In the non-probabilistic case (i.e. when using *CTL*), fairness of fulpaths can be expressed by path formulas of CTL^*. Typically, this is achieved by means of formulas of the form

$$\varphi_{fair} \;=\; \bigvee_i \bigwedge_j (\Diamond\Box\varphi_{i,j} \vee \Box\Diamond\psi_{i,j})$$

where a fulpath π is said to be fair if $\pi \models \varphi_{fair}$. The model checking for *CTL* under fairness assumptions (i.e. with respect to the satisfaction relation $\models_{fair}$ where the *CTL* path quantifiers $\forall$ and $\exists$ range over all fair fulpaths) can be reduced to the model checking problem for CTL^* with respect to the standard satisfaction relation $\models$ since one has an equivalence of the form

$$s \models_{fair} \forall\varphi \quad \text{iff} \quad s \models \forall(\varphi_{fair} \to \varphi).$$

Unfortunately, this equivalence does *not* hold in the probabilistic case. The problem is that formulas of the form $\mathsf{Prob}_{\bowtie p}(\varphi_{fair} \to \varphi)$ interpreted over $\models$ state that, in all adversaries (whether fair or unfair), the measure of all fair fulpaths that satisfy φ is $\bowtie p$, i.e.

$$Prob\left\{\pi \in Path_{ful}^{A}(s) : \pi \text{ is fair and } \pi \models \varphi\right\} \;\bowtie\; p \quad \text{for all } A \in \mathcal{A}dv$$

whereas the interpretation with respect to $\models_{fair}$ quantifies over the fair adversaries; thus, $\mathsf{Prob}_{\bowtie p}(\varphi)$ interpreted over $\models_{fair}$ asserts that

$$Prob\left\{\pi \in Path_{ful}^{F}(s) : \pi \models \varphi\right\} \;\bowtie\; p \quad \text{for all } F \in \mathcal{A}dv_{fair}.$$

Hence, the model checking algorithms for $PCTL^*$ cannot be used to handle fairness (at least not in a straightforward manner).

Recall the above mentioned result of [CoYa90, BidAl95] ((i) and (ii) on page 220) which asserts that satisfaction with respect to $\models$ (that ranges over all adversaries) only depends on the probabilities under the *simple* adversaries. Now we will see that item (i) carries over to the satisfaction relations $\models_{fair}$ and $\models_{Wfair}$ (Theorem 9.3.6, page 222, and Theorem 9.3.7, page 222), while (ii) does not (cf. Example 9.3.20, page 226). In particular, the maximal

[19] In Chapter 10 where we describe a MTBDD-based *PCTL* model checking algorithm for stratified systems we make use of the iterative method.

probabilities under all fair adversaries are given by the maximal probabilities under all simple adversaries. Even though (ii) does not hold for $\models_{fair}$ or $\models_{Wfair}$ we will see that also the minimal probabilities under all fair adversaries can be derived by an investigation of the simple adversaries. More precisely, the minimal probability for a *PCTL* path formula $\Phi_1 \mathcal{U} \Phi_2$ under all fair adversaries can be described in terms of the maximal probability for another *PCTL* path formula $a_1 \mathcal{U} a_2$ under all simple adversaries. Thus, both the minimal and maximal probabilities under all fair adversaries can be expressed by means of the simple adversaries. In our opinion, this is a surprising result since the simple adversaries are "extremely unfair".

Formulas of the form $\mathsf{Prob}_{\sqsubseteq p}(\Phi_1 \mathcal{U} \Phi_2)$

We consider formulas of the form $\mathsf{Prob}_{\sqsubseteq p}(\Phi_1 \mathcal{U} \Phi_2)$ for which we need the "maximal" probabilities under all fair (strictly fair or W-fair) adversaries. First, we deal with the satisfaction relation $\models_{fair}$. Clearly, $p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) \geq p_s^F(\Phi_1 \mathcal{U} \Phi_2)$ for all fair adversaries F. Vice versa, for each simple adversary A, there is a fair adversary F_A where $Path_{fin}^{F_A}$ contains all finite paths $\sigma \in Path_{fin}^A$ such that $\sigma(i) \models \Phi_1 \wedge \neg \Phi_2$, $i = 0, 1, \ldots, |\sigma| - 1$, and $last(\sigma) \models \Phi_2$ (cf. Lemma 9.5.15, page 243). Thus, if we take A to be a simple adversary where $p_s^A(\Phi_1 \mathcal{U} \Phi_2) = p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$ (which exists by the results of [CoYa90, BidAl95]) then we get $p_s^{F_A}(\Phi_1 \mathcal{U} \Phi_2) \geq p_s^A(\Phi_1 \mathcal{U} \Phi_2) = p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$. This yields

$$(*) \quad p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) = \max \left\{ p_s^F(\Phi_1 \mathcal{U} \Phi_2) : F \in \mathcal{A}dv_{fair} \right\}$$

and we obtain the following theorem.

Theorem 9.3.6 *For all $s \in S$:*

$$s \models_{fair} \mathsf{Prob}_{\sqsubseteq p}(\Phi_1 \mathcal{U} \Phi_2) \quad \textit{iff} \quad p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) \sqsubseteq p.$$

Proof: see Section 9.5.2, Theorem 9.5.19 (page 245). ■

As each fair adversary A is W-fair, (*) yields

$$p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) = \max \left\{ p_s^F(\Phi_1 \mathcal{U} \Phi_2) : F \in \mathcal{A}dv_{Wfair} \right\}.$$

Thus, Theorem 9.3.6 carries over to the satisfaction relation $\models_{Wfair}$:

Theorem 9.3.7 *For all $s \in S$:*

$$s \models_{Wfair} \mathsf{Prob}_{\sqsubseteq p}(\Phi_1 \mathcal{U} \Phi_2) \quad \textit{iff} \quad p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) \sqsubseteq p.$$

Proof: see Section 9.5.2, Theorem 9.5.19 (page 245). ■

It turns out that the satisfaction relation $\models_{sfair}$ differs from $\models_{fair}$ and $\models$ in that only a stronger statement for formulas of the form $\mathsf{Prob}_{\leq p}(\Phi_1 \mathcal{U} \Phi_2)$ can be shown.

Theorem 9.3.8 *For all $s \in S$:*

$$s \models_{sfair} \mathsf{Prob}_{\leq p}(\Phi_1 \mathcal{U} \Phi_2) \quad \textit{iff} \quad p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) \leq p.$$

Proof: see Section 9.5.2, Theorem 9.5.21 (page 245). ■

The following example shows that the inequality "$\leq p$" in Theorem 9.3.8 cannot be replaced by "$< p$" as $p_s^F(\Phi_1 \mathcal{U} \Phi_2) < p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$ for all $F \in \mathcal{A}dv_{sfair}$ is possible.

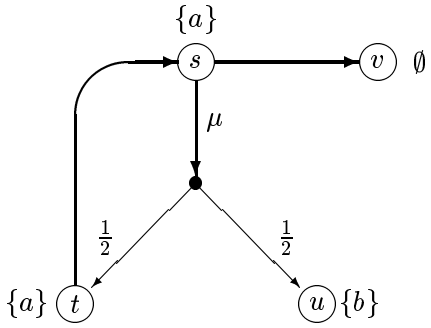


Figure 9.7: $s \models_{sfair} \mathsf{Prob}_{<1}(a\mathcal{U}b)$ while $p_s^{max}(a\mathcal{U}b) = 1$

Example 9.3.9 Consider the system shown in Figure 9.7 (page 223) and the path formula $a\mathcal{U}b$. Then, $p_s^F(a\mathcal{U}b) < 1$ for each strictly fair adversary F. Hence, $s \models_{sfair} \mathsf{Prob}_{<1}(a\mathcal{U}b)$. On the other hand, $p_s^A(a\mathcal{U}b) = 1$ for the simple adversary A with $A(s) = \mu$. Hence, $p_s^{max}(a\mathcal{U}b) = 1$.[20] ■

In order to describe how the set $Sat_{sfair}(\mathsf{Prob}_{<p}(\Phi_1\mathcal{U}\Phi_2))$ can be computed we first introduce some notations. We define $Reach_{\Phi_1\wedge\neg\Phi_2}(s)$ to be the set of states which are reachable in $\mathcal{S}$ from s via a path where all states – possibly except the last one – fulfill the formula $\Phi_1 \wedge \neg\Phi_2$.

Notation 9.3.10 [**The set** $Reach_{\Phi_1\wedge\neg\Phi_2}(s)$] *For $s \in S$, we define*

$$Path_{fin}(s, \Phi_1 \wedge \neg\Phi_2) = \{\sigma \in Path_{fin}(s) : \sigma(i) \models \Phi_1 \wedge \neg\Phi_2, i = 0, 1, \ldots, |\sigma| - 1\},$$
$$Reach_{\Phi_1\wedge\neg\Phi_2}(s) = \{last(\sigma) : \sigma \in Path_{fin}(s, \Phi_1 \wedge \neg\Phi_2)\}.$$

$S^+(\Phi_1, \Phi_2)$ is the set of all states from which one can reach a Φ_2-state via a path through Φ_1-states. Formally:

Notation 9.3.11 [**The set** $S^+(\Phi_1, \Phi_2)$] *We define*

$$S^+(\Phi_1, \Phi_2) = \{s \in S : Reach_{\Phi_1\wedge\neg\Phi_2}(s) \cap Sat(\Phi_2) \neq \emptyset\}.$$

Clearly, $s \in S^+(\Phi_1, \Phi_2)$ iff $p_s^A(\Phi_1\mathcal{U}\Phi_2) > 0$ for some $A \in \mathcal{A}dv$ iff $p_s^{max}(\Phi_1\mathcal{U}\Phi_2) > 0$.

Example 9.3.12 For the system of Figure 9.7 (page 223) we have

$$Reach_{a\wedge\neg b}(s) = Reach_{a\wedge\neg b}(t) = \{s, t, u, v\},$$
$$Reach_{a\wedge\neg b}(x) = \{x\} \text{ for } x \in \{u, v\}.$$

Moreover, $Sat(b) = \{u\}$, $Sat(a) = \{s, t\}$. Thus, $S^+(a, b) = \{s, t, u\}$. ■

We introduce $MaxSteps(s, \Phi_1, \Phi_2)$ as the set of steps $\mu \in Steps(s)$ that might be chosen in state s by an adversary A that yields the maximal probabilities for $\Phi_1\mathcal{U}\Phi_2$.

Notation 9.3.13 [**The sets** $MaxSteps(s, \Phi_1, \Phi_2)$] *If $s \in S \setminus Sat(\Phi_1)$ then we define $MaxSteps(s, \Phi_1, \Phi_2) = Steps(s)$. For $s \in Sat(\Phi_1)$, let $MaxSteps(s, \Phi_1, \Phi_2)$ be the set*

[20] Note that A is fair, cf. Example 3.2.18 on page 46. Thus, $s \not\models_{fair} \mathsf{Prob}_{<1}(a\mathcal{U}b)$.

of $\mu \in \mathit{Steps}(s)$ *such that*

$$p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) \;=\; \sum_{t \in S} \mu(t) \cdot p_t^{max}(\Phi_1 \mathcal{U} \Phi_2).$$

We define a set $T^{max}(\Phi_1, \Phi_2)$ for which we show (in Section 9.5.2, page 246 ff) that it contains exactly those states s such that $p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) = p_s^F(\Phi_1 \mathcal{U} \Phi_2)$ for some $F \in \mathcal{A}dv_{sfair}$.

Notation 9.3.14 [The set $T^{max}(\Phi_1, \Phi_2)$] *We define*

$$T^{max}(\Phi_1, \Phi_2) \;=\; \bigcup_{i \geq 0} T_i^{max}(\Phi_1, \Phi_2)$$

where $T_0^{max}(\Phi_1, \Phi_2) \;=\; \mathit{Sat}(\Phi_2) \cup (S \setminus S^+(\Phi_1, \Phi_2))$ *and*

$T_j^{max}(\Phi_1, \Phi_2) \;=\; T_{j,1}^{max}(\Phi_1, \Phi_2) \;\cup\; T_{j,2}^{max}(\Phi_1, \Phi_2)$ *for* $j \geq 1$.

Here,

- $T_{j,1}^{max}(\Phi_1, \Phi_2)$ *consists of all states* $t \in S \setminus \bigcup_{i<j} T_i^{max}(\Phi_1, \Phi_2)$ *such that*

  $$\mathit{Supp}(\mu) \subseteq \bigcup_{i<j} T_i^{max}(\Phi_1, \Phi_2)$$

  for some $\mu \in \mathit{MaxSteps}(t, \Phi_1, \Phi_2)$,
- $T_{j,2}^{max}(\Phi_1, \Phi_2) \;=\; \bigcup_{T \in \mathcal{T}} T$ *where* $\mathcal{T}$ *is the collection of all subsets* T *of*

  $$S \setminus \left(\bigcup_{i<j} T_i^{max}(\Phi_1, \Phi_2) \;\cup\; T_{j,1}^{adm}(\Phi_1, \Phi_2) \right)$$

  such that for all $t \in T$*:*

  (i) $\mathit{MaxSteps}(t, \Phi_1, \Phi_2) \;=\; \mathit{Steps}(t)$

  (ii) for all $\mu \in \mathit{Steps}(t)$*:*

  $$\mathit{Supp}(\mu) \;\subseteq\; T \;\cup\; \bigcup_{i<j} T_i^{max}(\Phi_1, \Phi_2) \;\cup\; T_{j,1}^{max}(\Phi_1, \Phi_2).$$

Example 9.3.15 We consider the system of Figure 9.8 (page 225). We write T_*^{max}, p_*^{max} and $\mathit{MaxSteps}(s)$ rather than $T_*^{max}(a, b)$, $p_*^{max}(a \mathcal{U} b)$ and $\mathit{MaxSteps}(s, a, b)$. Then, $p_{s_5}^{max} = 1/2$, $p_{s_4}^{max} = p_{s_3}^{max} = p_{s_2}^{max} = 1/3$,

$$p_{s_6}^{max} \;=\; \frac{1}{9} \cdot \sum_{i=0}^{\infty} \left(\frac{1}{2}\right)^i \;=\; \frac{2}{9}$$

and $p_{s_1}^{max} = \max\{2/3 \cdot 1/3, 2/9\} = 2/9$. Hence, $\mathit{Steps}(s_j) = \mathit{MaxSteps}(s_j)$, $j = 1, \ldots, 5$, $\nu_2 = \mu_{u_6}^1 \notin \mathit{MaxSteps}(s_6)$. Thus,

- $T_0^{max} = S \setminus S^+(a, b) \cup \mathit{Sat}(b) = \{u_1, u_3, u_5, u_6, u_6', t_5, t_6\}$,
- $T_{1,1}^{max} = \{s_5\}$, $T_{1,2}^{max} = \{s_3, s_4\}$,
- $T_{2,1}^{max} = \{s_2\}$, $T_{2,2}^{max} = \emptyset$,

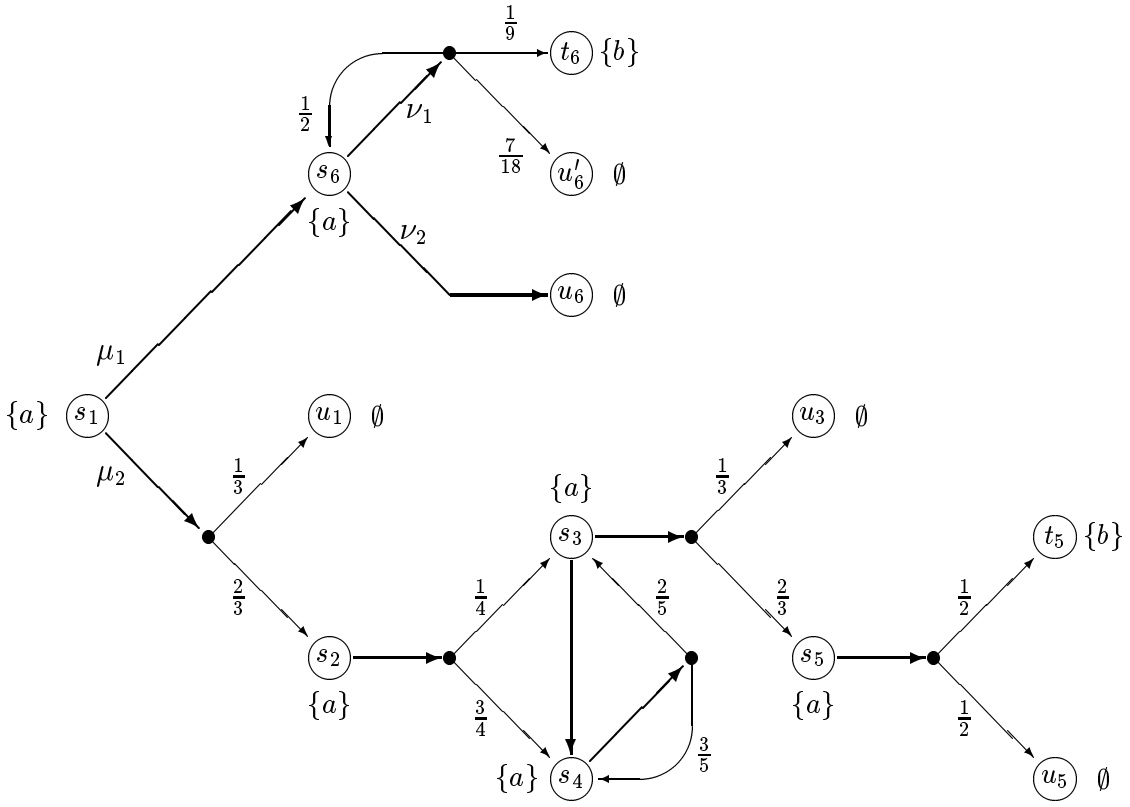


Figure 9.8: $s_6 \models_{sfair} \mathsf{Prob}_{<\frac{2}{9}}(a\mathcal{U}b)$ while $p^{max}_{s_6}(a\mathcal{U}b) = \frac{2}{9}$

- $T^{max}_{3,1} = \{s_1\}$

and $T^{max}_{*} = \emptyset$ in all other cases. We get $T^{max}(a,b) = S \setminus \{s_6\}$. ■

In Section 9.5.2 (page 246 ff) we show: For all $s \in S \setminus T^{max}(\Phi_1, \Phi_2)$ and strictly fair adversaries F, there exists $\sigma \in Path^F_{fin}(s)$ with $F(\sigma) \notin MaxSteps(last(\sigma), \Phi_1, \Phi_2)$ and $\sigma(i) \models \Phi_1 \wedge \neg\Phi_2$, $i = 0, 1, \ldots, |\sigma|$; thus, $p^F_\sigma(\Phi_1 \mathcal{U} \Phi_2) < p^{max}_{last(\sigma)}(\Phi_1 \mathcal{U} \Phi_2)$.[21] We conclude:

$$p^F_s(\Phi_1 \mathcal{U} \Phi_2) < p^{max}_s(\Phi_1 \mathcal{U} \Phi_2) \text{ for all } F \in \mathcal{A}dv_{sfair} \text{ and } s \notin T^{max}(\Phi_1, \Phi_2).$$

(See Lemma 9.5.33 on page 249.) For instance, for the state s_6 of the system in Example 9.3.15 (see Figure 9.8 on page 225) and each $F \in \mathcal{A}dv_{sfair}$, there is some finite path σ in $Path^F_{fin}$ of the form $s_6 \xrightarrow{\nu_1} s_6 \xrightarrow{\nu_1} \ldots \xrightarrow{\nu_1} s_6$ with $F(\sigma) = \nu_2 \notin MaxSteps(s_6, a, b)$. Hence,

$$p^F_{s_6}(a\mathcal{U}b) \;<\; \tfrac{2}{9} \;=\; p^{max}_{s_6}(a\mathcal{U}b).$$

Vice versa, a strictly fair adversary F with $F(\sigma) = \mu_t$ for all $\sigma \in Path^F_{fin}$ with $last(\sigma) = t \in T^{max}_{j,1}(\Phi_1, \Phi_2)$ (where μ_t is as in Notation 9.3.14, page 224) can be defined. For this adversary F,

$$p^F_t(\Phi_1 \mathcal{U} \Phi_2) \;=\; p^{max}_t(\Phi_1 \mathcal{U} \Phi_2) \text{ for all } t \in T^{max}(\Phi_1, \Phi_2).$$

(See Lemma 9.5.31 on page 248). For instance, for the system of Example 9.3.15 (see Figure 9.8 on page 225) and each strictly fair adversary F with $F(\sigma) = \mu_2$ if $\sigma \in Path^F_{fin}$

[21] Here, $p^F_\sigma(\cdots) = p^{F'}_{last(\sigma)}(\cdot)$ where F' is an adversary with $F'(\gamma) = F(\sigma \circ \gamma)$ for all $\gamma \in Path_{fin}(last(\sigma))$.

and $last(\sigma) = s_1$, we have $p_t^F(a\mathcal{U}b) = p_t^{max}(a\mathcal{U}b)$ for all $t \in T^{max}(a,b)$. Note that there is no fair fulpath where s_3 occurs infinitely often. Thus, $p_{s_3}^F(a\mathcal{U}b) = 1/3 = p_{s_3}^{max}(a\mathcal{U}b)$ for all $F \in \mathcal{A}dv_{sfair}$. These observations lead to the following theorem.

Theorem 9.3.16 *For all $s \in S$:*

$$s \models_{sfair} \mathsf{Prob}_{<p}(\Phi_1\mathcal{U}\Phi_2) \iff \begin{cases} p_s^{max}(\Phi_1\mathcal{U}\Phi_2) < p & : \text{ if } s \in T^{max}(\Phi_1,\Phi_2) \\ p_s^{max}(\Phi_1\mathcal{U}\Phi_2) \leq p & : \text{ otherwise.} \end{cases}$$

Proof: see Section 9.5.2, Theorem 9.5.35 (page 250). ■

Example 9.3.17 For the system of Example 9.3.15 (see Figure 9.8 on page 225) we have:

$$s_6 \models_{sfair} \mathsf{Prob}_{<\frac{2}{9}}(a\mathcal{U}b) \text{ but } s_6 \not\models_{fair} \mathsf{Prob}_{<\frac{2}{9}}(a\mathcal{U}b)$$

and $s_1 \not\models_{sfair} \mathsf{Prob}_{<\frac{2}{9}}(a\mathcal{U}b)$. ■

Corollary 9.3.18 *If $s \notin T^{max}(\Phi_1,\Phi_2)$ then $s \models_{sfair} \mathsf{Prob}_{<1}(\Phi_1\mathcal{U}\Phi_2)$.*

Example 9.3.19 For the system in Example 9.3.9 (page 223), we have $S^+(a,b) = \{s,t,u\}$ and $Sat(b) = \{u\}$. Hence, $T_0^{max}(a,b) = \{u,v\}$. For the simple adversary A with $A(s) = \mu$ we get $p_s^A(a\mathcal{U}b) = p_t^A(a\mathcal{U}b)$ and

$$p_s^A(a\mathcal{U}b) \;=\; \tfrac{1}{2} \;+\; \tfrac{1}{2} \cdot p_t^A(a\mathcal{U}b).$$

Hence, $p_s^A(a\mathcal{U}b) = 1$. Thus, $p_s^{max}(a\mathcal{U}b) = 1$. Since

$$\sum_x \mu_v^1(x) \cdot p_x^{max}(a\mathcal{U}b) \;=\; p_v^{max}(a,b) \;=\; 0$$

we get $MaxSteps(s,a,b) = \{\mu\} \neq Steps(s)$. This yields $T_{1,1}^{max}(a,b) = T_{1,2}^{max}(a,b) = \emptyset$ and

$$T^{max}(a,b) \;=\; T_0^{max}(a,b) \;=\; \{u,v\}.$$

By Corollary 9.3.18 (page 226), $s \models_{sfair} \mathsf{Prob}_{<1}(a\mathcal{U}b)$ as $s \notin T^{max}(a,b)$. ■

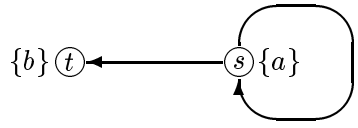
Formulas of the form $\mathsf{Prob}_{\sqsupseteq p}(\Phi_1\mathcal{U}\Phi_2)$

First, we consider $\models_{fair}$ and $\models_{sfair}$. The following example shows that

$$p_s^{min}(\Phi_1\mathcal{U}\Phi_2) \;<\; \inf\left\{p_s^F(\Phi_1\mathcal{U}\Phi_2) : F \in \mathcal{A}dv_{fair}\right\}$$

(thus, $s \models_{fair} \mathsf{Prob}_{\sqsupseteq p}(\Phi_1\mathcal{U}\Phi_2)$ while $p_s^{min}(\Phi_1\mathcal{U}\Phi_2) < p$) is possible. In particular, this example shows the difference between $\models_{fair}$ and $\models$, and that, in item (ii) on page 220, the satisfaction relation $\models$ cannot be replaced by $\models_{fair}$ or $\models_{sfair}$.

Example 9.3.20 Consider the following system and the path formula $a\mathcal{U}b$.



Then, $p_s^A(a\mathcal{U}b) = 0$ for the simple adversary A with $A(s) = \mu_s^1$, whereas $p_s^F(a\mathcal{U}b) = 1$ for each fair adversary F. Hence, $s \models_{fair} \mathsf{Prob}_{\geq 1}(a\ \mathcal{U}\ b\ )$ but $p_s^{min}(a\mathcal{U}b) = 0$. ■

The above simple example demonstrates that the progress property $\mathsf{Prob}_{\geq 1}(a\mathcal{U}b)$ cannot be established unless fairness is required. The "problem" with the simple adversary A where $A(s) = \mu_s^1$ is that it forces the system to stay forever in a "non-successful" state (s) from which a "successful" state (t) can be reached. In fair adversaries, with probability 1, all states that are reachable from a state that is visited infinitely often are also visited infinitely often (see Lemma 9.5.10 on page 242 and Lemma 9.5.12 on page 243). This explains why $p_s^A(a\mathcal{U}b)$ cannot be "approximated" by fair adversaries. Moreover, we will see (in Corollary 9.5.23, page 246) that, for each fair adversary F and state s, the measure of the fulpaths $\pi \in Path_{ful}^F(s)$ where either $\Phi_1\mathcal{U}\Phi_2$ holds or that eventually reach a state that is not contained in $S^+(\Phi_1, \Phi_2)$ is 1. Thus, $1 - p_s^F(\Phi_1\mathcal{U}\Phi_2)$ is the probability (with respect to the adversary F) for s to reach a state in $S \setminus S^+(\Phi_1, \Phi_2)$ via a finite path that only passes states in $S^+(\Phi_1, \Phi_2) \setminus Sat(\Phi_2)$.

Notation 9.3.21 [The set $S^?(\Phi_1, \Phi_2)$] *Let* $S^?(\Phi_1, \Phi_2) = S^+(\Phi_1, \Phi_2) \setminus Sat(\Phi_2)$.

Having computed the sets $S^+(\Phi_1, \Phi_2)$ and $S^?(\Phi_1, \Phi_2)$ (which will be explained in Section 9.3.5, page 231 ff) we may extend AP by "fresh" atomic propositions that characterize the sets $S^+(\Phi_1, \Phi_2)$ and $S^?(\Phi_1, \Phi_2)$.

Notation 9.3.22 [The atomic propositions a^+ and $a^?$] *In the sequel, we suppose* a^+, $a^? \in AP$ *with* $a^+ \in \mathcal{L}(s)$ *iff* $s \in S^+(\Phi_1, \Phi_2)$ *and* $a^? \in \mathcal{L}(s)$ *iff* $a^? \in S^?(\Phi_1, \Phi_2)$.

The following theorem states that to handle formulas of the type $\mathsf{Prob}_{\trianglerighteq p}(\Phi_1\mathcal{U}\Phi_2)$ with respect to $\models_{fair}$ it suffices to compute the values $p_s^{max}(a^?\mathcal{U}\neg a^+)$.

Theorem 9.3.23 *For all* $s \in S$:

$$s \models_{fair} \mathsf{Prob}_{\trianglerighteq p}(\Phi_1\mathcal{U}\Phi_2) \quad \textit{iff} \quad 1 - p_s^{max}(a^?\mathcal{U}\neg a^+) \trianglerighteq p.$$

Proof: see Section 9.5.2, Theorem 9.5.25 (page 246). ■

If $Reach_{\Phi_1\wedge\neg\Phi_2}(s) \subseteq S^+(\Phi_1, \Phi_2)$ and $s \in Sat(\Phi_1)$ then $p_s^{max}(a^?\mathcal{U}\neg a^+) = 0$. Hence, $s \models_{fair} \mathsf{Prob}_{\geq 1}(\Phi_1\mathcal{U}\Phi_2)$. Vice versa, if $Reach_{\Phi_1\wedge\neg\Phi_2}(s) \not\subseteq S^+(\Phi_1, \Phi_2)$ then there is a finite path $\sigma \in Path_{fin}(s)$ with $\sigma(i) \models \Phi_1 \wedge \neg\Phi_2$, $i = 0, 1, \ldots, |\sigma| - 1$, and $last(\sigma) \notin S^+(\Phi_1, \Phi_2)$. For any (fair) adversary F with $\sigma \in Path_{fin}^F(s)$, we have $p_s^F(\Phi_1\mathcal{U}\Phi_2) \leq 1 - \mathbf{P}(\sigma) < 1$. Thus, $s \not\models_{fair} \mathsf{Prob}_{\geq 1}(\Phi_1\mathcal{U}\Phi_2)$. This leads to the following corollary.

Corollary 9.3.24 *For all* $s \in S$:

$$s \models_{fair} \mathsf{Prob}_{\geq 1}(\Phi_1\mathcal{U}\Phi_2) \quad \textit{iff} \quad Reach_{\Phi_1\wedge\neg\Phi_2}(s) \subseteq S^+(\Phi_1, \Phi_2).$$

In particular, a concurrent probabilistic system $\mathcal{S}$ with initial state s_{init} satisfies a *qualitative progress property* expressed by a *PCTL* formula of the form $\mathsf{Prob}_{\geq 1}(\Diamond\Phi)$ (where satisfaction is understood to be with respect to $\models_{fair}$) if and only if the system is "safe" in the sense that no "deadlocked" state (a state t from which no Φ-state can be reached, i.e. $Reach(t) \cap Sat(\Phi) = \emptyset$) is reachable from the initial state s_{init}:

$$s_{init} \models_{fair} \mathsf{Prob}_{\geq 1}(\Diamond\Phi) \text{ iff } Reach(s_{init}) \subseteq \{s \in S : Reach(s) \cap Sat(\Phi) \neq \emptyset\}$$

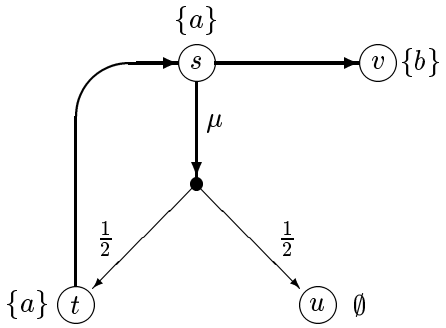


Figure 9.9: $p_s^{max}(a^?\mathcal{U}\neg a^+) = 1$ but $p_s^F(a\mathcal{U}b) > 0$ for all $F \in \mathcal{A}dv_{sfair}$

Thus, for verifying qualitative progress properties as explained above, an analysis of the "topology" of the system suffices. This result was first established in [HSP83]. A similar observation for fully probabilistic systems is made in Section 3.1 (Lemma 3.1.10, page 37).

Example 9.3.25 For the system of Example 9.3.20 (page 226), we have $Reach(s) = \{s, t\}$ and $Sat(b) = \{t\}$. Hence, $s \models_{fair} \mathsf{Prob}_{\geq 1}(\Diamond b)$. ■

Theorem 9.3.26 *For all $s \in S$:*

$$s \models_{sfair} \mathsf{Prob}_{\geq p}(\Phi_1\mathcal{U}\Phi_2) \quad \textit{iff} \quad 1 - p_s^{max}(a^?\mathcal{U}\neg a^+) \geq p.$$

Proof: see Section 9.5.2, Theorem 9.5.27 (page 246). ■

A stronger version of Theorem 9.3.26 stating that $s \models_{sfair} \mathsf{Prob}_{>p}(\Phi_1\mathcal{U}\Phi_2)$ iff $1 - p_s^{max}(a^?\mathcal{U}\neg a^+) > p$ is incorrect, as can be seen from Example 9.3.27 below. (This example again demonstrates the difference between $\models_{sfair}$ and $\models_{fair}$.)

Example 9.3.27 We consider the system shown in Figure 9.9 (page 228). Clearly, u is the only state that satisfies $\neg a^+$. Thus, for the simple adversary A with $A(s) = \mu$ we have $p_s^A(a^?\mathcal{U}\neg a^+) = 1$. This yields $p_s^{max}(a^?\mathcal{U}\neg a^+) = 1$ but $p_s^F(a\mathcal{U}b) > 0$ for all $F \in \mathcal{A}dv_{sfair}$. Hence, $s \models_{sfair} \mathsf{Prob}_{>0}(a\mathcal{U}b)$ while $p_s^{max}(a^?\mathcal{U}\neg a^+) = 1$. ■

The next result is an analogue of Theorem 9.3.16 (page 226), in which we show how to deal with formulas $\mathsf{Prob}_{>p}(\Phi_1\mathcal{U}\Phi_2)$ with respect to the satisfaction relation $\models_{sfair}$.

Theorem 9.3.28 *For all $s \in S$:*

$$s \models_{sfair} \mathsf{Prob}_{>p}(\Phi_1\mathcal{U}\Phi_2) \iff \begin{cases} 1 - p_s^{max}(a^?\mathcal{U}\neg a^+) > p & : \textit{ if } s \in T^{max}(a^?, \neg a^+) \\ 1 - p_s^{max}(a^?\mathcal{U}\neg a^+) \geq p & : \textit{ otherwise.} \end{cases}$$

Proof: see Section 9.5.2, Theorem 9.5.36 (page 250). ■

Corollary 9.3.29 *If $s \notin T^{max}(a^?, \neg a^+)$ then $s \models_{sfair} \mathsf{Prob}_{>0}(\Phi_1\mathcal{U}\Phi_2)$.*

Example 9.3.30 In Example 9.3.27 (page 228) we have $T^{max}(a^?, \neg a^+) = \{u, v\}$. Hence, $s \models_{sfair} \mathsf{Prob}_{>0}(a\mathcal{U}b)$ since $s \notin T^{max}(a^?, \neg a^+)$. ■

Next we consider formulas of the form $\mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U} \Phi_2)$ and the satisfaction relation $\models_{Wfair}$. First, we observe that Theorem 9.3.23 (page 227) does not carry over to $\models_{Wfair}$ since

$$\inf \left\{ p_t^F(\Phi_1 \mathcal{U} \Phi_2) : F \in \mathcal{A}dv_{Wfair} \right\} \;<\; \inf \left\{ p_t^F(\Phi_1 \mathcal{U} \Phi_2) : F \in \mathcal{A}dv_{fair} \right\}$$

is possible. For this, consider the simple system of Figure 9.10 and the simple adversary A

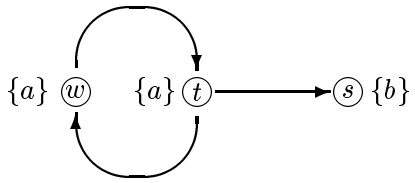


Figure 9.10: $t \not\models_{Wfair} \mathsf{Prob}_{\geq 1}(a \mathcal{U} b)$ while $t \models_{fair} \mathsf{Prob}_{\geq 1}(a \mathcal{U} b)$

with $A(t) = \mu_w^1$. Dealing with $W = \{w\}$, we get the W-fairness of A. Since $p_t^A(a \mathcal{U} b) = 0$, the minimal probability for state t under all W-fair adversaries is 0. On the other hand, if fairness in state t is assumed then the state s will eventually be reached from t. Thus, $p_t^F(a \mathcal{U} b) = 1$ for any adversary F that is fair in state t (e.g. $F \in \mathcal{A}dv_{fair}$); hence, the minimal probability for state t under all fair adversaries is 1. In particular, if $t \notin W$ then

$$t \not\models_{Wfair} \mathsf{Prob}_{\geq 1}(a \mathcal{U} b) \text{ while } t \models_{fair} \mathsf{Prob}_{\geq 1}(a \mathcal{U} b) \text{ (and } p_t^{max}(a^? \mathcal{U} \neg a^+) = 0).$$

To deal with $\models_{Wfair}$, we use an atomic proposition a_W^0 (that characterizes all states s which can reach $S \setminus S^+(\Phi_1, \Phi_2)$ without passing a Φ_2-state with probability 1 in a W-fair adversary) and replace in Theorem 9.3.23 (page 227) the path formula $a^? \mathcal{U} \neg a^+$ by $a^? \mathcal{U} a_W^0$.

Notation 9.3.31 [The set $S_W^0(\Phi_1, \Phi_2)$] *We define*

$$S_W^0(\Phi_1, \Phi_2) \;=\; \bigcup_{i \geq 0} T_i$$

where $T_0 = S \setminus S^+(\Phi_1, \Phi_2)$ *and, for* $i \geq 1$, $T_i \;=\; T_{i,1} \cup T_{i,2}$ *with:*

- $T_{i,1}$ *is the set of states* $t \in S \setminus (T_0 \cup \ldots \cup T_{i-1} \cup Sat(\Phi_2))$ *such that, for some* $\mu_t \in Steps(t)$,

$$Supp(\mu_t) \subseteq T_0 \cup \ldots \cup T_{i-1}.$$

- $T_{i,2} \;=\; \bigcup_{T \in \mathcal{T}_i} T$ *where* $\mathcal{T}_i$ *denotes the collection of all sets*

$$T \;\subseteq\; S \setminus (T_0 \cup \ldots \cup T_{i-1} \cup T_{i,1} \cup Sat(\Phi_2))$$

  such that for all $t \in T$*:*
  - *If* $t \in W$ *then* $Supp(\mu) \;\subseteq\; T_0 \cup \ldots T_{i-1} \cup T_{i,1} \cup T$ *for all* $\mu \in Steps(t)$.
  - *If* $t \notin W$ *then there exists* $\mu_t \in Steps(t)$ *with* $Supp(\mu_t) \;\subseteq\; T_0 \cup \ldots T_{i-1} \cup T_{i,1} \cup T$.

Notation 9.3.32 [The atomic proposition a_W^0] *We suppose* $a_W^0 \in AP$ *with*

$$a_W^0 \in \mathcal{L}(s) \text{ iff } s \in S_W^0(\Phi_1, \Phi_2).$$

The following theorem is an analogue of Theorem 9.3.23 (page 227) which shows that, to handle formulas of the type $\mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U} \Phi_2)$ with respect to $\models_{Wfair}$, it suffices to compute the values $p_s^{max}(a^? \mathcal{U} a_W^0)$.

Theorem 9.3.33 *For all $s \in S$:*

$$s \models_{Wfair} \mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U} \Phi_2) \quad \textit{iff} \quad 1 - p_s^{max}(a^? \mathcal{U} a_W^0) \sqsupseteq p.$$

Proof: see Section 9.5.2, Theorem 9.5.40 (page 252). ■

Example 9.3.34 Consider the path formula $a\mathcal{U}b$ and the system of Figure 9.11 (page 230) where $W = \{w_1, w_2\}$. The set $S_W^0(a,b)$ is obtained as follows. We have $S^+(a,b) = S$. Let $T_{i,1}$, $T_{i,2}$ be as in Notation 9.3.31 (page 229). Then, we get:

- $T_0 = S \setminus S^+(a,b) = \emptyset$ which yields $T_{1,1} = \emptyset$
- $T_{1,2} = \{v_1\}$ (consider the distribution $\mu_{v_1} = \mu_{v_1}^1 \in \mathit{Steps}(v_1)$)
- $T_{2,1} = \{w_1\}$ (consider the distribution $\mu_{w_1} = \mu_{v_1}^1 \in \mathit{Steps}(w_1)$)
- $T_{2,2} = \{w_2, v_2\}$

and $T_{i,1} = T_{i,2} = \emptyset$ for all $i \geq 3$. Thus, $S_W^0(a,b) = \{v_1, v_2, w_1, w_2\}$ and $p_t^{max}(a^? \mathcal{U} a_W^0) = \frac{4}{5}$. Hence,

$$t \models_{Wfair} \mathsf{Prob}_{\geq 0.2}(a\mathcal{U}b) \text{ while } t \not\models_{Wfair} \mathsf{Prob}_{0.5}(a\mathcal{U}b).$$

On the other hand, $t \not\models_{fair} \mathsf{Prob}_{\geq 0.5}(a\mathcal{U}b)$ since $p_t^{max}(a^? \mathcal{U} \neg a^+) = 0$ (as $\mathit{Sat}(\neg a^+) = \emptyset$). ■

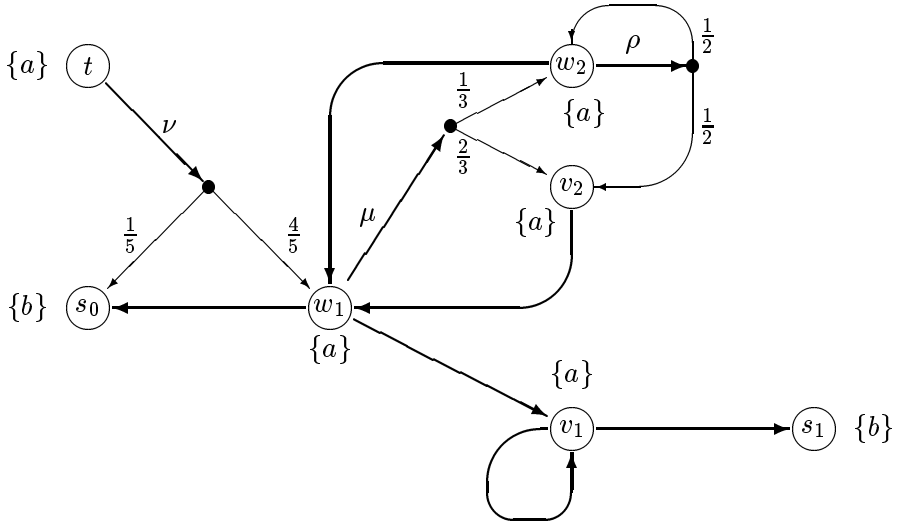


Figure 9.11: $S_W^0(a,b) = \{v_1, v_2, w_1, w_2\}$ for $W = \{w_1, w_2\}$

9.3.4 The connection between $\models$, $\models_{fair}$, $\models_{sfair}$ and $\models_{Wfair}$

From the results of the previous sections we get that the four satisfaction relations only differ for *PCTL* formulas whose outermost operator is the until operator (i.e. formulas of the type $\mathsf{Prob}_{\bowtie}(\Phi_1 \mathcal{U} \Phi_2)$). The difference between the standard satisfaction relation $\models$ and the satisfaction relations with fairness (see e.g. Example 9.1.6, page 211) is due to the well-known fact that appropriate fairness assumptions might be essential for establishing certain liveness properties. However, the satisfaction relations $\models$, $\models_{fair}$ and $\models_{Wfair}$ coincide when dealing with formulas of the type $\mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U} \Phi_2)$ (provided that Φ_1, Φ_2 cannot be distinguished by $\models$, $\models_{fair}$ and $\models_{Wfair}$). Thus, we get:

Theorem 9.3.35 *Let Φ be a PCTL formula that does not contain subformulas of the form $\mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U} \Phi_2)$ then, for all states s,*

$$s \models \Phi \quad \textit{iff} \quad s \models_{\mathit{fair}} \Phi \quad \textit{iff} \quad s \models_{\mathit{Wfair}} \Phi.$$

Proof: by (i) (page 220), Theorem 9.3.6 (page 222), Theorem 9.3.7 (page 222). ■

From the results of the previous section, we get that the difference between the interpretations $\models_{\mathit{fair}}$ and $\models_{\mathit{sfair}}$ is only marginal. This result is not surprising as it is already shown in [HSP83] that each strictly fair scheduler can be "approximated" by fair schedulers. The precise connection between $\models_{\mathit{fair}}$ and $\models_{\mathit{sfair}}$ is as follows.

Theorem 9.3.36 *Let Φ be a PCTL formula that does not contain subformulas of the form $\mathsf{Prob}_{<p}(\varphi)$ or $\mathsf{Prob}_{>p}(\varphi)$. Then, for all states s:*

$$s \models_{\mathit{fair}} \Phi \quad \textit{iff} \quad s \models_{\mathit{sfair}} \Phi.$$

Proof: follows by Theorem 9.3.6 (page 222), Theorem 9.3.8 (page 222), Theorem 9.3.23 (page 227) and Theorem 9.3.26 (page 228). ■

Clearly, fairness with respect to a proper subset W of S induces a satisfaction relation which, in general, differs from $\models_{\mathit{fair}}$; see e.g. Figure 9.10 (page 229). Dealing with $W = S$, the satisfaction relations $\models_{\mathit{fair}}$ and $\models_{\mathit{Wfair}}$ coincide, although fairness with respect to $W = S$ (in the sense of Definition 3.2.20 page 47) does *not* coincide with fairness in the sense of Definition 3.2.17 (page 46).

Theorem 9.3.37 *If $W = S$ then for all states s and PCTL formulas Φ:*

$$s \models_{\mathit{fair}} \Phi \quad \textit{iff} \quad s \models_{\mathit{Wfair}} \Phi.$$

Proof: see Section 9.5.2 (Theorem 9.5.41, page 252). ■

9.3.5 Complexity of *PCTL* model checking

We summarize the results of the previous sections and investigate the complexity of the resulting *PCTL* model checking algorithm. As before, we assume a fixed finite concurrent probabilistic system $\mathcal{S} = (S, \mathit{Steps}, AP, \mathcal{L})$. Let n be the set of states (i.e. $n = |S|$) and m the number of transitions (i.e. $m = \sum_{s \in S} |\mathit{Steps}(s)|$). W.l.o.g. we may assume that $m \geq n$. Moreover, we fix a *PCTL* formula Ψ for which we want to compute $\mathit{Sat}_{\mathcal{A}}(\Psi)$. The size of the parse tree (the number of nodes) is linear in the length $|\Psi|$. For every node v of the parse tree where the associated formula Φ_v is tt, an atomic proposition or of the form $\neg\Phi$ or $\Phi_1 \wedge \Phi_2$, the costs for computing the set of states fulfilling Φ_v is $\mathcal{O}(n)$. The nodes which represent formulas whose outermost operator is the probabilistic operator combined with next-step (i.e. formulas of the form $\mathsf{Prob}_{\bowtie p}(X\Phi)$) require $\mathcal{O}(nm)$ time since, for every transition $\mu \in \mathit{Steps}(s)$, we have to compute the sum $\sum_{t \in \mathit{Sat}_{\mathcal{A}}(\Phi)} \mu(t)$ (cf. Lemma 9.3.2, page 218). Computing the states that fulfill a formula whose outermost operator is the probabilistic operator combined with bounded until $\mathcal{U}^{\leq k}$ takes $\mathcal{O}(knm)$ time when using the method of Lemma 9.3.3 (page 219). We now show that the time needed for the handling of unbounded until is polynomial in n and m.

Dealing with $\models_{fair}$, $\models_{sfair}$ or $\models_{Wfair}$ and formulas of the form $\mathsf{Prob}_{\bowtie p}(\Phi_1 \mathcal{U} \Phi_2)$ we propose the following procedure. As before, we assume that the sets $Sat_{\mathcal{A}}(\Phi_i)$ are already known. We first compute the set $S^+(\Phi_1, \Phi_2)$ from which we derive the sets $Sat(\neg a^+) = S \setminus S^+(\Phi_1, \Phi_2)$, $Sat(a^?) = S^?(\Phi_1, \Phi_2)$ and $S^0_W(\Phi_1, \Phi_2)$. Using well-known methods of linear programming, the maximal probabilities $p_s^{max}(a_1 \mathcal{U} a_2)$ under *all* adversaries can be computed in time polynomial in n and m (cf. Remark 3.2.12, page 43). Here,

$$(a_1, a_2) \;=\; \begin{cases} (\Phi_1, \Phi_2) & : \text{ if } \bowtie \in \{\leq, <\} \\ (a^?, \neg a^+) & : \text{ if } \bowtie \in \{\geq, >\} \text{ and dealing with } \models_{fair} \text{ or } \models_{sfair} \\ (a^?, a^0_W) & : \text{ if } \bowtie \in \{\geq, >\} \text{ and dealing with } \models_{Wfair}. \end{cases}$$

Moreover, for the satisfaction relation $\models_{sfair}$ we compute the set $T^{max}(a_1, a_2)$. We now show that the above mentioned sets can be computed in polynomial time.

Computation of $S^+(\Phi_1, \Phi_2)$: Let $G^+(\Phi_1, \Phi_2)$ be the directed graph (S, E) where $(s, t) \in E$ iff $t \models \Phi_1 \wedge \neg\Phi_2$ and $\mu(s) > 0$ for some $\mu \in Steps(t)$. Then, $S^+(\Phi_1, \Phi_2)$ is the set of states which are reachable in $G^+(\Phi_1, \Phi_2)$ from a state $s \in Sat(\Phi_2)$. Hence, $S^+(\Phi_1, \Phi_2)$ can be derived by a depth-first search in $G^+(\Phi_1, \Phi_2)$. This yields the time complexity $\mathcal{O}(nm)$ for the computation of $S^+(\Phi_1, \Phi_2)$.[22]

Computation of $T^{max}(a_1, a_2)$**:** In what follows, we simply write $MaxSteps(s)$ rather than $MaxSteps(s, a_1, a_2)$. First, we compute $MaxSteps(s)$ for all $s \in S$ and the sets

$$T_0 = Sat(a_2) \cup (S \setminus S^+(a_1, a_2)), \quad U = \{v \in S \setminus T_0 : MaxSteps(s) \neq Steps(s)\}.$$

We compute the strongly connected components in the directed graph $(S \setminus (T_0 \cup U), E)$ where $(s, t) \in E$ iff $\mu(t) > 0$ for some $\mu \in MaxSteps(s) = Steps(s)$. Let $C_1, \ldots, C_k$ be an enumeration of the strongly connected components which satisfies: if $s \in C_j$, $s' \in C_l$ with $(s, s') \in E$ then $l \leq j$. For $i = 1, \ldots, k$ we compute the set S_i of states $w \in S \setminus T_0$ such that $\mu(w) > 0$ for some $\mu \in Steps(s)$ and $s \in C_i$. Let Z be the set of pairs (v, V) such that $v \in S \setminus T_0$, $\emptyset \neq V \subseteq S \setminus T_0$ and $V = Supp(\mu) \setminus T_0$ for some $\mu \in Steps(v)$. For $z \in Z$, we denote the first component of z by $z.state$, the second component by $z.next$ and we define $|z| = |z.next|$. Let S_0 be the set of states $s \in S \setminus T_0$ with $s = z.state$ for some $z \in Z$ with $|z| = 0$. Initially, we define $T = T_0$. We successively modify S_0, T and $|z|$ by the following procedure:

For $i = 1, 2, \ldots, k+1$ do:

(1) While $S_0 \neq \emptyset$ do:

(1.1) choose some $s \in S_0$

(1.2) $S_0 := S_0 \setminus \{s\}$, $T := T \cup \{s\}$

(1.3) For all $z \in Z$ do:

(1.3.1) If $s \in z.next$ then $|z| := |z| - 1$.

(1.3.2) If $|z| = 0$ then $S_0 := S_0 \cup \{z.state\}$.

(2) If $i \leq k$ and $S_i \subseteq C_i \cup T$ then $S_0 := S_0 \cup C_i \setminus T$.

Then, $T^{max}(a_1, a_2) \;=\; T$.

[22] The construction of $G^+(\Phi_1, \Phi_2)$ needs $\mathcal{O}(nm)$ steps. The time for performing a depth-first search in a directed graph G is linear in the number of nodes and edges. As the number of edges in $G^+(\Phi_1, \Phi_2)$ is bounded by $\min\{n^2, nm\}$ we get the time complexity $\mathcal{O}(nm)$ for the computation of $S^+(\Phi_1, \Phi_2)$.

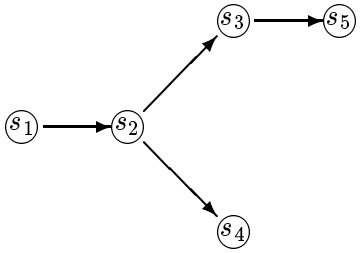


Figure 9.12:

Example 9.3.38 We consider the system of Example 9.3.15 (Figure 9.8, page 225) and deal with $a_1 = a$, $a_2 = b$. Then, μ_1, $\mu_2 \in \mathit{MaxSteps}(s_1)$ and $\mathit{MaxSteps}(s_6) = \{\nu_1\}$. We obtain $U = \{s_6\}$ and $T_0 = S \setminus \{s_1, \ldots, s_6\}$. We first compute the strongly connected components of the directed graph shown in Figure 9.12 (page 233) and obtain

$$C_1 = \{s_5\},\ C_2 = \{s_3, s_4\},\ C_3 = \{s_2\},\ C_4 = \{s_1\},$$
$$S_1 = \emptyset,\ S_2 = \{s_3, s_4, s_5\},\ S_3 = \{s_3, s_4\},\ S_4 = \{s_2, s_6\}.$$

Initially, the set Z consists of the pairs

$$(s_5, \emptyset),\ (s_3, \{s_5\}),\ (s_3, \{s_4\}),\ (s_4, \{s_3, s_4\}), (s_2, \{s_3, s_4\}),\ (s_1, \{s_2\}),\ (s_1, \{s_6\}),\ (s_6, \{s_6\}).$$

This yields $S_0 = \{s_5\}$. In the first iteration step ($i = 1$), we first remove s_5 from S_0 and obtain $S_0 = \{s_3\}$ and $s_5 \in T_0$. Then, we remove s_3 from S_0 and obtain $S_0 = \emptyset$, $s_3 \in T_0$. Thus, in the second iteration step ($i = 2$), step (1) is not applicable (since $S_0 = \emptyset$). In step (2) we have $S_2 = \{s_3, s_4, s_5\} \subseteq C_2 \cup T_0$ and obtain $S_0 = \{s_4\}$. The third iteration step ($i = 3$) removes s_4 from S_0 and yields $S_0 = \{s_2\}$, $s_4 \in T_0$. Then, we remove s_2 from S_0 and obtain $S_0 = \{s_1\}$, $s_2 \in T_0$. Finally, we remove s_1 from S_0 and get $S_0 = \emptyset$ and $s_1 \in T_0$. In the iteration steps $i = 4, 5$, only step (2) is applicable that yields $S_0 = \emptyset$. The algorithm returns $T^{max}(a, b) = S \setminus \{s_6\}$. ■

For the computation of $\mathit{MaxSteps}(\cdot)$, U, the components $C_1, \ldots, C_k$, the sets $S_1, \ldots, S_k$, Z and the function $|\cdot|$, we need $\mathcal{O}(nm)$ time.[23] We suppose the sets T, $C_1, \ldots, C_k$ and $z.\mathit{next}$ for $z \in Z$ to be represented as boolean vectors (one bit for each state $s \in S \setminus T_0$) and that each of the sets Z, $S_0, S_1, \ldots, S_k$ is represented as a list consisting of pointers to their elements. Then, the test in (1) and steps (1.1), (1.2) can be performed in constant time. Step (1.3) can be performed in time linear in the size of Z. As $|Z| \leq m$ we get the time complexity $\mathcal{O}(m)$ for step (1.3). As each state $s \in S \setminus T_0$ can only be chosen once in step (1.1) the while-loop can be performed at most n-times. Hence, ranging over all $i \in \{1, 2, \ldots, k+1\}$ and all executions of the while loop we need $\mathcal{O}(nm)$ time to perform steps (1.1), (1.2) and (1.3). Ranging over all $i \in \{1, 2, \ldots, k\}$ we need

$$\sum_{i=1}^{k} \mathcal{O}\left(|S_i|\right) \;=\; \mathcal{O}\left(\; k \cdot |S \setminus (T_0 \cup U)| \;\right) \;=\; \mathcal{O}(n^2)$$

[23] Note that for the computation of $\mathit{MaxSteps}(s)$ we have to calculate $\sum_{t \in S} \mu(t) \cdot p_s^{max}(a_1 \mathcal{U} a_2)$ for each $\mu \in \mathit{Steps}(s)$. As G has at most $\min\{n^2, nm\}$ edges and as the strongly connected components of a directed graph can always be computed in time linear in the number of states and edges, the computation of $C_1, \ldots, C_k$ takes $\mathcal{O}(nm)$ time.

time for step (2). We conclude that the time complexity of computing $T^{max}(a_1, a_2)$ by the method described above is $\mathcal{O}(nm)$. (Recall that we assume $m \geq n$.)

Computation of $S^0_W(\Phi_1, \Phi_2)$: $S^0_W(\Phi_1, \Phi_2)$ can be computed in a similar way as we obtained $T^{max}(a_1, a_2)$. Thus, also the computation of $S^0_W(\Phi_1, \Phi_2)$ needs $\mathcal{O}(nm)$ time.

Complexity of *PCTL* model checking: Let $p(n, m)$ be a polynomial that stands for the cost function for the time for computing the values $p_s^{max}(a_1, a_2)$ for atomic propositions a_1, a_2. Summing up over all nodes in the parse tree we obtain the time complexity

$$\mathcal{O}\Big(\ |\Psi| \cdot \Big(k^{\Psi} \cdot n \cdot m \ + \ p(n,m) \Big) \ \Big) \, .$$

Here, k^{Ψ} is either 1 (in the case where Ψ does not contain the bounded until operator) or the maximal value k such that Ψ contains a subformula of the form $\mathsf{Prob}_{\bowtie p}\left(\Phi_1 \mathcal{U}^{\leq k} \Phi_2\right)$. I.e., the time complexity is polynomial in the size of the structure and linear in the size of the formula. The space complexity is $\mathcal{O}(n(|\Psi| + m))$. This can be seen as follows. The representation of the set associated with each node v of the parse tree requires $\mathcal{O}(n)$ space. For the system $(S, steps, AP, \mathcal{L})$ itself, we need $\mathcal{O}(nm)$ space (where we neglect the space needed for the representation of the labelling function $\mathcal{L}$). For the computation of $p_s^{max}(a_1 \mathcal{U} a_2)$, we need $\mathcal{O}(n^2)$ space while the computation of the sets $T^{max}(a_1, a_2)$ or $S^0_W(\Phi_1, \Phi_2)$ needs $\mathcal{O}(nm)$ space. (Note that $n \leq m$.) We summarize:

Theorem 9.3.39 *Let $\mathcal{S}$ a finite concurrent probabilistic system, Ψ a PCTL formula and W a subset of the state space of $\mathcal{S}$. Then, $Sat_{\mathcal{A}}(\Psi)$ can be computed in time and space polynomial in the the size of $\mathcal{S}$ and linear in the size of Ψ where $\mathcal{A}$ is one of the adversary types $\mathcal{A}dv$, $\mathcal{A}dv_{fair}$, $\mathcal{A}dv_{sfair}$ or $\mathcal{A}dv_{Wfair}$.*

9.4 Model checking for *LTL*

In the literature, several methods are proposed to verify a probabilistic system against *LTL* formulas or similar specification formalisms. A wide range of these methods is based on the deductive approach and/or deal with qualitative properties stating that a linear time formula holds with probability 0 or 1. See e.g. [LeSh82, HaSh84, Vard85, VaWo86, CoYa88, ACD91a] where methods for fully probabilistic systems are proposed and [HSP83, Pnue83, Vard85, PnZu86a, PnZu86b, VaWo86, PnZu93, CoYa95] where methods for concurrent probabilistic systems are presented.

Following the ω-automata approach proposed by Vardi & Wolper [Vard85, VaWo86] for verifying qualitative linear time properties, algorithms to establish quantitative linear time properties (and derived model checking algorithms for *PCTL**) have been developed by several authors (see [CoYa95, IyNa96] for the fully probabilistic case and [dAlf97b] where concurrent probabilistic systems and the standard interpretation $\models$ are considered). The basic idea behind the ω-automata theoretic approach can be sketched as follows. The starting point is a probabilistic system $\mathcal{S}$ and a *LTL* formula φ over AP. Using well-known methods [WVS83, SVW85, Safr88, VaWo94], one constructs an ω-automata A for the formula φ (i.e. an ω-automata over the alphabet 2^{AP} that accepts exactly those words over 2^{AP} for which the formula φ holds). Then, one defines a new probabilistic system $\mathcal{S} \times \mathsf{A}$ which can be viewed as the *product* of $\mathcal{S}$ and A and, for which, there is a natural

"embedding" $s \mapsto s_{\mathsf{A}}$ of the states s of $\mathcal{S}$ into the the state space of the product system $\mathcal{S} \times \mathsf{A}$. From the acceptance condition of A, a set U' of states in $\mathcal{S} \times \mathsf{A}$ can be derived such that the "probability" that φ holds in state s agrees with the "probability" for s_{A} to reach a state in U'.[24]

In the fully probabilistic case (where it is possible to deal with a *non-deterministic* ω-automaton), the time complexity of the ω-automata-based method is (single) exponential in the size of the system and linear in the size of the system, see [CoYa95, IyNa96]. An alternative algorithm (with the same time complexity) to compute the probabilities $p_s(\varphi) = Prob\{\pi \in Path_{ful}(s) : \pi \models \varphi\}$ in a finite fully probabilistic system is given by Courcoubetis & Yannakakis [CoYa88]. The main idea of this method is successively to remove the temporal operators from the given formula φ (finally resulting in a propositional formula φ') where at the same time the fully probabilistic systems is modified. As described in Section 9.2, both methods can be used for a $PCTL^*$ model checking algorithm with the same time complexity.

For the concurrent case, the above mentioned relation between the original system $\mathcal{S}$ and the product $\mathcal{S} \times \mathsf{A}$ requires that the ω-automaton A is *deterministic* (or at least "deterministic in limit" [VaWo86, CoYa95]). The time complexity of the resulting method for verifying concurrent probabilistic systems against quantitative LTL specifications (and the derived $PCTL^*$ model checking algorithm) with respect to the standard satisfaction relation $\models$ is double exponential in the size of the formula and linear in the size of the system. By the results of [CoYa95], this meets the lower bound for verifying concurrent probabilistic systems against linear time specifications.

In this section, we present methods for verifying concurrent probabilistic systems against quantitative LTL specifations when fairness assumptions are made. More precisely, we explain the ω-automaton approach can be applied for LTL model checking with respect to the satisfaction relations $\models_{fair}$, $\models_{sfair}$ and $\models_{Wfair}$. The method we present here is an adaption of the one developped by deAlfaro [dAlf97b] for a $PCTL^*$ model checking algorithm with respect to the standard satisfaction relation $\models$.

Remark 9.4.1 [Avoiding terminal states] The presented method assumes a finite concurrent probabilistic system *without* terminal states. This is a harmless restriction since any system can be transformed into an "equivalent" system without terminal states. Given a system $\mathcal{S} = (S, Steps, AP, \mathcal{L})$ with terminal states, we insert a special state $\mathbf{0}$ with a self-loop and transitions from any terminal state in $\mathcal{S}$ to $\mathbf{0}$.[25] Given a LTL formula φ, we replace each subformula $X\psi$ by $X(\psi \wedge \neg a_{\mathbf{0}})$, $\varphi_1 \mathcal{U}^{\leq k} \varphi_2$ by $\varphi_1 \mathcal{U}^{\leq k}(\varphi_2 \wedge \neg a_{\mathbf{0}})$ and each subformula $\varphi_1 \mathcal{U} \varphi_2$ by $\varphi_1 \mathcal{U}(\varphi_2 \wedge \neg a_{\mathbf{0}})$. Let φ' be the resulting LTL formula over AP'. It is easy to see that the interpretation of φ over $\mathcal{S}$ corresponds to the interpretation of φ' over $\mathcal{S}'$. Hence, we may assume w.l.o.g. that the system does not have terminal states. ■

Verifying ω-automaton specifications: As suggested by Luca deAlfaro, we consider

[24] Here, in the fully probabilistic case, "probability" stands for the usual probability measure; while in the concurrent case, "probability" stands for the minimal or maximal probability under a certain kind of adversaries.

[25] Formally, we consider the system $\mathcal{S}' = (S', Steps, AP', \mathcal{L}')$ where $S' = S \uplus \{\mathbf{0}\}$ and $AP' = AP \uplus \{a_{\mathbf{0}}\}$, $\mathcal{L}'(s) = \mathcal{L}(s)$ if $s \in S$ and $\mathcal{L}'(\mathbf{0}) = \{\mathbf{a_0}\}$. If $s \in S$ is nonterminal then $Steps'(s) = Steps(s)$. If $s \in S$ is terminal then $Steps'(s) = \{\mu_{\mathbf{0}}^1\}$. The self-loop at the auxiliary state $\mathbf{0}$ is modelled by $Steps'(\mathbf{0}) = \{\mu_{\mathbf{0}}^{\mathbf{1}}\}$.

ω-automaton with the Rabin acceptance condition. We briefly recall the definition. A *deterministic Rabin automaton* is a tuple $\mathsf{A} = (\mathsf{Q}, \mathsf{q}_0, \mathsf{Alph}, \mathsf{d}, \mathsf{AccCond})$ where

- Q is a finite set of states,
- $\mathsf{q}_0 \in \mathsf{Q}$ the initial state,
- Alph a nonempty finite alphabet,
- $\mathsf{d} : \mathsf{Q} \times \mathsf{Alph} \to \mathsf{Q}$ the transition function,
- $\mathsf{AccCond}$ the (Rabin) acceptance condition, i.e. $\mathsf{AccCond} = \{(\mathsf{H}_j, \mathsf{K}_j) : j = 1, \ldots, r\}$ is a set consisting of subsets H_j, K_j of Q.

A *run* over A is a "sequence" $\mathsf{p}_0 \stackrel{\mathsf{a}_0}{\to} \mathsf{p}_1 \stackrel{\mathsf{a}_1}{\to} \mathsf{p}_2 \stackrel{\mathsf{a}_2}{\to} \ldots$ such that $\mathsf{p}_0 = \mathsf{q}_0$ and $\mathsf{p}_{i+1} = \mathsf{d}(\mathsf{p}_i, \mathsf{a}_i)$, $i = 0, 1, 2, \ldots$. In what follows, we refer to an infinite sequence over Alph as a *word* over Alph. Each word $\mathbf{a} = \mathsf{a}_0\mathsf{a}_1 \ldots$ over Alph ia associated with the run

$$\mathsf{run}(\mathbf{a}) \;=\; \mathsf{p}_0 \stackrel{\mathsf{a}_0}{\to} \mathsf{p}_1 \stackrel{\mathsf{a}_1}{\to} \mathsf{p}_2 \stackrel{\mathsf{a}_2}{\to} \ldots$$

where $\mathsf{p}_0 = \mathsf{q}_0$ and $\mathsf{p}_i = \mathsf{d}(\mathsf{p}_{i-1}, \mathsf{a}_{i-1})$, $i = 1, 2, \ldots$. Let $\mathbf{q_a} = \mathsf{p}_0\mathsf{p}_1\mathsf{p}_2 \ldots$ be the associated sequence of (automata) states. The set $\mathsf{AccWords}(\mathsf{A})$ of *accepted words* over Alph is the set of words $\mathbf{a}$ over Alph such that, for the induced word $\mathbf{q_a} = \mathsf{q}_0\mathsf{q}_1 \ldots$ over Q,

$$\mathit{inf}(\mathbf{q}) \subseteq \mathsf{H}_j \text{ and } \mathit{inf}(\mathbf{q}) \cap \mathsf{K}_j \neq \emptyset \text{ for some } j \in \{1, \ldots, r\}.$$

Here, $\mathit{inf}(\mathbf{q})$ denotes the set of states $\mathsf{q} \in \mathsf{Q}$ that occur infinitely often in $\mathbf{q}$. In what follows, we fix a concurrent probabilistic system $\mathcal{S} = (S, \mathit{Steps}, AP, \mathcal{L})$ without terminal states and a deterministic Rabin automata $\mathsf{A} = (\mathsf{Q}, \mathsf{q}_0, 2^{AP}, \mathsf{d}, \mathsf{AccCond})$ over the alphabet $\mathsf{Alph} = 2^{AP}$. Let $\mathsf{AccCond} = \{(\mathsf{H}_j, \mathsf{K}_j) : j = 1, \ldots, r\}$.

Notation 9.4.2 [Accepted fulpaths] *If π is a fulpath in $\mathcal{S}$ then*

$$\mathsf{word}(\pi) \;=\; \mathcal{L}(\pi(0))\; \mathcal{L}(\pi(1))\; \mathcal{L}(\pi(2)) \ldots$$

denotes the induced word over $\mathsf{Alph} = 2^{AP}$. The set of accepted *fulpaths is defined by*

$$\mathit{AccPath} = \{\pi \in \mathit{Path}_{\mathit{ful}} : \mathsf{word}(\pi) \in \mathsf{AccWords}(\mathsf{A})\}.$$

For $A \in \mathcal{A}dv$, $s \in S$, we put $\mathit{AccPath}^A(s) = \mathit{AccPath} \cap \mathit{Path}^A_{\mathit{ful}}(s)$. Let $I \subseteq [0,1]$ be an interval of the form $I = I_{\bowtie p} = \{q \in [0,1] : q \bowtie p\}$. As before, $\mathcal{A}$ denotes a certain type of adversaries, e.g. $\mathcal{A} = \mathcal{A}dv$ or $\mathcal{A} = \mathcal{A}dv_{\mathit{fair}}$. We aim at a method for computing

$$\mathit{Sat}_{\mathcal{A}}(\langle \mathsf{A}, I_{\bowtie p}\rangle) \;=\; \left\{ s \in S : \mathit{Prob}(\mathit{AccPath}^A(s)) \;\bowtie p \;\text{ for all } A \in \mathcal{A} \right\}.$$

De Alfaro [dAlf97a, dAlf97b] describes a method for the case $\mathcal{A} = \mathcal{A}dv$. We now present a modification of this method for the cases $\mathcal{A} \in \{\mathcal{A}dv_{\mathit{fair}}, \mathcal{A}dv_{\mathit{sfair}}, \mathcal{A}dv_{\mathit{Wfair}}\}$. As in [dAlf97a, dAlf97b], we built the product of $\mathcal{S}$ and A, thus obtaining a new proposition-labelled concurrent probabilistic system $\mathcal{S} \times \mathsf{A}$.

Notation 9.4.3 [The distributions μ^{q}] *For $\mu \in \mathit{Distr}(S)$, $\mathsf{q} \in \mathsf{Q}$, we put*

$$\mu^{\mathsf{q}}(\langle t, \mathsf{p}\rangle) \;=\; \begin{cases} \mu(t) & : \; \mathit{if}\ \mathsf{p} = \mathsf{d}(\mathsf{q}, \mathcal{L}(t)) \\ 0 & : \; \mathit{otherwise.} \end{cases}$$

The steps in the product system are given by these "lifted" distributions $\mu^{\mathsf{q}} \in \mathit{Distr}(S \times \mathsf{Q})$.

Notation 9.4.4 [The product system $\mathcal{S} \times \mathsf{A}$] *The product system*

$$\mathcal{S} \times \mathsf{A} \;=\; (S_{\mathsf{A}}, \mathit{Steps}_{\mathsf{A}}, AP, \mathcal{L}_{\mathsf{A}})$$

is given by $S_{\mathsf{A}} = S \times \mathsf{Q}$, $\mathcal{L}_{\mathsf{A}}(\langle s, \mathsf{q}\rangle) = \mathcal{L}(s)$ *and* $\mathit{Steps}_{\mathsf{A}}(\langle s, \mathsf{q}\rangle) = \{\mu^{\mathsf{q}} : \mu \in \mathit{Steps}(s)\}$.

Clearly, $\mathcal{S} \times \mathsf{A}$ is a proposition-labelled concurrent probabilistic system. The original system $\mathcal{S}$ can be "embedded" into the product system by adding to each state $s \in S$ those automata state q that is reached from the initial automata state q_0 by the $\mathcal{L}(s)$-labelled transition.

Notation 9.4.5 [The state s_{A}] *For $s \in S$, let* $s_{\mathsf{A}} \;=\; \langle s, \mathsf{d}(\mathsf{q}_0, \mathcal{L}(s))\rangle$.

The resulting embedding $s \mapsto s_{\mathsf{A}}$ of the state space S of the original system into the state space of the product can be extended to an embedding of the paths. For this, we lift any path γ in $\mathcal{S}$ to a path γ_{A} in $\mathcal{S} \times \mathsf{A}$ as follows.

Notation 9.4.6 [The paths γ_{A}] *Let $\gamma = s_0 \xrightarrow{\mu_1} s_1 \xrightarrow{\mu_2} \ldots$ be a (finite or infinite) path in $\mathcal{S}$. We define γ_{A} to be the following path in $\mathcal{S}_{\mathsf{A}}$.*

$$\langle s_0, \mathsf{p}_0\rangle \xrightarrow{\nu_1} \langle s_1, \mathsf{p}_1\rangle \xrightarrow{\nu_2} \langle s_2, \mathsf{p}_2\rangle \xrightarrow{\nu_3} \ldots$$

where $s_i = \pi(i)$, $\mathsf{p}_0 = \mathsf{d}(\mathsf{q}_0, \mathcal{L}(s_0))$, $\mathsf{p}_{i+1} = \mathsf{d}(\mathsf{p}_i, \mathcal{L}(s_{i+1}))$ *and* $\nu_i = \mu_i^{\mathsf{p}_{i-1}}$.

Notation 9.4.7 [The sets Π_{A}] *Let $\Pi \subseteq \mathit{Path}^{\mathcal{S}}_{\mathit{ful}}$. Then,* $\Pi_{\mathsf{A}} \;=\; \{\pi_{\mathsf{A}} : \pi \in \Pi\}$.

The function $\gamma \mapsto \gamma_{\mathsf{A}}$ yields bijections $\mathit{Path}^{\mathcal{S}}_{\mathit{fin}}(s) \to \mathit{Path}^{\mathcal{S}\times\mathsf{A}}_{\mathit{fin}}(s_{\mathsf{A}})$ and $\mathit{Path}^{\mathcal{S}}_{\mathit{ful}}(s) \to \mathit{Path}^{\mathcal{S}\times\mathsf{A}}_{\mathit{ful}}(s_{\mathsf{A}})$ between the finite paths starting in s and s_{A} and the fulpaths starting in s and s_{A}. Clearly, $\pi \in \mathit{Fair}^{\mathcal{S}}$ iff $\pi_{\mathsf{A}} \in \mathit{Fair}^{\mathcal{S}\times\mathsf{A}}$. This also induces a connection between the (fair) adversaries of $\mathcal{S}$ and $\mathcal{S} \times \mathsf{A}$. Any adversary A for $\mathcal{S}$ induces a *set* of adversaries in $\mathcal{S} \times \mathsf{A}$. The adversaries of this set only differ in those paths σ' that do not start in a state of $\{s_{\mathsf{A}} : s \in S\}$.[26]

Notation 9.4.8 [The adversary set A_{A}] *Let $A \in \mathcal{A}dv^{\mathcal{S}}$. Then, A_{A} denotes the set of adversaries $A' \in \mathcal{A}dv^{\mathcal{S}\times\mathsf{A}}$ such that, for any finite path $\sigma \in \mathit{Path}^{\mathcal{S}}_{\mathit{fin}}$,*

$$\textit{if } A(\sigma) = \mu \textit{ and } \mathit{last}(\sigma_{\mathsf{A}}) = \langle s, \mathsf{q}\rangle \textit{ then } A'(\sigma_{\mathsf{A}}) = \mu^{\mathsf{q}}.$$

Clearly, the function $A \mapsto A_{\mathsf{A}}$ is injective and, for each $A' \in \mathcal{A}dv^{\mathcal{S}\times\mathsf{A}}$, there is a (unique) adversary A with $A' \in A_{\mathsf{A}}$. It is easy to see that, for any $A \in \mathcal{A}dv^{\mathcal{S}}$ and $A' \in A_{\mathsf{A}}$, the functions $\mathit{Path}^{A}_{\mathit{fin}}(s) \mapsto \mathit{Path}^{A'}_{\mathit{fin}}(s_{\mathsf{A}})$, $\sigma \mapsto \sigma_{\mathsf{A}}$, and $\mathit{Path}^{A}_{\mathit{ful}}(s) \mapsto \mathit{Path}^{A'}_{\mathit{ful}}(s_{\mathsf{A}})$, $\pi \mapsto \pi_{\mathsf{A}}$, are bijections. Moreover, we have $\mathbf{P}(\sigma) = \mathbf{P}(\sigma_{\mathsf{A}})$ for any finite path σ. This yields an isomorphism between the induced probability spaces on $\mathit{Path}^{A}_{\mathit{ful}}(s)$ and $\mathit{Path}^{A'}_{\mathit{ful}}(s_{\mathsf{A}})$. More precisely: Let $A \in \mathcal{A}dv^{\mathcal{S}}$, $A' \in A_{\mathsf{A}}$ and $\Pi \subseteq \mathit{Path}_{\mathit{ful}}$. Then, $\Pi^{A}(s)$ is measurable iff $\Pi^{A'}_{\mathsf{A}}(s_{\mathsf{A}})$ is measurable; in which case, the probability measures of $\Pi^{A}(s)$ and $\Pi^{A'}_{\mathsf{A}}(s_{\mathsf{A}})$ are the same. Moreover,

[26] Of course, we do not have a one-to-one correspondence between the adversaries of $\mathcal{S}$ and $\mathcal{S} \times \mathsf{A}$ since paths in $\mathcal{S} \times \mathsf{A}$ that do not start in a state s_{A} do not have a counterpart in $\mathcal{S}$.

- F is a (strictly) fair adversary for $\mathcal{S}$ iff there is a (strictly) fair adversary $F' \in F_{\mathsf{A}}$,
- F is W-fair iff there is $(W \times \mathsf{Q})$-fair adversary F' in F_{A}.

If we take $\Pi = AccPath$ then these observations lead to the following lemma.

Notation 9.4.9 [The set AccPath] *Let* AccPath *be the set of fulpaths* π_{A} *in* $\mathcal{S} \times \mathsf{A}$ *where* π *is an accepted fulpath in* $\mathcal{S}$. *I.e.* $\mathsf{AccPath} = AccPath_{\mathsf{A}}$.

Lemma 9.4.10 *Let* $p \in [0,1]$ *and* $\bowtie \in \{\geq, >, \leq, <\}$ *and* $s \in S$. *Then:*

(a) $Prob\left(AccPath^F(s)\right) \bowtie p$ *for all fair adversaries* F *of* $\mathcal{S}$

iff $Prob\left(\mathsf{AccPath}^{F'}(s_{\mathsf{A}})\right) \bowtie p$ *for all fair adversaries* F' *of* $\mathcal{S} \times \mathsf{A}$.

(b) $Prob\left(AccPath^F(s)\right) \bowtie p$ *for all strictly fair adversaries* F *of* $\mathcal{S}$

iff $Prob\left(\mathsf{AccPath}^{F'}(s_{\mathsf{A}})\right) \bowtie p$ *for all strictly fair adversaries* F' *of* $\mathcal{S} \times \mathsf{A}$.

(c) $Prob\left(AccPath^F(s)\right) \bowtie p$ *for all* W*-fair adversaries* F *of* $\mathcal{S}$

iff $Prob\left(\mathsf{AccPath}^{F'}(s_{\mathsf{A}})\right) \bowtie p$ *for all* $(W \times \mathsf{Q})$*-fair adversaries* F' *of* $\mathcal{S} \times \mathsf{A}$.

Proof: easy verification. Uses the above mentioned facts. ■

Next we show that, for any fair (strictly fair, W-fair) adversary F' of $\mathcal{S} \times \mathsf{A}$ and state $s \in S$, the probability $Prob\left(\mathsf{AccPath}^{F'}(s_{\mathsf{A}})\right)$ is given by $Prob\left\{\pi' \in Path^{F'}_{ful}(s_{\mathsf{A}}) : \pi' \models \Diamond a_{U'}\right\}$ where $a_{U'}$ is an atomic proposition that characterizes a certain set U' of states in $\mathcal{S} \times \mathsf{A}$. The definition of U' is derived from the acceptance condition in A and depends on the chosen satisfaction relation. When dealing with $\models_{fair}$ or $\models_{sfair}$, the definition of U' is quite simple:

Notation 9.4.11 [The sets U', H'_j and K'_j] *We define* $H'_j = S \times \mathsf{H}_j$, $K'_j = S \times \mathsf{K}_j$, $j = 1, \ldots, r$, *and*

$$U' = \bigcup_{1 \leq j \leq r} U'_j$$

where U'_j *is the largest subset of* H'_j *such that, for all* $u' \in U'_j$:

$$Reach^{\mathcal{S} \times \mathsf{A}}(u') \subseteq U'_j \quad \text{and} \quad Reach^{\mathcal{S} \times \mathsf{A}}(u') \cap K'_j \neq \emptyset.$$

We define $AP' = AP \cup \{a_{U'}\}$ where $a_{U'} \notin AP$ and extend the labelling function of $\mathcal{S} \times \mathsf{A}$ to a labelling function $S \times \mathsf{Q} \to 2^{AP'}$ (also called $\mathcal{L}_{\mathsf{A}}$) where $a_{U'} \in \mathcal{L}_{\mathsf{A}}(s')$ iff $s' \in U'$. In Section 9.5.3 (Lemma 9.5.43, page 253) we show that

$$Prob(\mathsf{AccPath}^{F'}(s')) = Prob\left\{\pi' \in Path^{F'}_{ful}(s') : \pi' \models \Diamond a_{U'}\right\}.$$

for all states $s' \in S \times \mathsf{Q}$ and fair adversaries F' of $\mathcal{S} \times \mathsf{A}$. Thus, part (a) and (b) of Lemma 9.4.10 (page 238) yield the following characterization of the states satisfying the quantitative ω-automaton specification $\langle \mathsf{A}, I_{\bowtie p} \rangle$ with respect to the adversary types $\mathcal{A}dv_{fair}$ and $\mathcal{A}dv_{sfair}$.

$$Sat_{fair}(\langle \mathsf{A}, I_{\bowtie p} \rangle) = \{s \in S : s_{\mathsf{A}} \models_{fair} \mathsf{Prob}_{\bowtie p}(\Diamond a_{U'})\},$$

$$Sat_{sfair}(\langle \mathsf{A}, I_{\bowtie p} \rangle) = \{s \in S : s_{\mathsf{A}} \models_{sfair} \mathsf{Prob}_{\bowtie p}(\Diamond a_{U'})\}.$$

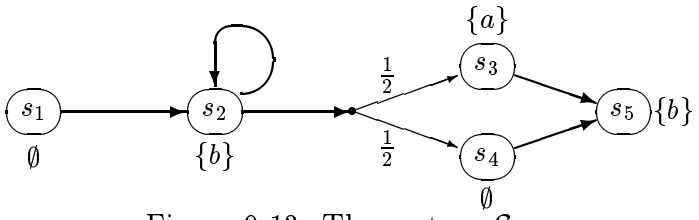


Figure 9.13: The system $\mathcal{S}$

Thus, the sets $Sat_{fair}(\langle \mathsf{A}, I_{\bowtie p}\rangle)$ and $Sat_{sfair}(\langle \mathsf{A}, I_{\bowtie p}\rangle)$ can be obtained by building the product $\mathcal{S} \times \mathsf{A}$, computing the set U' and then applying the $PCTL$ model checking algorithm of Section 9.3 to compute the sets $Sat_{fair}\,(\,\mathsf{Prob}_{\bowtie p}(\Diamond a_{U'})\,)$ and $Sat_{sfair}\,(\,\mathsf{Prob}_{\bowtie p}(\Diamond a_{U'})\,)$. For computing U', one might apply graph theoretical methods to obtain the sets U'_j. Alternatively, the set U'_j can be described as greatest fixed point of the operator $F_j : 2^{S \times \mathsf{Q}} \to 2^{S \times \mathsf{Q}}$,

$$F_j(V') \;=\; \left\{ v' \in H'_j : Reach^{\mathcal{S}\times\mathsf{A}}(v') \subseteq V' \;\wedge\; Reach^{\mathcal{S}\times\mathsf{A}}(v') \cap K'_j \neq \emptyset \right\},$$

and computed by the iteration $V'_0 = S \times \mathsf{Q}$, $V'_{i+1} = F_j(V'_i)$, $i = 0, 1, \ldots$. Dealing with W-fairness, similar ideas can be applied. The only difference is that the set U' has to be replaced by the following set U'_W. We define $W' = W \times \mathsf{Q}$ and $U' = \bigcup_{1 \le j \le r} U'_j$ where

$$U'_j \;=\; \bigcup_{T' \in \mathcal{T}_j} T'$$

and where $\mathcal{T}_j$ is defined as follows. $\mathcal{T}_j$ consists of all subsets T' of H'_j such that, for each $t' \in T' \setminus W'$, there is some $\nu_{t'} \in Steps_{\mathsf{A}}(t')$ where the following conditions are satisfied:

(1) If $t' \in T' \cap W'$ and $\nu \in Steps_{\mathsf{A}}(t')$, then $Supp(\nu) \subseteq T'$.
(2) If $t' \in T' \setminus W'$ then $Supp(\nu_{t'}) \subseteq T'$.
(3) Each state $t' \in T'$ can reach a state $v' \in K'_j$ in the system $(T', Steps')$ where

$$Steps'(t') = \begin{cases} Steps_{\mathsf{A}}(t') & : \text{ if } t' \in T' \cap W', \\ \{\nu_{t'}\} & : \text{ if } t' \in T' \setminus W'. \end{cases}$$

We state (without proof) that

$$Sat_{Wfair}(\langle \mathsf{A}, I_{\bowtie p}\rangle) \;=\; \left\{ s \in S : s_{\mathsf{A}} \;\models_{W'fair}\; \mathsf{Prob}_{\bowtie p}\left(\Diamond a_{U'_W}\right) \right\}.$$

Model checking for LTL**:** As before, we fix a finite concurrent probabilistic system $\mathcal{S} = (S, Steps, AP, \mathcal{L})$ without terminal states. Let $\langle \varphi, I\rangle$ be a quantitative LTL specification. Using well-known methods [WVS83, SVW85, Safr88, VaWo94], we can construct a deterministic Rabin automata A_φ over the alphabet $\mathsf{Alph} = 2^{AP}$ such that $\mathsf{AccWords}(\mathsf{A}_\varphi)$ is the set of infinite words over 2^{AP} for which φ holds.[27] For this, we need double exponential time in the size of φ. Then, we obtain $Sat_{\mathcal{A}}(\langle \varphi, I\rangle) \;=\; Sat_{\mathcal{A}}(\langle \mathsf{A}_\varphi, I\rangle)$ with the method explained before. The time complexity is polynomial in the size of $\mathcal{S}$ and double exponential in the size of φ. (Thus, the method is optimal by the results of [CoYa95]).

Example 9.4.12 We apply the method described above to the system shown in Figure 9.13 (page 239) and the quantitative LTL specification $\langle \Diamond(a \wedge Xb), I_{\geq 0.5}\rangle$. The deterministic Rabin automaton $\mathsf{A} = \mathsf{A}_{\Diamond(a \wedge Xb)}$ is shown in Figure 9.14 (page 240) where we deal

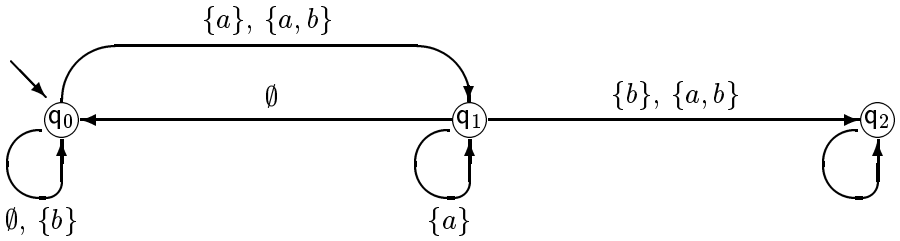


Figure 9.14: Rabin automata A for $\Diamond(a \wedge Xb)$

with the acceptance condition $\mathsf{AccCond} = \{(\mathsf{Q}, \{\mathsf{q}_2\})\}$.[28] The system $\mathcal{S} \times \mathsf{A}$ is shown in Figure 9.15 on page 240 where states that are not reachable from the state $\langle s_1, \mathsf{q}_0 \rangle$ are omitted. We get $H' = S \times \mathsf{Q}$ and $K' = \{(s_i, \mathsf{q}_2) : i = 1, \ldots, 5\}$. Thus, U' contains $\langle s_3, \mathsf{q}_0 \rangle$,

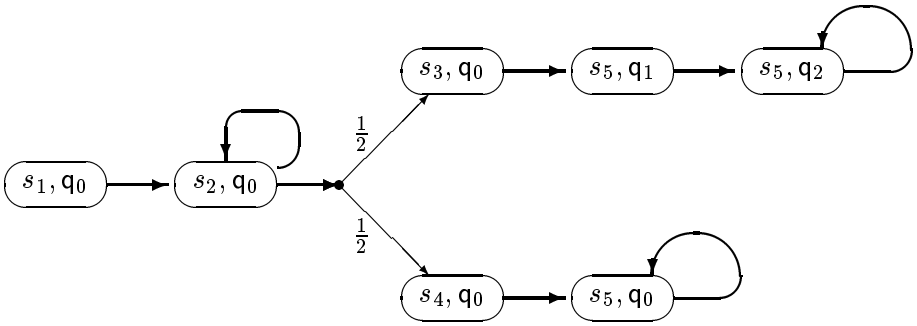


Figure 9.15: The product system $\mathcal{S} \times \mathsf{A}$

$\langle s_5, \mathsf{q}_1 \rangle$ and $\langle s_5, \mathsf{q}_2 \rangle$ but none of the other states shown in Figure 9.15. Clearly,

$$Prob\left\{\pi' \in Path^{F'}_{ful}(\langle s_1, \mathsf{q}_0 \rangle) : \pi' \models \Diamond a_{U'}\right\} \;=\; \tfrac{1}{2}$$

for each fair adversary F' of $\mathcal{S} \times \mathsf{A}$. Hence, $\langle s_1, \mathsf{q}_0 \rangle \;\models_{fair}\; \mathsf{Prob}_{\geq 0.5}(\Diamond(a \wedge Xb))$ which yields $s_1 \in Sat_{fair}(\langle \Diamond(a \wedge Xb), I_{\geq 0.5} \rangle)$. ■

Remark 9.4.13 By the results of [CoYa90, BidAl95], the minimal and maximal probabilities for *PCTL* path formulas under all adversaries agree with the minimal or maximal probabilities under all *simple* adversaries (see items (i) and (ii) on page 220). This result does not longer hold when dealing with general *LTL* formulas rather than *PCTL* path formulas. We consider the *LTL* formula $\varphi = Xb \rightarrow \Box a$ and the system shown in Figure

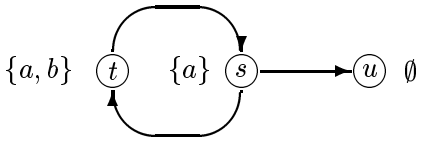


Figure 9.16: $p^A_s(Xb \rightarrow \Box a) = 1$ for all $A \in \mathcal{A}dv_{simple}$

9.16 (page 240). Then, $p^A_s(\varphi) = Prob\left\{\pi \in Path^A_{ful}(s) : \pi \models \varphi\right\} = 1$ for all simple adversaries A, while $p^F_s(\varphi) = Prob\left\{\pi \in Path^F_{ful}(s) : \pi \models \varphi\right\} = 0$ for the fair adversaries F with

[27] Here, the underlying satisfaction relation $\models \subseteq 2^{AP} \times I\!N \times LTL$ is defined in the obvious way.

[28] This ω-automata can be viewed as a deterministic Büchi automata where the acceptance set is $\{\mathsf{q}_2\}$.

$F(s) = \mu_t^1$ and $F(\sigma) = \mu_u^1$ for all paths σ with $last(\sigma) = s$ and $|\sigma| \geq 1$. This example also shows that, unlike in Theorem 9.3.6 (page 222), a result stating that the equality of $\sup\left\{p_s^F(\varphi) : F \in \mathcal{A}dv_{fair}\right\}$ and $\sup\left\{p_s^A(\varphi) : A \in \mathcal{A}dv\right\}$ cannot be established. ■

9.5 Proofs

This section includes the proofs of the theorems established in Section 9.3 and Section 9.4 which we have used to derive the model checking procedure for $PCTL^*$.

9.5.1 State and total fairness

In this section we introduce state and total fairness. State fairness is an instance of p-fairness (see Chapter 8, page 193 ff) which is defined in the fully probabilistic and concurrent case. Total fairness for concurrent probabilistic systems requires both fairness of adversaries (see Section 3.2.3, page 45 ff) and state fairness. State and total fairness are introduced for technical reasons only; they yield a simple proof technique for showing the equality of the probability measures of certain events. For instance, state fairness in fully probabilistic systems yields a simple proof for Lemma 3.1.10 (page 37) that gives a graph-theoretical criteria for establishing "qualitative progress properties".

Definition 9.5.1 [State fairness (fully probabilistic case)] *Let $\mathcal{S} = (S, \mathbf{P})$ be a fully probabilistic system and $\pi \in Path_{ful}^{\mathcal{S}}$. π is called* state fair *iff, for each $s \in inf(\pi)$, if $\mathbf{P}(s,t) > 0$ then there are infinitely many indices j with $\pi(j) = s$ and $\pi(j+1) = t$.*

Clearly, state fairness is a special instance of p-fairness (cf. Definition 8.1.1, page 194). We take $\mathsf{L} = S \times S$ and $\mathsf{l}(s,t) = \{(s,t)\}$. Then, for each fulpath π, π is state fair iff π is (L, l)-fair.

Lemma 9.5.2 *Let $\mathcal{S} = (S, \mathbf{P})$ be a finite fully probabilistic system and π a state fair fulpath in $\mathcal{S}$. Then, $inf(\pi) = Reach(s)$ for all states $s \in inf(\pi)$.*

Proof: Let $s \in inf(\pi)$. Clearly, $inf(\pi) \subseteq Reach(s)$. For $t \in Reach(s)$, let $dist(s,t)$ be the length of a shortest path from s to t. By induction on k it is easy to see that, if $dist(s,t) = k$ then $t \in inf(\pi)$. ■

Notation 9.5.3 [The set *StateFair*] *$StateFair^{\mathcal{S}}$ (or shortly StateFair) denotes the set of state fair fulpaths in $\mathcal{S}$.*

Lemma 9.5.4 *Let $(S, \mathbf{P})$ be a bounded fully probabilistic system. Then, for all $s \in S$,*

$$Prob\left(StateFair(s)\right) = 1.$$

In particular, whenever $\Pi \subseteq Path_{ful}$ such that $\Pi(s)$ is measurable then

$$Prob(\Pi(s)) \;=\; Prob\left(StateFair \cap \Pi(s)\right).$$

Proof: follows immediately from Theorem 8.1.5 (page 196). ■

Corollary 9.5.5 (cf. Lemma 3.1.10, page 37) *Let $(S, \mathbf{P})$ be a finite fully probabilistic system. Let U be a subset of S and Σ the set of finite paths σ where $\sigma(i) \in S \setminus U$, $i = 0, 1, \ldots, |\sigma| - 1$, and $last(\sigma) \in U$. Let $\Pi = \Sigma\uparrow$. Let $s \in S$ and $T = \{last(\sigma) : \sigma \in Path_{fin}(s), \sigma \notin \Sigma\uparrow_{fin}\}$. Then, we have:*

$$\Sigma(t) \neq \emptyset \text{ for all states } t \in T \text{ iff } Prob(\Pi(s)) = 1.$$

Proof: The "if" part is clear. For the "only if" part, we assume that $\Sigma(t) \neq \emptyset$ for all states $t \in T$. Using Lemma 9.5.2 (page 241) it is easy to see that $StateFair(s) \subseteq \Pi(s)$. Thus, Lemma 9.5.4 (page 241) yields the claim. ■

The definition of state fairness in the concurrent case is as follows.

Definition 9.5.6 [State fairness (concurrent case)] *Let $\mathcal{S} = (S, Steps)$ be a concurrent probabilistic system and π a fulpath in $\mathcal{S}$. π is called* state fair *iff, for each $s \in S$ and $\mu \in Steps(s)$ such that $\pi(i) = s$, $step(\pi, i) = \mu$ for infinitely many i and each $t \in Supp(\mu)$, there are infinitely many indices j with $\pi(j) = s$, $step(\pi, j) = \mu$ and $\pi(j+1) = t$.*

Note that state fairness is a special instance of p-fairness (cf. Definition 8.2.1, page 200). Let $\mathsf{L} = \{(s, \mu, t) : s \in S, \mu \in Steps(s), t \in Supp(\mu)\}$ and $\mathsf{l}(s, \mu, t) = \{(s, \mu, t)\}$. Then, for each fulpath π, π is state fair iff π is (L, l)-fair. As in the fully probabilistic case, $StateFair^{\mathcal{S}}$ (or shortly $StateFair$) denotes the set of state fair fulpaths in $\mathcal{S}$.

Lemma 9.5.7 *Let $\mathcal{S} = (S, Steps)$ be a finite concurrent probabilistic system, A a simple adversary for $\mathcal{S}$. Then, for each $\pi \in StateFair^A$, $Reach^A(s) = inf(\pi)$ for all $s \in inf(\pi)$.*

Proof: follows immediately by Lemma 9.5.2 (page 241) applied to the finite fully probabilistic system $\mathcal{S}^A$ induced by the simple adversary A. ■

Lemma 9.5.8 *Let $(S, Steps)$ be a finite concurrent probabilistic system. Then:*

$$Prob\left(StateFair^A(s)\right) = 1$$

for all adversaries A and $s \in S$. In particular, whenever $\Pi \subseteq Path_{ful}$ such that $\Pi^A(s)$ is measurable then $Prob\left(\Pi^A(s)\right) = Prob\left(StateFair \cap \Pi^A(s)\right)$.

Proof: follows immediately from Theorem 8.2.3 (page 200). ■

We define total fairness as the combination of state fairness and fairness with respect to the non-deterministic choices (in the sense of Definition 3.2.14, page 45).

Definition 9.5.9 [Total fairness] *Let $\mathcal{S} = (S, Steps)$ be a concurrent probabilistic system and π a fulpath in $\mathcal{S}$. π is called* total fair *iff π is fair and state fair.*

Lemma 9.5.10 *Let $\mathcal{S} = (S, Steps)$ be a finite concurrent probabilistic system. Then, for each total fair fulpath π in $\mathcal{S}$, $Reach(s) = inf(\pi)$ for all $s \in inf(\pi)$.*

Proof: easy verification. Uses induction on the "distance" between two states as in the proof of Lemma 9.5.2 (page 241). ■

Notation 9.5.11 [The set *TotalFair*] *$TotalFair^{\mathcal{S}}$ (or shortly $TotalFair$) denotes the set of total fair fulpaths.*

Lemma 9.5.12 *Let $(S, Steps)$ be a finite concurrent probabilistic system. Then:*

$$Prob\left(TotalFair^F(s)\right) = 1$$

for all fair adversaries F and $s \in S$. In particular, if $\Pi \subseteq Path_{ful}$ such that $\Pi^F(s)$ is measurable then $Prob\left(\Pi^F(s)\right) = Prob\left(TotalFair \cap \Pi^F(s)\right)$.

Proof: follows immediately from Lemma 9.5.8 (page 242). ■

9.5.2 Correctness of the $PCTL$ model checking algorithm

We fix a proposition-labelled concurrent probabilistic system $\mathcal{S} = (S, Steps, AP, \mathcal{L})$, a subset W of S and two $PCTL$ formulas Φ_1 and Φ_2 which we treat as atomic propositions (i.e. we assume that $\Phi_1, \Phi_2 \in AP$). Moreover, we assume atomic propositions $a^?$, a^+ and a^0_W as in Notation 9.3.22 (page 227) and Notation 9.3.32 (page 229). We often use the following lemma which follows from the results of [BidAl95] (Corollary 20, part 1, in [BidAl95]), cf. item (i) and (ii) on page 220.

Lemma 9.5.13 (cf. [BidAl95]) *There exist A^{max}, $A^{min} \in \mathcal{A}dv_{simple}$ with*

$$p_s^{A^{max}}(\Phi_1 \mathcal{U} \Phi_2) \;\geq\; p_s^B(\Phi_1 \mathcal{U} \Phi_2) \;\geq\; p_s^{A^{min}}(\Phi_1 \mathcal{U} \Phi_2)$$

for all states $s \in S$ and all adversaries B. In particular,

$$p_s^{A^{max}}(\Phi_1 \mathcal{U} \Phi_2) \;=\; p_s^{max}(\Phi_1 \mathcal{U} \Phi_2), \quad p_s^{A^{min}}(\Phi_1 \mathcal{U} \Phi_2) \;=\; p_s^{min}(\Phi_1 \mathcal{U} \Phi_2).$$

Maximal probabilities under all fair adversaries: We give the proof of Theorem 9.3.6 (page 222) and Theorem 9.3.8 (page 222).

Lemma 9.5.14 *Let $(S, Steps)$ be a finite concurrent probabilistic system and S_1, $S_2 \subseteq S$. Let A be a simple adversary for $\mathcal{S}$ and $\Sigma \subseteq Path^A_{fin}$ be the set of all fulpaths σ such that $\sigma(i) \in S_1 \setminus S_2$, $i = 0, 1, \ldots, |\sigma| - 1$, and $last(\sigma) \in S_2$. Then, we have:*

If $\pi \in StateFair$ then there are only finitely many indices i such that $\pi^{(i)} \in \Sigma\downarrow$.

Proof: We assume that there is a fulpath $\pi \in StateFair$ such that $\pi^{(i)} \in \Sigma\downarrow$ for infinitely many i. Then, $\pi^{(i)} \in \Sigma\downarrow$ for all i. Hence, $\pi \in Path^A_{ful}$. Thus, $\pi \in StateFair^A$. By Lemma 9.5.7 (page 242),

(*) $\inf(\pi) = Reach^A(s)$.

By definition of Σ and since $\pi^{(i)} \in \Sigma\downarrow$ for infinitely many (all) i, we have $\pi(i) \in S_1 \setminus S_2$ and $Reach^A(\pi(i)) \cap S_2 \neq \emptyset$ for all i. This contradicts (*). Thus, we get the claim. ■

Lemma 9.5.15 *Let $(S, Steps)$ be a finite concurrent probabilistic system and S_1, $S_2 \subseteq S$. Let $\Sigma \subseteq Path_{fin}$ be the set of all fulpaths σ such that $\sigma(i) \in S_1 \setminus S_2$, $i = 0, 1, \ldots, |\sigma| - 1$, and $last(\sigma) \in S_2$. Then:*

For each simple adversary A, there exists a fair adversary F with $\Sigma^A \subseteq Path^F_{fin}$.

In particular, $\Sigma^A \subseteq \Sigma^F$ *and*

$$Prob\left(\Sigma^A(s)\uparrow\right) \;\leq\; Prob\left(\Sigma^F(s)\uparrow\right)$$

for all states $s \in S$*. Moreover, there is a sequence* $(F_k)_{k\geq 0}$ *of strictly fair adversaries such that*

$$Prob\left(\Sigma^A(s)\uparrow\right) \;\leq\; \sup_{k\geq 0} Prob\left(\Sigma^{F_k}(s)\uparrow\right)$$

for all states $s \in S$*.*

Proof: Let A be a simple adversary. Let Γ a subset of Σ^A. We define an adversary F_Γ as follows. For each state $s \in S$, we choose an enumeration $\nu_0^s, \ldots, \nu_{m_s-1}^s$ of $Steps(s)$.

- If $\gamma \in Path_{fin}$ is a proper prefix of some $\sigma \in \Gamma$, i.e. if $\gamma = \sigma^{(i)}$ for some $\sigma \in \Gamma$ and some integer i with $i < |\sigma|$ then we define $F_\Gamma(\gamma) \;=\; step(\sigma, i)$.
- If $\gamma \in Path_{fin}$ is not a proper prefix of some $\sigma \in \Gamma$ then we define $F_\Gamma(\gamma) \;=\; \nu_j^s$ where $s = last(\gamma)$, $j = r \bmod m_s$ and r the number of indices $i < |\gamma|$ with $\gamma(i) = s$.

Here, mod denotes the "modulo-division" function. Clearly, $\Gamma \subseteq Path_{fin}^{F_\Gamma}$. Thus,

(1) $Prob(\Gamma(s)\uparrow) \;\leq\; Prob(\Sigma^{F_\Gamma}(s)\uparrow)$

It is easy to see that,

(2) if $\pi \in Path_{ful}^{F_\Gamma}$ is not fair then, for each i, $\pi^{(i)}$ is a proper prefix of some $\sigma_i \in \Gamma$.

Clearly, (2) yields that, if Γ is finite then F_Γ is strictly fair. Thus, dealing with the sequence $(F_k)_{k\geq 0}$ where $F_k = F_{\Gamma_k}$ and $\Gamma_k = \{\sigma \in \Sigma^A : |\sigma| \leq k\}$, we get: F_k is strictly fair and

$$\sup_{k\geq 0} Prob\left(\Sigma^{F_k}(s)\uparrow\right) \;\geq\; \sup_{k\geq 0} Prob\left(\Gamma_k(s)\uparrow\right) \;=\; Prob\left(\Sigma^A(s)\uparrow\right).$$

Here, we use (1), the fact that $\bigcup_{k\geq 0} \Gamma_k(s) \;=\; \Sigma^A(s)$ and that Γ_k is finite.[29]

Next we consider $F = F_{\Sigma^A}$. Since $\Sigma^A \subseteq Path_{fin}^F$ we have

$$Prob\left(\Sigma^A(s)\uparrow\right) \;\leq\; Prob\left(\Sigma^F(s)\uparrow\right)$$

for all $s \in S$. Next we show that F is fair. Lemma 9.5.14 (page 243) yields:

(3) If $\pi \in StateFair^F$ then there are only finitely many indices i with $\pi^{(i)} \in \Sigma^A\uparrow$.

By (2) and (3), we get $StateFair^F \subseteq Fair^F$. By Lemma 9.5.8 (page 242),

$$Prob\left(Fair^F(s)\right) \;=\; 1$$

for all $s \in S$. Thus, F is fair. ■

Remark 9.5.16 In Lemma 9.5.15 (page 243) we cannot ensure the existence of a strictly fair adversary F with $\Sigma^A \subseteq Path_{fin}^F$ (unless Σ^A is finite). For instance, consider the system

[29] The finiteness of Γ_k can be seen as follows. Recall that A is simple and $(S, Steps)$ finite. Thus, the fully probabilistic system associated with A is finite, and hence, the set of all finite paths in A up to a fixed length k is finite.

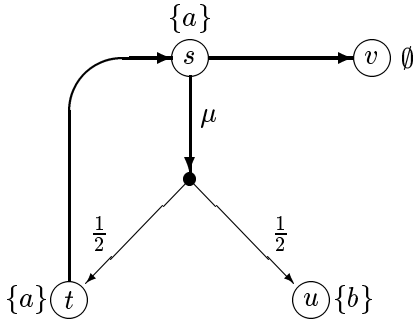


Figure 9.17:

shown in Figure 9.17 (page 245) and the simple adversary A with $A(s) = \mu$. Then, with $S_1 = Sat(a) = \{s, t\}$, $S_2 = Sat(b) = \{u\}$, we have:

$$\Sigma^A = \{\sigma \in Path^A_{fin} : last(\sigma) = u, \sigma(i) \neq u,\ i = 0, 1, \ldots |\sigma| - 1\}.$$

Then, for each adversary F with $\Sigma^A \subseteq Path^F_{fin}$: if $last(\sigma) = s$ then $F(\sigma) = \mu$. Hence, F contains the unfair fulpath $s \xrightarrow{\mu} t \xrightarrow{\mu_s} s \xrightarrow{\mu} t \xrightarrow{\mu_s} \ldots$, i.e. F cannot be strictly fair. ■

Lemma 9.5.17 *For each $A \in \mathcal{A}dv_{simple}$ there exist*

(a) $F \in \mathcal{A}dv_{fair}$ with $p^A_s(\Phi_1 \mathcal{U} \Phi_2) \leq p^F_s(\Phi_1 \mathcal{U} \Phi_2)$ for all $s \in S$
(b) a sequence $(F_k)_{k \geq 1}$ in $\mathcal{A}dv_{sfair}$ such that, for all $s \in S$,

$$p^A_s(\Phi_1 \mathcal{U} \Phi_2) \leq \sup_{k \geq 1} p^{F_k}_s(\Phi_1 \mathcal{U} \Phi_2).$$

Proof: follows immediately by Lemma 9.5.15 (page 243). ■

Corollary 9.5.18 *For all $s \in S$:*

$$\max\left\{p^F_s(\Phi_1 \mathcal{U} \Phi_2) : F \in \mathcal{A}dv_{fair}\right\} = \max\left\{p^F_s(\Phi_1 \mathcal{U} \Phi_2) : F \in \mathcal{A}dv_{Wfair}\right\} = p^{max}_s(\Phi_1 \mathcal{U} \Phi_2).$$

Proof: follows immediately by Lemma 9.5.13 (page 243), part (a) of Lemma 9.5.17 (page 245) and the fact that $\mathcal{A}dv_{fair} \subseteq \mathcal{A}dv_{Wfair}$. ■

Theorem 9.5.19 (cf. Theorem 9.3.6, page 222, and Theorem 9.3.7, page 222)

$$s \models_{fair} \mathsf{Prob}_{\sqsubseteq p}(\Phi_1 \mathcal{U} \Phi_2) \ \textit{iff}\ s \models_{Wfair} \mathsf{Prob}_{\sqsubseteq p}(\Phi_1 \mathcal{U} \Phi_2) \ \textit{iff}\ p^{max}_s(\Phi_1 \mathcal{U} \Phi_2) \sqsubseteq p.$$

Proof: follows immediately by Corollary 9.5.18 (page 245). ■

Corollary 9.5.20 *For all $s \in S$:* $\sup\left\{p^F_s(\Phi_1 \mathcal{U} \Phi_2) : F \in \mathcal{A}dv_{sfair}\right\} = p^{max}_s(\Phi_1 \mathcal{U} \Phi_2)$.

Proof: by Lemma 9.5.13 (page 243) and part (b) of Lemma 9.5.17 (page 245). ■

Theorem 9.5.21 (cf. Theorem 9.3.8, page 222) *For all $s \in S$:*

$$s \models_{sfair} \mathsf{Prob}_{\leq p}(\Phi_1 \mathcal{U} \Phi_2) \ \textit{iff}\ p^{max}_s(\Phi_1 \mathcal{U} \Phi_2) \leq p.$$

Proof: follows by Corollary 9.5.20 (page 245). ■

Minimal probabilities under all fair adversaries: We give the proof of Theorem 9.3.23 (page 227) and Theorem 9.3.26 (page 228).

Lemma 9.5.22 *Let π be a total fair fulpath. Then, $\pi \models \Phi_1 \mathcal{U} \Phi_2$ iff $\pi \not\models a^? \mathcal{U} \neg a^+$.*

Proof: Clearly, if $\pi \models \Phi_1 \mathcal{U} \Phi_2$ then $\pi \not\models a^? \mathcal{U} \neg a^+$. Let $\pi \not\models a^? \mathcal{U} \neg a^+$. Then,

(*) $\pi(i) \in S^+(\Phi_1, \Phi_2) \subseteq Sat(\Phi_1)$ for all $i \geq 0$.

Hence, $inf(\pi) \subseteq S^+(\Phi_1, \Phi_2)$. Let $s \in inf(\pi)$. By Lemma 9.5.10 (page 242), $Reach(s) \subseteq inf(\pi)$. By definition of $S^+(\Phi_1, \Phi_2)$ (and since $s \in S^+(\Phi_1, \Phi_2)$), $Reach(s) \cap Sat(\Phi_2) \neq \emptyset$. Thus, $Sat(\Phi_2) \cap inf(\pi) \neq \emptyset$. From this and (*), we get $\pi \models \Phi_1 \mathcal{U} \Phi_2$. ■

Corollary 9.5.23 *For all $F \in \mathcal{A}dv_{fair}$, $s \in S$: $p_s^F(\Phi_1 \mathcal{U} \Phi_2) = 1 - p_s^F(a^? \mathcal{U} \neg a^+)$.*

Proof: follows from Lemma 9.5.22 (page 246) and Lemma 9.5.12 (page 243). ■

Corollary 9.5.24 *For all $s \in S$:*

$$\min \left\{ p_s^F(\Phi_1 \mathcal{U} \Phi_2) : F \in \mathcal{A}dv_{fair} \right\} = 1 - p_s^{max}(a^? \mathcal{U} \neg a^+).$$

Proof: If $F \in \mathcal{A}dv_{fair}$ then $p_s^F(\Phi_1 \mathcal{U} \Phi_2) = 1 - p_s^F(a^? \mathcal{U} \neg a^+) \geq 1 - p_s^{max}(a^? \mathcal{U} \neg a^+)$ (by Corollary 9.5.23). By Corollary 9.5.18 (page 245), $p_s^F(a^? \mathcal{U} \neg a^+) = p_s^{max}(a^? \mathcal{U} \neg a^+)$ for some $F \in \mathcal{A}dv_{fair}$. For this adversary F, we get (again by Corollary 9.5.23),

$$p_s^F(\Phi_1 \mathcal{U} \Phi_2) = 1 - p_s^F(a^? \mathcal{U} \neg a^+) = 1 - p_s^{max}(a^? \mathcal{U} \neg a^+).$$

This yields the claim. ■

Theorem 9.5.25 (cf. Theorem 9.3.23, page 227) *For all $s \in S$:*

$$s \models_{fair} \mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U} \Phi_2) \text{ iff } 1 - p_s^{max}(a^? \mathcal{U} \neg a^+) \sqsupseteq p.$$

Proof: follows immediately by Corollary 9.5.24 (page 246). ■

Corollary 9.5.26 *For all $s \in S$:*

$$\inf \left\{ p_s^F(\Phi_1 \mathcal{U} \Phi_2) : F \in \mathcal{A}dv_{sfair} \right\} = 1 - p_s^{max}(a^? \mathcal{U} \neg a^+).$$

Proof: follows by Corollary 9.5.23 (page 246) and Corollary 9.5.18 (page 245). ■

Theorem 9.5.27 (cf. Theorem 9.3.26, page 228) *For all $s \in S$:*

$$s \models_{sfair} \mathsf{Prob}_{\geq p}(\Phi_1 \mathcal{U} \Phi_2) \text{ iff } 1 - p_s^{max}(a^? \mathcal{U} \neg a^+) \geq p.$$

Proof: follows immediately by Corollary 9.5.26 (page 246). ■

Maximal and minimal probabilities under all strictly fair adversaries: We now give the proof of Theorem 9.3.16 (page 226) and Theorem 9.3.28 (page 228).

Lemma 9.5.28 *Let $F \in \mathcal{A}dv_{fair}$, $\Pi = \{\pi \in Path^F_{ful} : \pi \models \Phi_1 \mathcal{U} \Phi_2\}$ and $\Pi_k = \bigcup_{\lambda \in \Lambda_k} \lambda \uparrow^F$ where $\Lambda_k = \left\{\pi^{(k)} : \pi \in \Pi\right\}$ Then, for all $s \in S$:*

$$p^F_s(\Phi_1 \mathcal{U} \Phi_2) \;=\; \lim_{k \to \infty} \; Prob(\Pi_k(s))$$

Proof: We have $\Pi_0 \supseteq \Pi_1 \supseteq \ldots \supseteq \Pi$. Let $\Pi' = \bigcap_{k \geq 1} \Pi_k$. Then, $\Pi'(s)$ is measurable and $\Pi'(s) \supseteq \Pi(s)$. Hence,

$$p^F_s(\Phi_1 \mathcal{U} \Phi_2) \;=\; Prob(\Pi(s)) \;\leq\; Prob(\Pi'(s)) \;=\; \lim_{k \to \infty} Prob(\Pi_k(s)).$$

Using Lemma 9.5.10 (page 242) it can be shown that $TotalFair \cap \Pi'(s) \subseteq \Pi(s)$. By Lemma 9.5.12 (page 243):

$$Prob(\Pi'(s)) \;=\; Prob(TotalFair^F(s) \cap \Pi'(s)) \;\leq\; Prob(\Pi(s)) \;=\; p^F_s(\Phi_1 \mathcal{U} \Phi_2).$$

Hence, $p^F_s(\Phi_1 \mathcal{U} \Phi_2) = Prob(\Pi'(s)) = \lim\; Prob(\Pi_k(s))$. ■

Notation 9.5.29 [The probabilities $p^A_\sigma(\Phi_1 \mathcal{U} \Phi_2)$] *Let $A \in \mathcal{A}dv$, $\sigma \in Path^A_{fin}$. Then,*

$$p^A_\sigma(\Phi_1 \mathcal{U} \Phi_2) \;=\; p^{A'}_\sigma(\Phi_1 \mathcal{U} \Phi_2)$$

where A' is an adversary with $A'(\gamma) = A(\sigma \circ \gamma)$ for each $\gamma \in Path_{fin}(last(\sigma))$.

Lemma 9.5.30 *Let $F \in \mathcal{A}dv_{sfair}$, $s \in S$ and $\Sigma = \{\sigma \in Path^A_{fin}(s) : \sigma(i) \;\models\; \Phi_1 \wedge \neg\Phi_2, i = 0, 1, \ldots, |\sigma|\}$. The following are equivalent:*

(i) $F(\sigma) \in MaxSteps(last(\sigma), \Phi_1, \Phi_2)$ *for all* $\sigma \in \Sigma$.
(ii) $p^{max}_s(\Phi_1 \mathcal{U} \Phi_2) \;=\; p^F_s(\Phi_1 \mathcal{U} \Phi_2)$.
(iii) $p^{max}_{last(\sigma)}(\Phi_1 \mathcal{U} \Phi_2) \;=\; p^F_\sigma(\Phi_1 \mathcal{U} \Phi_2)$ *for all* $\sigma \in \Sigma$.

Proof: For simplicity, we omit the argument $\Phi_1 \mathcal{U} \Phi_2$ and shortly write p^F_σ and p^{max}_s rather than $p^F_\sigma(\Phi_1 \mathcal{U} \Phi_2)$ and $p^{max}_s(\Phi_1 \mathcal{U} \Phi_2)$. The implication (iii) $\Longrightarrow$ (ii) is obvious.

(ii) $\Longrightarrow$ (iii): We suppose $p^F_{\sigma_0} < p^{max}_{last(\sigma_0)}$ for some $\sigma_0 \in \Sigma$. Let B be an adversary with $p^B_{last(\sigma_0)} = p^{max}_{last(\sigma_0)}$ (which exists by Lemma 9.5.13, page 243). Let A be the adversary given by: $A(\lambda) = F(\lambda)$ if $\sigma_0 \not\leq_{prefix} \lambda$ and $A(\sigma_0 \circ \gamma) = B(\gamma)$ if $last(\sigma_0) = first(\gamma)$. Then,

(*) $p^A_{\sigma_0} \;=\; p^B_{last(\sigma_0)} \;=\; p^{max}_{last(\sigma_0)} \;>\; p^F_{\sigma_0}$.

Let $k = |\sigma_0|$ and $\Sigma_k \;=\; \{\sigma \in \Sigma : |\sigma| = k\}$. Let Λ_k the set of paths $\lambda \in Path^F_{fin}(s)$ with

- $|\lambda| \leq k$,
- $\lambda^{(l)} \models \Phi_1 \wedge \neg\Phi_2$, $l = 0, 1, \ldots, |\lambda| - 1$,
- $last(\lambda) \;\models\; \Phi_2$.

Clearly, $\Sigma_k, \Lambda_k \subseteq Path^A_{fin}(s)$. Moreover, by definition of A, $p^A_\sigma \;=\; p^F_\sigma$ for all $\sigma \in \Sigma_k \setminus \{\sigma_0\}$. Since $\sigma_0 \in \Sigma_k$ we obtain by (*):

$$p^F_s \;=\; \sum_{\sigma \in \Sigma_k} \mathbf{P}(\sigma) \cdot p^F_\sigma \;+\; \sum_{\lambda \in \Lambda_k} \mathbf{P}(\lambda) \;<\; \sum_{\sigma \in \Sigma_k} \mathbf{P}(\sigma) \cdot p^A_\sigma \;+\; \sum_{\lambda \in \Lambda_k} \mathbf{P}(\lambda) \;=\; p^A_s \;\leq\; p^{max}_s$$

Contradiction.

(iii) $\Longrightarrow$ (i): If $\sigma \in \Sigma$, $s = last(\sigma)$ and $\mu = F(\sigma)$ then

$$p_s^{max} \;=\; p_\sigma^F \;=\; \sum_{t \in S} \mu(t) \cdot p_{\sigma_t}^F \;=\; \sum_{t \in S} \mu(t) \cdot p_t^{max}$$

where σ_t is the path $\sigma \stackrel{\mu}{\to} t$. Hence, $\mu \in MaxSteps(s, \Phi_1, \Phi_2)$.

(i) $\Longrightarrow$ (iii): We define $\Pi = \{\pi \in Path_{ful}^F : \pi \models \Phi_1 \mathcal{U} \Phi_2\}$ and Let $\sigma \in \Sigma$ and $\Pi(\sigma)$ be the set of fulpaths $\pi \in Path_{ful}(last(\sigma))$ where $\sigma \circ \pi \in \Pi^F$ and

$$\Lambda_k(\sigma) \;=\; \left\{\pi^{(k)} : \pi \in \Pi(\sigma)\right\}, \quad \Lambda(\sigma) \;=\; \bigcup_{k \geq 0} \Lambda_k(\sigma).$$

Lemma 9.5.28 (page 247) applied to the fair adversary F' with $F'(\lambda) = F(\sigma \circ \lambda)$ if $first(\lambda) = last(\sigma)$ yields

$$p_\sigma^F \;=\; \lim_{k \to \infty} p_\sigma^k \quad \text{where} \quad p_\sigma^k \;=\; \sum_{\lambda \in \Lambda_k(\sigma)} \mathbf{P}(\lambda).$$

By induction on k it can be shown that $p_\sigma^k \geq p_{last(\sigma)}^{max}$ for all $\sigma \in \Sigma$. This yields $p_\sigma^F \geq p_{last(\sigma)}^{max}$ for all $\sigma \in \Sigma$. Hence, $p_\sigma^F = p_{last(\sigma)}^{max}$ for all $\sigma \in \Sigma$. ■

Lemma 9.5.31 *There exists $F \in \mathcal{A}dv_{sfair}$ with $p_s^F(\Phi_1 \mathcal{U} \Phi_2) = p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$ for all $s \in T^{max}(\Phi_1, \Phi_2)$.*

Proof: We simplify the notations introduced in Notation 9.3.14 (page 224) and write T^{max}, T_j^{max} and $T_{j,l}^{max}$ rather than $T^{max}(\Phi_1, \Phi_2)$, $T_j^{max}(\Phi_1, \Phi_2)$ and $T_{j,l}^{max}(\Phi_1, \Phi_2)$. Let

$$T \;=\; \bigcup_{j \geq 1} T_{j,1}^{max}.$$

For each $j \geq 1$, $t \in T_{j,1}^{max}$ we choose some $\mu_t \in MaxSteps(t, \Phi_1, \Phi_2)$ with

$$Supp(\mu_t) \;\subseteq\; \bigcup_{i<j} T_i^{max}.$$

We define an adversary F as follows. For each $s \in S$, let $\nu_0^s, \ldots, \nu_{m_s-1}^s$ be an enumeration of $Steps(s)$ (and m_s the cardinality of $Steps(s)$). For $\sigma \in Path_{fin}$, let $\eta(\sigma)$ be the number of indices $i < |\sigma|$ with $\sigma(i) = last(\sigma)$. Let

$$\Sigma \;=\; \{\sigma \in Path_{fin} : \sigma(i) \;\models\; \Phi_1 \wedge \neg\Phi_2, i = 0, 1, \ldots, |\sigma|\}.$$

We define

$$F(\sigma) \;=\; \begin{cases} \nu_j^s & : \text{ if } s = last(\sigma),\ j = \eta(\sigma) \bmod m_s \text{ and either } \sigma \notin \Sigma \text{ or } s \in S \setminus T \\ \mu_t & : \text{ if } t = last(\sigma) \in T \text{ and } \sigma \in \Sigma \end{cases}$$

It is easy to see that F is strictly fair. By definition of F it is immediately clear that

- $\{last(\sigma) : \sigma \in \Sigma^F(t)\} \;\subseteq\; T^{max}$ for all $t \in T^{max}$,
- $F(\sigma) \in MaxSteps(last(\sigma), \Phi_1, \Phi_2)$ for all $\sigma \in \Sigma$ with $last(\sigma) \in T^{max}$.

By Lemma 9.5.30 (page 247), we get $p_t^F(\Phi_1 \mathcal{U} \Phi_2) = p_t^{max}(\Phi_1 \mathcal{U} \Phi_2)$ for all $t \in T^{max}$. ■

Corollary 9.5.32 *There exists $F \in \mathcal{A}dv_{sfair}$ with $p_s^F(\Phi_1 \mathcal{U} \Phi_2) = 1 - p_s^{max}(a^? \mathcal{U} \neg a^+)$ for all $s \in T^{max}(a^? \mathcal{U} \neg a^+)$.*

Proof: follows from Lemma 9.5.31 (page 248) and Corollary 9.5.26 (page 246). ■

Lemma 9.5.33 *If $F \in \mathcal{A}dv_{sfair}$, $s \notin T^{max}(\Phi_1, \Phi_2)$ then $p_s^F(\Phi_1 \mathcal{U} \Phi_2) < p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$.*

Proof: We write p_s^A instead of $p_s^A(\Phi_1 \mathcal{U} \Phi_2)$, and p_s^{max} instead of $p_s^{max}(\Phi_1 \mathcal{U} \Phi_2)$. Similarly, we simply write T^{max}, T_j^{max}, $T_{j,l}^{max}$ and $MaxSteps(s)$ rather than $T^{max}(\Phi_1, \Phi_2)$, $T_j^{max}(\Phi_1, \Phi_2)$, $T_{j,l}^{max}(\Phi_1, \Phi_2)$ and $MaxSteps(s, \Phi_1, \Phi_2)$.

We suppose $p_s^F = p_s^{max}$ for some $F \in \mathcal{A}dv_{sfair}$ and $s \in S \setminus T^{max}$. Let Σ be the set of paths $\sigma \in Path_{fin}^F(s)$ with $\sigma(i) \models \Phi_1 \wedge \neg\Phi_2$ for all $i \leq |\sigma|$. By Lemma 9.5.30 (page 247):

(1) $F(\sigma) \in MaxSteps(last(\sigma))$ for all $\sigma \in \Sigma$.

By definition of T^{max} we have $S \setminus T^{max} \subseteq S^+ \setminus Sat(\Phi_2)$. Let

$$U = \{u \in S \setminus T^{max} : MaxSteps(u) \neq Steps(u)\}.$$

Claim 1: For each $\sigma \in \Sigma$ with $last(\sigma) \notin T^{max}$, there exists

$$\overline{\sigma} \in \Sigma \text{ with } \sigma <_{prefix} \overline{\sigma} \text{ and } last(\overline{\sigma}) \in U.$$

Proof: Let $\sigma \in \Sigma$ with $last(\sigma) \in S \setminus T^{max}$. We suppose that there does not exists a path $\overline{\sigma} \in \Sigma$ with $\sigma <_{prefix} \overline{\sigma}$ and $last(\overline{\sigma}) \in U$.

First, we observe that $last(\sigma)$ cannot be terminal. All terminal states either belong to $Sat(\Phi_2)$ or $S \setminus S^+(\Phi_1, \Phi_2)$. Thus, all terminal states are contained in T^{max}. Moreover, there exists $\mu \in Steps(last(\sigma))$ with $Supp(\mu) \cap (Sat(\Phi_1) \setminus Sat(\Phi_2)) \neq \emptyset$.[30] Thus, the set of finite paths $\sigma' \in \Sigma$ with $\sigma <_{prefix} \sigma'$ is not empty. Let

$$T = \{last(\sigma') : \sigma' \in \Sigma,\ \sigma <_{prefix} \sigma'\} \setminus T^{max}.$$

By our assumption, $T \cap U = \emptyset$. For each $t \in T$, we choose some $\sigma_t \in \sigma \uparrow_{fin}^F \cap \Sigma$ with $t = last(\sigma_t)$ and define $\mu_t = F(\sigma_t)$. Then,

(2) $Supp(\mu_t) \subseteq T \cup T^{max}$ for all $t \in T$.[31]

Since $T \cap U = \emptyset$, we have $MaxSteps(t) = Steps(t)$ for all $t \in T$. By definition of T^{max}, we get $T \subseteq T^{max}$ and therefore $T = \emptyset$ (as T is defined as a subset of $S \setminus T^{max}$). Let $\mu = F(\sigma)$. Then, we get $Supp(\mu) \subseteq T^{max}$ which yields $last(\sigma) \in T_{j,1}^{\max}$ for some j. Thus, $last(\sigma) \in T^{max}$. Contradiction. ⌋

Claim 2: There exists $\pi \in Path_{ful}^F(s)$ with

$$\pi(i) \models \Phi_1 \wedge \neg\Phi_2,\ i = 0, 1, 2, \ldots \text{ and } U \cap inf(\pi) \neq \emptyset.$$

[30] Note that $Supp(\mu) \cap (Sat(\Phi_1) \setminus Sat(\Phi_2)) = \emptyset$ for all $\mu \in Steps(last(\sigma))$ implies $Supp(\mu) \subseteq T^{max}$ for all $\mu \in Steps(s)$ which yields $last(\sigma) \in T^{max}$.

[31] Note that, for $u \in Supp(\mu_t)$, the finite path $\sigma_t \xrightarrow{\mu_t} u$ belongs to $\sigma \uparrow_{fin}^F \cap \Sigma$.

Proof: For each $\sigma \in \Sigma$ with $last(\sigma) \in S \setminus T^{max}$, we choose some $\overline{\sigma} \in \Sigma$ with $\sigma <_{prefix} \overline{\sigma}$ and $last(\overline{\sigma}) \in U$ (which exists by Claim 1). Let $\sigma_0 = s$ and $\sigma_{j+1} = \overline{\sigma_j}$, $j = 0, 1, 2, \ldots$. Then, the unique fulpath π with $\sigma_i <_{prefix} \pi$, $i = 0, 1, 2, \ldots$, has the desired properties. $\rfloor$

We choose some $\pi \in Path^F_{ful}(s)$ with $\pi(i) \models \Phi_1 \wedge \neg\Phi_2$, $i = 0, 1, 2, \ldots$ and $u \in inf(\pi) \cap U$ (which exists by Claim 2). As F is strictly fair and $MaxSteps(u) \neq Steps(u)$ for all $u \in U$ there exists $j \geq 0$ with $F(\pi^{(j)}) \notin MaxSteps(\pi(j))$. Contradiction (to (1)) as $\pi^{(j)} \in \Sigma$. ■

Corollary 9.5.34 *If $F \in \mathcal{A}dv_{sfair}$ and $s \notin T^{max}(a^?, \neg a^+)$ then*

$$p^F_s(\Phi_1 \mathcal{U} \Phi_2) > 1 - p^{max}_s(a^? \mathcal{U} \neg a^+).$$

Proof: follows by Lemma 9.5.33 (page 249) and Corollary 9.5.26 (page 246). ■

Theorem 9.5.35 (cf. Theorem 9.3.16, page 226) *For all $s \in S$:*

$$s \models_{sfair} \mathsf{Prob}_{<p}(\Phi_1 \mathcal{U} \Phi_2) \iff \begin{cases} p^{max}_s(\Phi_1 \mathcal{U} \Phi_2) < p & : \text{ if } s \in T^{max}(\Phi_1, \Phi_2) \\ p^{max}_s(\Phi_1 \mathcal{U} \Phi_2) \leq p & : \text{ otherwise.} \end{cases}$$

Proof: follows by Lemma 9.5.31 (page 248), Lemma 9.5.33 (page 249) and Theorem 9.5.21 (page 245). ■

Theorem 9.5.36 (cf. Theorem 9.3.28, page 228) *For all $s \in S$:*

$$s \models_{sfair} \mathsf{Prob}_{>p}(\Phi_1 \mathcal{U} \Phi_2) \iff \begin{cases} 1 - p^{max}_s(a^? \mathcal{U} \neg a^+) < p & : \text{ if } s \in T^{max}(a^?, \neg a^+) \\ 1 - p^{max}_s(a^? \mathcal{U} \neg a^+) \leq p & : \text{ otherwise.} \end{cases}$$

Proof: follows by Corollary 9.5.32 (page 249), Corollary 9.5.34 (page 250) and Corollary 9.5.26 (page 246). ■

Minimal probabilities under all W-fair adversaries: We give the proof of Theorem 9.3.33 (page 230). In the sequel, W is a fixed subset of S. For simplicity, we write S^0_W rather than $S^0_W(\Phi_1, \Phi_2)$, S^+ rather than $S^+(\Phi_1, \Phi_2)$ and $S^?$ rather than $S^?(\Phi_1, \Phi_2)$.

Lemma 9.5.37 *Let π be a fulpath which is W-fair and state fair and such that $\pi \not\models \Phi_1 \mathcal{U} \Phi_2$. Then, $\pi \models a^? \mathcal{U} a^0_W$.*

Proof: We assume $\pi \not\models a^? \mathcal{U} a^0_W$. It is easy to see that π is infinite and $\pi(i) \in S^?$ for all i. Let $T = inf(\pi)$. Then, $T \subseteq S^? \subseteq S \setminus Sat(\Phi_2)$. For $t \in T \setminus W$, we choose some $\mu_t \in Steps(t)$ such that $\mu_t = step(\pi, i)$ for infinitely many indices i with $\pi(i) = t$. Then, we have:

- Let $t \in T \cap W$ and $\mu \in Steps(t)$. Since π is W-fair we have $\mu = step(\pi, i)$ for infinitely many i. By the state fairness of π, we get $Supp(\mu) \subseteq inf(\pi) = T$.
- Let $t \in T \setminus W$. Then, $Supp(\mu_t) \subseteq inf(\pi) = T$ (by the state fairness of π and the choice of μ_t).

By definition of S^0_W, we get $T \subseteq S^0_W$ which yields $\pi \models a^? \mathcal{U} a^0_W$. Contradiction. ■

Lemma 9.5.38 *If $F \in \mathcal{A}dv_{Wfair}$, $s \in S$ then $p^F_s(\Phi_1 \mathcal{U} \Phi_2) \geq 1 - p^{max}_s(a^? \mathcal{U} a^0_W)$.*

Proof: Let $\Pi = \{\pi \in Path^F_{ful} : \pi \not\models \Phi_1 \mathcal{U} \Phi_2\}$ and let Γ be the set of fulpaths $\pi \in Path^F_{ful}$ that are W-fair and state fair. Then, by Lemma 9.5.8 (page 242) and the W-fairness of F, $Prob(\Gamma(s)) = 1$ and $Prob(\Pi(s)) = Prob(\Pi(s) \cap \Gamma)$. By Lemma 9.5.37 (page 250),

$$\Pi(s) \cap \Gamma \subseteq \left\{\pi \in Path^F_{ful}(s) : \pi \models a^? \mathcal{U} a^0_W\right\}.$$

Thus, $Prob(\Pi(s) \cap \Gamma) \leq p^F_s(a^? \mathcal{U} a^0_W) \leq p^{max}_s(a^? \mathcal{U} a^0_W)$. We conclude

$$1 - p^F_s(\Phi_1 \mathcal{U} \Phi_2) = Prob(\Pi(s)) = Prob(\Pi(s) \cap \Gamma) \leq p^{max}_s(a^? \mathcal{U} a^0_W)$$

which yields $p^F_s(\Phi_1 \mathcal{U} \Phi_2) \geq 1 - p^{max}_s(a^? \mathcal{U} a^0_W)$. ■

Lemma 9.5.39 *There exists $F \in \mathcal{A}dv_{W fair}$ with $p^F_s(\Phi_1 \mathcal{U} \Phi_2) = 1 - p^{max}_s(a^? \mathcal{U} a^0_W)$ for all $s \in S$.*

Proof: Let A be a simple adversary with $p^A_s(a^? \mathcal{U} a^0_W) = p^{max}_s(a^? \mathcal{U} a^0_W)$ (which exists by Lemma 9.5.13, page 243) and

$$\Sigma = \{\sigma \in Path_{fin} : \sigma(i) \models a^? \wedge \neg a^0_W, i = 0, 1, \ldots, |\sigma| - 1, last(\sigma) \models a^0_W\}.$$

We define a W-fair adversary F such that $\Sigma^A \subseteq \Sigma^F$ and $p^F_s(\Phi_1 \mathcal{U} \Phi_2) = 0$ for all $s \in S^0_W$.

Let Λ be the set of finite paths $\lambda \in Path^A_{fin}$ such that $\lambda <_{prefix} \sigma$ for some $\sigma \in \Sigma^A$. Let T_i, $T_{i,1}$, $T_{i,2}$ be as in the definition of $S^0_W = S^0_W(\Phi_1, \Phi_2)$ (see Notation 9.3.31, page 229). Then, $S^0_W = \bigcup_{i \geq 0} T_i$, $T_0 = S \setminus S^+$ and $T_i = T_{i,1} \cup T_{i,2}$. Let

$$T = \bigcup_{i \geq 1} (T_{i,1} \cup (T_{i,2} \setminus W)).$$

By definition of T_i, for each $t \in T$, there exists some $\mu_t \in Steps(t)$ such that:

(1) If $t \in T_{i,1}$ then $Supp(\mu_t) \subseteq T_0 \cup \ldots \cup T_{i-1}$.
(2) If $t \in T_{i,2} \setminus W$ then $Supp(\mu_t) \subseteq T_0 \cup \ldots \cup T_i$.

By definition of the sets $T_{i,2}$, we have

(3) $Supp(\mu) \subseteq T_0 \cup \ldots \cup T_i$ for all $t \in T_{i,2} \cap W$ and $\mu \in Steps(t)$.

For $s \in S \setminus T$, let m_s be the cardinality of $Steps(s)$ and $\nu^s_0, \ldots, \nu^s_{m_s - 1}$ an enumeration of $Steps(s)$. We define an adversary F as follows. Let $\sigma \in Path_{fin}$.

(I) If $\sigma \in \Lambda$ then we put $F(\sigma) = A(last(\sigma))$.
(II) If $last(\sigma) \in S \setminus T$ and $\sigma \notin \Lambda$ then we define $F(\sigma) = \nu^s_i$ where $s = last(\sigma)$ and $i = \eta(\sigma) \bmod m_s$. Here, $\eta(\sigma)$ denotes the number of indices $i < |\sigma|$ such that $\sigma(i) = last(\sigma)$ and mod the "modulo-division" function.
(III) If $last(\sigma) \in T$ and $\sigma \notin \Lambda$ then $F(\sigma) = \mu_t$ where $t = last(\sigma)$.

It is easy to see that each fulpath $\pi \in StateFair^F$ is W-fair. Thus, Lemma 9.5.8 (page 242) yields the W-fairness of F. Moreover, (I) yields $\Sigma^A \subseteq \Sigma^F$. Thus,

$$p^F_s(a^? \mathcal{U} a^0_W) \geq p^A_s(a^? \mathcal{U} a^0_W) = p^{max}_s(a^? \mathcal{U} a^0_W)$$

which yields $p^F_s(a^? \mathcal{U} a^0_W) = p^{max}_s(a^? \mathcal{U} a^0_W)$ for all $s \in S$. Clearly, by (1), (2), (3), (II) and (III), we get that,

whenever $\pi \in Path^F_{ful}$ with $\pi(i) \in S^0_W$ for some i then $\pi(j) \in S^0_W$ for all $j \geq i$.

In particular, if $\pi \in Path^F_{ful}$ and $\pi \models a^?\mathcal{U}a^0_W$ then $\pi \not\models \Phi_1\mathcal{U}\Phi_2$. From this,

(4) $p^{max}_s(a^?\mathcal{U}a^0_W) \;=\; p^F_s(a^?\mathcal{U}a^0_W) \;\leq\; 1 - p^F_s(\Phi_1\mathcal{U}\Phi_2)$ for all $s \in S$.

By Lemma 9.5.38 (page 250) and (4), $p^F_s(\Phi_1\mathcal{U}\Phi_2) \;=\; 1 - p^{max}_s(a^?\mathcal{U}a^0_W)$ for all $s \in S$. ■

Theorem 9.5.40 (cf. Theorem 9.3.33, page 230) *For all $s \in S$:*

$$s \;\models_{Wfair}\; \mathsf{Prob}_{\sqsupseteq p}(\Phi_1\mathcal{U}\Phi_2) \;\; \textit{iff} \;\; 1 - p^{max}_s(a^?\mathcal{U}a^0_W) \sqsupseteq p.$$

Proof: follows by Lemma 9.5.38 (page 250) and Lemma 9.5.39 (page 251). ■

The connection between $\models_{fair}$ and $\models_{Sfair}$: We give the proof of Theorem 9.3.37 (page 231) which states that, when dealing with $W = S$, the satisfaction relations $\models_{fair}$ and $\models_{Wfair}$ coincide.

Theorem 9.5.41 (cf. Theorem 9.3.37, page 231) *If $W = S$ then for all states s and PCTL formulas Φ:*

$$s \models_{fair} \Phi \;\; \textit{iff} \;\; s \models_{Wfair} \Phi.$$

Proof: Because of Theorem 9.5.19 (page 245), Theorem 9.5.25 (page 246) and Theorem 9.5.40 (page 252) it suffices to show that $p^{max}_s(a^?\mathcal{U}\neg a^+) \;=\; p^{max}_s(a^?\mathcal{U}a^0_S)$ for all $s \in S$. Since $S \setminus S^+(\Phi_1, \Phi_2) \subseteq S^0_S(\Phi_1, \Phi_2)$ we have $p^{max}_s(a^?\mathcal{U}\neg a^+) \;\leq\; p^{max}_s(a^?\mathcal{U}a^0_S)$. Let F be defined as in the proof of Lemma 9.5.39 (page 251). Since we deal with $W = S$, the so obtained adversary F is fair. Using Lemma 9.5.10 (page 242) it is easy to see that, for each $\pi \in TotalFair^F$,

$$\pi \models a^?\mathcal{U}a^0_S \quad \text{iff} \quad \pi \models a^?\mathcal{U}\neg a^+.$$

By Lemma 9.5.12 (page 243), $p^F_s(a^?\mathcal{U}\neg a^+) = p^F_s(a^?\mathcal{U}a^0_S)$ for all $s \in S$. From this, we get

$$p^{max}_s(a^?\mathcal{U}\neg a^+) \;\geq\; p^F_s(a^?\mathcal{U}\neg a^+) \;=\; p^F_s(a^?\mathcal{U}a^0_S) \;=\; p^{max}_s(a^?\mathcal{U}a^0_S)$$

for all $s \in S$. Hence, $p^{max}_s(a^?\mathcal{U}\neg a^+) \;=\; p^{max}_s(a^?\mathcal{U}\neg a^0_S)$ for all $s \in S$. ■

9.5.3 Correctness of the *LTL* model checking algorithm

We now show the correctness of our *LTL* model checking algorithm (Section 9.4, page 234 ff). Recall that our algorithm is based on a method for verifying concurrent probabilistic systems systems against ω-automata specifications. For this, we needed the fact that, for any fair adversary of the product system $\mathcal{S} \times \mathsf{A}$ (of a concurrent probabilistic system $\mathcal{S}$ and a deterministic Rabin automata A), the probability of accepting paths agrees with the probability eventually to reach the set U' (defined as in Notation 9.4.11, page 238). This fact can be derived from the following observation whose proof uses total fairness (see Section 9.5.1, page 241 ff).

Lemma 9.5.42 *Let $(S, Steps, AP, \mathcal{L})$ be a finite proposition-labelled concurrent probabilistic system that does not contain terminal states. Let a_1, $a_2 \in AP$ and let U be the largest subset of $Sat(a_1)$ such that, for all $u \in U$,*

$$Reach(u) \subseteq U \text{ and } Reach(u) \cap Sat(a_2) \neq \emptyset.$$

Then, for all $\pi \in TotalFair$*:*

$$\pi \models \Diamond\Box(a_1 \wedge \Diamond a_2) \quad \textit{iff} \quad \pi \models \Diamond a_U$$

where $a_U \in AP$ *such that* $Sat(a_U) = U$*.*

Proof: If $\pi \models \Diamond\Box(a_1 \wedge \Diamond a_2)$ then $inf(\pi) \subseteq Sat(a_1)$ and $inf(\pi) \subseteq Sat(a_2)$. By definition of U, we get $inf(\pi) \subseteq U$; in particular, $\pi \models \Diamond a_U$. Now we assume that π is total fair and $\pi \models \Diamond a_U$. Let $i \geq 0$ be an integer with $\pi(i) \in U$. Then,

(1) $inf(\pi) \subseteq Reach(\pi(i)) \subseteq U \subseteq Sat(a_1)$.

Let $u \in inf(\pi)$. By definition of U, we have $Reach(u) \cap Sat(a_2) \neq \emptyset$. By Lemma 9.5.10 (page 242), $inf(\pi) = Reach(u)$. Hence,

(2) $inf(\pi) \cap Sat(a_2) \neq \emptyset$.

(1) and (2) yield $\pi \models \Diamond\Box(a_1 \wedge \Diamond a_2)$. ■

Lemma 9.5.43 *In the notations of Section 9.4 (page 236 ff), we have:*

$$Prob(\mathsf{AccPath}^{F'}(s_{\mathsf{A}})) = Prob\left\{\pi' \in Path_{ful}^{F'}(s_{\mathsf{A}}) : \pi' \models \Diamond a_{U'}\right\}.$$

for all states $s \in S$ *and fair adversaries* F' *of* $\mathcal{S} \times \mathsf{A}$*.*

Proof: Because of Lemma 9.5.12 (page 243), it suffices to show that, for any total fair fulpath $\pi' \in Path_{ful}^{\mathcal{S}\times\mathsf{A}}(s_{\mathsf{A}})$:

(*) $\pi' \models \Diamond a_{U'}$ iff $\pi' \in \mathsf{AccPath}(s_{\mathsf{A}})$.

We assume atomic propositions $a_{j,1}, a_{j,2}, b_j \in AP$ such that $a_{j,1} \in \mathcal{L}_{\mathsf{A}}(s')$ iff $s' \in H'_j$, $a_{j,2} \in \mathcal{L}_{\mathsf{A}}(s')$ iff $s' \in K'_j$ and $b_j \in \mathcal{L}_{\mathsf{A}}(s')$ iff $s' \in U'_j$. Clearly,

$$\mathsf{AccPath}(s_{\mathsf{A}}) = \left\{\pi' \in Path_{ful}^{\mathcal{S}\times\mathsf{A}}(s_{\mathsf{A}}) : \pi' \models \bigvee_{1\leq j\leq r} \Diamond\Box(a_{j,1} \wedge \Diamond a_{j,2})\right\}$$

and $\{\pi' : \pi' \models \Diamond a_{U'}\} = \{\pi' : \pi' \models \bigvee_{1\leq j\leq r} b_j\}$. Then, by Lemma 9.5.42 (page 252), if π' is total fair then $\pi' \models \Diamond\Box(a_{j,1} \wedge \Diamond a_{j,2})$ iff $\pi' \models \Diamond b_j$. Thus, we obtain (*) which yields the claim. ■

Chapter 10

Symbolic model checking

As in the non-probabilistic case, the verification methods for probabilistic systems that assume an *explicit* representation of the state space suffer from the state explosion problem and might fail for systems of industrial size. In the last decade, two general techniques have been developed to attack the state explosion problem for (non-probabilistic) parallel systems:

- the *symbolic methods* that are based on an *implicit* representation of the state space by ordered binary decision diagrams [BCM+90, McMil92]
- the *partial order methods* which can be classified into *reduction techniques* that investigate only certain parts of the state space [Pele93, Valm94, Gode94] and techniques that work with *net unfoldings* [McMil92a, Espa94].

Both techniques have been implemented in tools and successful applied to realistic (very large) systems. In the literature, only a few work has been done on how to avoid the state explosion problem for probabilistic systems. To the best of the author's knowledge, the adaption of the partial order approach for probabilistic systems has not yet been investigated. The research on symbolic verification methods for probabilistic systems has started so far. Clarke proposed an extension of Bryant's ordered BDDs to *multiterminal* BDDs (MTBDDs) [CFM+93] and their use for the symbolic representation of Markov chains. This idea has been further developed by Hachtel et al [HMP+94] and Hartonas-Garmhausen [HarG98]. [HMP+94] presents MTBDD-based algorithms to compute the steady-state probabilities for very large finite state machines and reports on experimental results for systems with more than 10^{27} states. In her thesis, Vicky Hartonas-Garmhausen has implemented a MTBDD-based tool for verifying probabilistic systems against *PCTL* specifications [HarG98].[1] The probabilistic systems in [HarG98] arise through the (lazy) synchronous parallel composition of several sequential components. The use of a (lazy) synchronous parallel composition allows for a representation by a fully probabilistic system whose transition probability function is described by MTBDD. As far as the author knows, symbolic methods for concurrent probabilistic systems are not yet investigated.

In this chapter we present MTBDD-based algorithms for verifying fully probabilistic and concurrent probabilistic systems against several types of specification formalisms.

[1] The theoretical foundations of the underlying symbolic *PCTL* model checker can be found in [BCH+97].

More precisely, we will describe MTBDD-based *PCTL* model checking algorithms for fully probabilistic systems and stratified systems[2] and the satisfaction relations $\models$ and $\models_{fair}$ (and briefly sketch how to deal with $\models_{sfair}$ and $\models_{Wfair}$). Moreover, we will explain how the MTBDD-based approach can be applied for deciding strong or weak bisimulation equivalence.[3] The results of this chapter are based on the joint work with Ed Clarke [BaCl98] and uses results from the joint work with Ed Clarke, Vicky Hartonas-Garmhausen, Marta Kwiatkowska and Mark Ryan [BCH+97].

In what follows, the reader is supposed to be familiar with ordered binary decision diagrams (OBDDs or BDDs for short) [Brya86] and the main ideas behind the BDD-approach for verifying parallel systems, see e.g. [BCM+90, McMil92, CGL93]. The definition of multi-terminal BDDs (MTBDDs for short) as introduced by Clarke et al [CFM+93] and related notations are summarized in the appendix (Section 12.3, page 315 ff). In Section 10.4 (page 295 ff) and the remainder of this introduction, we assume familiarity with the logic *PCTL* and *PCTL* model checking (see Section 9.3, page 216), strong bisimulation (see Section 3.4.1, page 54) and weak bisimulation (see Section 7.1.1, page 161). Throughout this chapter, we assume *finite* systems.

The basic idea behind the MTBDD-based approach for verifying probabilistic systems is the representation of the system by a (real-valued) MTBDD. Using an *encoding* of the state space in $\{0,1\}^k$ for some k, the transition probability matrix of a fully probabilistic or stratified system can be viewed as a function from bit vectors into the unit interval and represented by a MTBDD. To obtain symbolic MTBDD-based verification methods the operators used in the verification algorithms of the literature have to be replaced by operators on MTBDDs. The main operators that are used in almost all verification algorithms for probabilistic systems are the following.

(1) The computation of the probabilities of certain events requires *arithmetic operators* (like summation $+$ or multiplication $*$, minimum and maximum) and *least fixed points* of certain self-mappings of the function space $S \to [0,1]$. For instance, for *PCTL* model checking for fully probabilistic systems, the probabilities

$$p_s(\Phi_1 \mathcal{U} \Phi_2) \;=\; Prob\,\{\pi \in Path_{ful}(s) : \pi \models \Phi_1 \mathcal{U} \Phi_2\}$$

are needed to compute the set of states where the formula $\mathsf{Prob}_{\bowtie p}(\Phi_1 \mathcal{U} \Phi_2)$ holds. The function $s \mapsto p_s(\Phi_1 \mathcal{U} \Phi_2)$ can be characterized as the least fixed point of the operator $F : (S \to [0,1]) \to (S \to [0,1])$,

$$F(f)(s) \;=\; \begin{cases} 1 & : \text{ if } s \models \Phi_2 \\ \sum_{t \in S} \mathbf{P}(s,t) \cdot f(t) & : \text{ if } s \models \Phi_1 \wedge \neg \Phi_2 \\ 0 & : \text{ otherwise} \end{cases}$$

and computed either by solving a linear equation system or iteration (see Theorem 3.1.6, page 36, and Remark 3.1.8, page 36). Dealing with concurrent probabilistic systems, the corresponding operator F involves minimum or maximum operations.

[2] The reason why we deal with stratified systems rather than (general) concurrent probabilistic systems will be explained on page 297.

[3] We deal with fully probabilistic or reactive systems in the case of strong bisimulation and fully probabilistic systems in the case of weak bisimulation.

E.g. the maximal probabilities

$$p_s^{max}(\Phi_1 \mathcal{U} \Phi_2) \;=\; \sup_{A \in \mathcal{A}dv} \; Prob\left\{\pi \in Path_{ful}^{A}(s) : \pi \models \Phi_1 \mathcal{U} \Phi_2\right\}$$

in a stratified system are given by the least fixed point of the operator $F : (S \to [0,1]) \to (S \to [0,1])$ which is defined by: $F(f)(s) = 1$ if $s \models \Phi_2$, $F(f)(s) = 0$ if $s \not\models \Phi_1 \vee \Phi_2$ and, for $s \models \Phi_1 \wedge \neg\Phi_2$,

$$F(f)(s) \;=\; \begin{cases} \sum_{t \in S} \mathbf{P}(s,t) \cdot f(t) & : \text{ if } s \text{ is a probabilistic state} \\ \max_{t \in S} \mathbf{P}(s,t) \cdot f(t) & : \text{ if } s \text{ is non-probabilistic} \end{cases}$$

and can be calculated by solving a linear optimization problem or iteration (see Theorem 3.2.11, page 43, and Remark 3.2.12, page 43).

(2) For some specification formalisms, *comparison operators* (like $\leq$, $<$, $=$) are needed. For instance, for *PCTL* model checking for fully probabilistic systems, the computation of the set $Sat(\mathsf{Prob}_{\bowtie p}(\Phi_1 \mathcal{U} \Phi_2))$ requires the comparison of the constant p with the probabilities $p_s(\Phi_1 \mathcal{U} \Phi_2)$. For deciding bisimulation equivalence, we need an equality test for the probabilities $\mathbf{P}(s, \alpha, C)$ and $\mathbf{P}(s', \alpha, C)$ for the states s, s' to reach a C-state via an α-labelled transition.

(3) Several algorithms require a reachability analysis in the underlying directed graph. This can be performed with the help of set-based operators like $\cup$, $\cap$, $\setminus$ and operators for computing *least* or *greatest fixed points* of monotonic set-valued functions. For example, for *PCTL* model checking with respect to the satisfaction relation $\models_{fair}$ we have to compute the set $S^+(\Phi_1, \Phi_2)$ of all states $s \in S$ that can reach a Φ_2-state via a path through Φ_1-states (cf. Section 9.3, Notation 9.3.11, page 223). The set $S^+(\Phi_1, \Phi_2)$ can be described as the least fixed point of the operator $F : 2^S \to 2^S$,

$$F(Z) \;=\; Sat(\Phi_2) \;\cup\; \{ s \in Sat(\Phi_1) \;:\; \exists \mu \in Steps(s) \,[\; Supp(\mu) \cap Z \neq \emptyset \;]\; \}\,.$$

Hence, a general MTBDD-based framework in which a wide range of verification algorithms for probabilistic systems requires a *language for manipulating MTBDDs* via the above mentioned operators.

(1) requires binary arithmetic operators *op* (like $+$, $*$, min, max, etc.) on MTBDDs, i.e. operators that take as their input two MTBDDs Q_1 and Q_2 and return the MTBDD for the function f_{Q_1} *op* f_{Q_2}.[4] In our applications, the self-mappings of $S \to [0,1]$ for which the least fixed points have to be computed meet the conditions of Tarski's fixed point theorem for continuous operators on the complete lattice $S \to [0,1]$. Thus, the least fixed points can be obtained by iteration. Hence, we aim at an operator that computes least fixed points of certain MTBDD-valued operators (representing self-mappings of the function space $S \to [0,1]$) by iteration. Moreover, the definition of the higher-order function F requires *arithmetic quantifiers* like $\sum_t$ or $\max_t$ where the index t ranges over *all* states. (2) requires an operator that takes as its input two MTBDDs Q_1 and Q_2 and returns the BDD for the boolean function

$$f(x_1, \ldots, x_n) \;=\; \begin{cases} 1 & : \text{ if } f_{\mathrm{Q}_1}(x_1, \ldots, x_n) \;\bowtie\; f_{\mathrm{Q}_2}(x_1, \ldots, x_n) \\ 0 & : \text{ otherwise.} \end{cases}$$

[4] Here, f_{Q} denotes the function that is associated with the MTBDD Q (see Section 12.3, page 315 ff).

Here, $\bowtie$ is a comparsion operator like $\leq$, $<$ or $=$. This operator can just be seen as a special kind of combining two MTBDDs via a binary arithmetic operator, namely the operator $op_{\bowtie} : \mathbb{R} \times \mathbb{R} \to \{0,1\}$ where $q_1 \; op_{\bowtie} \; q_2$ is 1 iff $q_1 \bowtie q_2$. The operators required in (3) arise as special instances of the operators obtained from (2). For this, we identify each finite set with its characteristic function that we represent by a BDD (which is just a special case of a MTBDD). The boolean connectives on sets like union $\cup$ or intersection $\cap$ correspond to the maximum and minimum operators applied to the BDDs for the characteristic functions; similarly, the boolean quantifiers $\exists t$ or $\forall t$ are obtained from the "arithmetic quantifiers" $\max_t$ and $\min_t$. Any operator $F : 2^S \to 2^S$ can be viewed as a self-mapping of the function space $S \to \{0,1\}$. In our applications, these mappings have a natural extension to self-mappings of the function space $S \to [0,1]$ and can be viewed as MTBDD-valued operators. Thus, the above mentioned iteration operator on MTBDDs should also be applicable to compute the BDD for the least or greatest fixed point of a monotonic set-valued operator. In summary, the necessary ingredients for a uniform language for MTBDDs that is expressive enough to subsume a wide range of verification methods for probabilistic systems are

- operators for combining two MTBDDs via arithmetic operators,
- "arithmetic quantifiers" like $\sum_t$, $\min_t$ or $\max_t$,
- an iteration operator that returns (an approximation of) the limit of (the MTBDD representations of) certain function sequences.

The algebraic mu-calculus: These requirements have lead to the *algebraic mu-calculus* which can be viewed as a generalization Park's relational mu-calculus [Park74]. While the relational mu-calculus deals with *formulas* (interpreted by the usual truth values 0 and 1) and *relational terms* (interpreted by relations, or equivalently, boolean-valued functions) the algebraic mu-calculus deals with *algebraic expressions* (interpreted by real numbers) and *algebraic terms* (interpreted by real-valued functions). The main concepts of the relational mu-calculus are the boolean connectives $\wedge$, $\vee$, $\neg$, quantification $\exists t$ and $\forall t$, λ-abstraction of the formulas and least/greatest fixed point operators. These are replaced by arithmetic connectives $+$, $*$, etc., the algebraic quantifiers $\sum_t$, $\min_t$ and $\max_t$, λ-abstraction of the algebraic expressions and an iteration operator (which might specialize to a fixed point operator).

MTBDD-based compilation: We present an algorithm that computes the semantics for the expressions and terms where the underlying data structure are MTBDDs. In the same way as the relational mu-calculus (together with the BDD-based method of [BCM$^+$90] for evaluating the formulas and terms of the relational mu-calculus) can be viewed as a language for BDDs the algebraic mu-calculus yields a language for MTBDDs where the algorithm to compute the MTBDD representations for the expressions and terms can be viewed as "compiler".

Applications: The algebraic mu-calculus together with this MTBDD-based algorithm has applications in various areas. Several temporal and modal logics can be embedded into the algebraic mu-calculus. Hence, the algebraic mu-calculus itself can serve as a *specification language* for several types of programs. In particular, the algebraic mu-calculus subsumes the logic *PCTL* (with the interpretations over fully probabilistic [HaJo94] and stratified systems with e.g. the standard interpretation à la [BidAl95] or

$\models_{fair}$ as introduced in Chapter 9) and can express bisimulation equivalence à la Larsen & Skou [LaSk89] or weak bisimulation in the sense of Chapter 7. In these applications, the algebraic mu-calculus together with its MTBDD-based "compiler" specializes to a *symbolic model checker* for probabilistic systems. Beside the verification of probabilistic (or other types of) programs, the algebraic mu-calculus is applicable in other contexts, e.g. for solving graph-theoretical problems (such as shortest path problems) or for MTBDD-based numerical methods in linear algebra (such as solving linear equation systems or computing eigenvalues).

Organization of that chapter: The syntax and semantics of the algebraic mu-calculus is presented in Section 10.1. Section 10.2 shows that the algebraic mu-calculus subsumes several temporal or modal logics (and hence, can serve itself as specification language for several types of parallel systems). In Section 10.3 we describe the MTBDD-based algorithm for computing the semantics of the algebraic mu-calculus. Section 10.4 explains how the algebraic mu-calculus can be applied to obtain symbolic model checking algorithms for verifying probabilistic systems.

10.1 The algebraic mu-calculus

This section presents the syntax and semantics of the *algebraic mu-calculus*. The algebraic mu-calculus can be viewed as an extension of Park's relational mu-calculus [Park74]. While the relational mu-calculus contains *formulas* (interpreted by the usual truth values 0 and 1) and *relational terms* (interpreted by relations that – when identified with their characteristic function – can be viewed as boolean-valued functions), the algebraic mu-calculus deals with *algebraic expressions* (interpreted by real numbers) and *algebraic terms* (interpreted by real-valued functions). The relational terms are mainly built by predicate symbols, λ-abstraction from the formulas and a least or greatest fixed point operator. For an interpretation by real-valued functions (rather than boolean-valued functions), the predicate symbols are replaced by function symbols; the concept of λ-abstraction is maintained. The fixed point operators of the relational mu-calculus are partial operators that can only be applied to those relational terms where the induced semantic operator yields a monotonic set-valued function. The existence of the least or greatest fixed point is then ensured by Tarski's fixed point theorem. Dealing with real-valued functions rather than boolean-valued functions (sets) leads to the problem that least or greatest fixed point (or even fixed points at all) of monotonic operators might not exist; or, if they exist, one might be interested in other fixed points than the least or greatest ones. For this reason, we replace the least/greatest fixed point operators by a *limit operator*. The intended meaning of this limit operator is the limit of function sequences of the form $f, F(f), F(F(f)), F(F(F(f))), \ldots$ for some function f and some higher-order operator F. In the case where F can be restriced to a monotonic operator on boolean-valued functions (sets), i.e. an operator $(D \to \{0,1\}) \to (D \to \{0,1\})$, or equivalently, $2^D \to 2^D$ for some finite set D, and where f is the boolean function that always returns the truth value 0 (resp. 1), i.e. f represents the empty set (resp. the set D), the above sequence converges to the least (resp. greatest) fixed point of F (as an operator $2^D \to 2^D$ on sets). Thus, our limit operator generalizes the least and greatest fixed point operators of the relational mu-calculus. Clearly, for arbitrary f and F, the above function sequence does not converge.

$$
\begin{array}{rcl}
expr & ::= & q \;\Big|\; expr_1\ op\ expr_2 \;\Big|\; term(z_1,\ldots,z_n) \;\Big|\; \sum_z [expr] \;\Big|\; \\
 & & \min_z [expr] \;\Big|\; \max_z [expr] \\
term & ::= & fct \;\Big|\; Z \;\Big|\; \lambda z_1,\ldots,z_n[expr] \;\Big|\; lim\ Z\ [\ term \uparrow term_0\ ] \;\Big|\; \\
 & & iterate\ Z\ [\ term \uparrow^k term_0\ ]
\end{array}
$$

Figure 10.1: Syntax of the algebraic mu-calculus

For this reason, the meanings of the algebraic terms are *partial* real-valued functions. The semantics of the limit operator returns a partial function that is undefined for those arguments d where the sequence $f(d), F(f)(d), F(F(f))(d), \ldots$ does not converge.

10.1.1 Syntax of the algebraic mu-calculus

The syntax of the algebraic mu-calculus arises from Park's relational mu-calculus [Park74] by using arbitary arithmetic operators (e.g. summation $+$ or multiplication $*$) instead of the boolean connectives $\vee$ and $\wedge$, replacing the quantifiers $\exists z$ and $\forall z$ by arithmetic ones $\sum_z$, $\min_z$ and $\max_z$ and the least/greatest fixed point operators by a limit operator. Moreover, we add a bounded iteration operator (that could be added to the relational mu-calculus as well) whose intended meaning is the function f_k obtained by an iteration of the form $f_0 = f\ f_{i+1} = F(f_i)$ for a certain function f and a higher-order operator F.

The algebraic mu-calculus: Let *IndVar* be a set of of *individual variables*, *TermVar* a set of *term variables* and *Fct* a set of *function symbols*. The term variables and function symbols are associated with an arity (a natural number ≥ 1). $TermVar^n$ and Fct^n denote the set of n-ary term variables resp. n-ary function symbols. Let *Op* be a set of binary arithmetic operators on the reals including summation $+$, minus $-$, multiplication $*$, the binary minimum and maximum operators op_{min} and op_{max} (where e.g. $q_1\ op_{min}\ q_2 = \min\{q_1, q_2\}$) and the comparison operators $op_{\bowtie}$ where $\bowtie \in \{\leq, <, \geq, >, =, \neq\}$ and

$$q_1\ op_{\bowtie}\ q_2 \quad = \quad \begin{cases} 1 & : \text{ if } q_1 \bowtie q_2 \\ 0 & : \text{ otherwise.} \end{cases}$$

Op might also contain partial operators such as division $\%$ which is undefined if the second argument is 0. Expressions and n-ary terms of the algebraic mu-calculus (called *algebraic expressions* and *algebraic terms*) are built from the production system shown in Figure 10.1 on page 260. Here, q is a real number, $op \in Op$, z, $z_1, \ldots, z_n$ are individual variables such that $z_1, \ldots, z_n$ are pairwise distinct, *fct* is an n-ary function symbol, Z is an n-ary term variable and k a natural number. For the terms $lim\ Z\ [\ term \uparrow term_0\ ]$ and $iterate\ Z\ [\ term \uparrow^k term_0\ ]$, we require that Z is a term variable and *term* and $term_0$ are algebraic terms such that Z, *term* and $term_0$ have the same arity. As usual, we define

the *boundedness* of occurrences of variables in algebraic expressions or algebraic terms. The individual variables can be bounded by the operators $\sum$, min, max and λ-abstraction while the term variables can be bounded by the limit or bounded iteration operator. An occurrence of an individual variable or a term variable in an algebraic expression or term is said to be *free* if it is not bounded. An algebraic expression or term is called *closed* if it does not contain free occurrences of term and individual variables. For the expressions $term(z_1, \ldots, z_n)$, we require that none of the individual variables $z_1, \ldots, z_n$ occurs free in *term*. In what follows, we write

- $-expr$ rather than $0 - expr$,
- $\min\{expr_1, expr_2\}$ rather than $expr_1\ op_{min}\ expr_2$,
- $\max\{expr_1, expr_2\}$ rather than $expr_1\ op_{max}\ expr_2$,
- $|expr|$ rather than $\max\{expr, -expr\}$.

Moreover, we often write

$$\sum_{z_1,\ldots,z_n} [expr] \quad \text{rather than} \quad \sum_{z_1} \left[\sum_{z_2} \left[\ldots \sum_{z_n} [expr] \ldots \right] \right].$$

The notations $\min_{z_1,\ldots,z_n}[expr]$ and $\max_{z_1,\ldots,z_n}[expr]$ are used with corresponding meanings. Intuitively, *lim* $Z[term \uparrow term_0]$ stands for the "limit" of the sequence $(term_i)_{i\geq 0}$ where $term_{i+1} = term\{Z \leftarrow term_i\}$, $i = 0, 1, 2, \ldots$ and where the brackets $\{\ldots\}$ denote syntactic replacement. The intended meaning of *iterate* Z [$term \uparrow^k term_0$] is $term_k$ where $term_0, term_1, \ldots$ are defined as before.

The boolean mu-calculus: The boolean mu-calculus is a subcalculus of the algebraic mu-calculus where only those operators *op* are allowed that are closed under the boolean values 0 and 1 (i.e. that can be restricted to operators $\{0,1\}^2 \to \{0,1\}$). Formally, expressions and terms of the *boolean mu-calculus* are built from the production system shown in Figure 10.2 (page 261). Here, $expr_1$, $expr_2$ are arbitrary algebraic expressions

$$\begin{array}{lll}
bexpr & ::= & 0 \mid 1 \mid bexpr_1 \wedge bexpr_2 \mid bexpr_1 \vee bexpr_2 \mid \neg bexpr \mid \\
 & & expr_1\ op_{\bowtie}\ expr_2 \mid bterm(z_1, \ldots, z_n) \mid \forall z\ [bexpr] \mid \exists z\ [bexpr] \\
bterm & ::= & fct \mid \lambda z_1, \ldots, z_n\ [bexpr] \mid Z \mid lfp\ Z\ [bterm] \mid gfp\ Z\ [bterm] \mid \\
 & & iterate\ Z\ [\ bterm \uparrow^k bterm_0\ ]
\end{array}$$

Figure 10.2: Syntax of the boolean mu-calculus

and $\wedge = op_{min}$, $\vee = op_{max}$, $\neg bexpr = 1 - bexpr$ and

$$\forall z\ [bexpr] = \min_z [bexpr], \quad \exists z\ [bexpr] = \max_z [bexpr].$$

The *least* and *greatest fixed point operators* $lfp\ Z\ [\ldots]$ and $gfp\ Z\ [\ldots]$ are given by:[5]

$$lfp\ Z\ [bterm] \ = \ lim\ Z[\ bterm \uparrow \lambda z_1, \ldots, z_n[0]\ ],$$
$$gfp\ Z\ [bterm] \ = \ lim\ Z[\ bterm \uparrow \lambda z_1, \ldots, z_n[1]\ ].$$

Here, we assume that n is the arity of Z and *bterm*. As usual, other boolean connectives, such as "implication" $\rightarrow$ or "equivalence" $\leftrightarrow$, can be derived from $\wedge$, $\vee$ and $\neg$. In the boolean subcalculus, we write $expr_1 \bowtie expr_2$ rather than $expr_1\, op_{\bowtie}\, expr_2$. We refer to the terms and expressions of the boolean mu-calculus as *boolean terms* or *boolean expressions*.

The relational mu-calculus à la Park can be viewed as a subcalculus of the boolean mu-calculus. More precisely, formulas (resp. terms) of the relational mu-calculus are those expressions (resp. terms) of the boolean mu-calculus that do not contain subexpressions of the form $expr_1 \bowtie expr_2$ and the bounded iteration operator. In Section 10.2.1 (see page 275) we show that the standard semantics for the relational mu-calculus à la Park coincides with the induced semantics of the relational mu-calculus as a sublanguage of the algebraic mu-calculus.

10.1.2 Semantics of the algebraic mu-calculus

Intuitively, algebraic expressions are interpreted by real numbers, algebraic terms by real-valued functions.[6] To handle non-converging behaviour in the case of the limit operator $lim\ Z[\ldots]$, we extend the real line by a special symbol $\perp$ which can be interpreted as "undefined" or "divergence". Functions with range $\mathbb{R} \cup \{\perp\}$ can be viewed as *partial* functions that are undefined for those arguments where the value $\perp$ is returned.

Definition 10.1.1 [Extended reals] *Let $\mathbb{R}$ be the set of real numbers and $\perp \notin \mathbb{R}$. Then, $\overline{\mathbb{R}} = \mathbb{R} \cup \{\perp\}$ is called the domain of* extended reals.

In the sequel, we use subscripts to denote certain subsets of reals or extended reals. For instance, $\mathbb{R}_{>0}$ denotes the set of positive reals and $\overline{\mathbb{R}}_{\leq 1} = \{q \in \mathbb{R} : q \leq 1\} \cup \{\perp\}$.

The limit operator for converging sequences of reals is extended to an operator *lim* on arbitrary sequences of extended reals.

Notation 10.1.2 [The operator *lim*] *Let $q_0, q_1, q_2, \ldots$ be an infinite sequence in $\overline{\mathbb{R}}$.*

- *If $q_n \in \mathbb{R}$ for almost all n, e.g. $q_n \in \mathbb{R}$ for all $n \geq n_0$, then*

$$lim(q_0, q_1, q_2, \ldots) \ = \ \begin{cases} \perp & : \ \text{if } (q_n)_{n \geq n_0} \text{ does not converge in } \mathbb{R} \\ \lim q_n & : \ \text{if } (q_n)_{n \geq n_0} \text{ converges in } \mathbb{R}. \end{cases}$$

  Here, $\lim q_n$ denotes the usual limit of $(q_n)_{n \geq n_0}$ in $\mathbb{R}$.

[5] The reason not to use the standard notations $\mu Z[\ldots]$ and $\nu Z[\ldots]$ to denote the least and greatest fixed point operators is that, in the other chapters of that thesis, the greek letters μ and ν range over distributions. Of course, instead of the least and greatest fixed point operators, the more general limit operator $lim\ Z\ [bterm \uparrow bterm_0]$ could be added to the boolean mu-calculus. However, for the applications of the boolean mu-calculus that are considered in that thesis, only the least and greatest fixed point operators are needed.

[6] For the boolean expressions, we have an interpretation by the usual truth values 0 or 1 and for the boolean terms an interpretation by boolean-valued functions in mind.

- *If $q_n = \bot$ for infinitely many n then $\lim(q_0, q_1, q_2, \ldots) = \bot$.*

If X is a set and $(f_j)_{j\geq 0}$ is a sequence of functions $f_j : X \to \overline{\mathbb{R}}$ then $\lim(f_0, f_1, f_2, \ldots)$ denotes the function $X \to \overline{\mathbb{R}}$, $x \mapsto \lim(f_0(x), f_1(x), f_2(x), \ldots)$.

Algebraic expressions are interpreted by extended real numbers, n-ary algebraic terms by n-ary functions into the extended reals where the interpretation for the individual and term variables and the function symbols is given by a *model* for the algebraic mu-calculus.

Definition 10.1.3 [Models for the algebraic mu-calculus] *A* model *for the algebraic mu-calculus is a pair $\mathcal{M} = (D, I)$ consisting of a nonempty finite set D (called the* domain*) and an interpretation I for the individual and term variables and the function symbols, i.e. a function I which assigns*

- *to each individual variable z an element $I(z) \in D$,*
- *to each n-ary term variable Z a function $I(Z) : D^n \to \overline{\mathbb{R}}$,*
- *to each n-ary function symbol fct a function $I(fct) : D^n \to \overline{\mathbb{R}}$.*

Remark 10.1.4 From a purely mathemetical point of view, the concept of function symbols can be removed from the algebraic mu-calculus since function symbols can be considered as special term variables (namely, term variables that cannot be bounded by the limit or bounded iteration operator). However, the use of both function symbols and term variables is motivated by the convention that function symbols represent functions for which we have a fixed meaning in mind (e.g. the transition probability function of a fully probabilistic system) while the term variables are used in the scope of the limit or bounded iteration operator *lim* Z [...] or *iterate* Z [...]. Thus, the term variables are auxiliary symbols that are needed for technical reasons only while the function symbols stand for objects of the "real world". ■

Notation 10.1.5 [The models $\mathcal{M}[\ldots]$] *Let $\mathcal{M} = (D, I)$ be a model. If n, $m \geq 0$ and $z_1, \ldots, z_n$ are pairwise distinct individual variables and $Z_1, \ldots, Z_m$ are pairwise distinct term variables where the arity of Z_j is k_j and $d_i \in D$, $f_j : D^{k_j} \to \overline{\mathbb{R}}$ then*

$$\mathcal{M}[z_1 := d_1, \ldots, z_n := d_n, Z_1 := f_1, \ldots, Z_m := f_m]$$

denotes the model $(D, I[z_1 := d_1, \ldots, z_n := d_n, Z_1 := f_1, \ldots, Z_m := f_m])$ where

$$I[z_1 := d_1, \ldots, z_n := d_n, Z_1 := f_1, \ldots, Z_m := f_m]$$

is those interpretation J for the individual and term variables and the function symbols such that $J(fct) = I(fct)$ for all function symbols fct and $J(z_i) = d_i$, $i = 1, \ldots, n$, $J(Z_j) = f_j$, $j = 1, \ldots, m$, and $J(z) = I(z)$, $J(Z) = I(Z)$ in all other cases.

Let $\mathcal{M} = (D, I)$ be a model. For each algebraic expression *expr* and algebraic term *term*, $[\![expr]\!]^{\mathcal{M}} \in \overline{\mathbb{R}}$ and $[\![term]\!]^{\mathcal{M}} : D^n \to \overline{\mathbb{R}}$ are defined as shown in Figure 10.3 on page 264. Here, we assume that all operators $op \in Op$ are extended to *total* operators on $\overline{\mathbb{R}}$. For instance, the partial division operator % on the reals is extended to a total operator on $\overline{\mathbb{R}}$ by $q_1 \% q_2 = \bot$ if $q_2 = 0$ or $\bot \in \{q_1, q_2\}$.

$$
\begin{array}{l}
[\![q]\!]^{\mathcal{M}} = q \\
[\![\ expr_1\ op\ expr_2\ ]\!]^{\mathcal{M}} \ = \ [\![expr_1]\!]^{\mathcal{M}}\ op\ [\![expr_2]\!]^{\mathcal{M}} \\
[\![\ term(z_1,\ldots,z_n)\ ]\!]^{\mathcal{M}} \ = \ [\![term]\!]^{\mathcal{M}}(I(z_1),\ldots,I(z_n)) \\
[\![\ \textstyle\sum_z[expr]\ ]\!]^{\mathcal{M}} \ = \ \sum_{d\in D}\ [\![expr]\!]^{\mathcal{M}[z:=d]} \\
[\![\ \min_z[expr]\ ]\!]^{\mathcal{M}} \ = \ \min\left\{[\![expr]\!]^{\mathcal{M}[z:=d]} : d \in D\right\} \\
[\![\ \max_z[expr]\ ]\!]^{\mathcal{M}} \ = \ \max\left\{[\![expr]\!]^{\mathcal{M}[z:=d]} : d \in D\right\} \\
[\![fct]\!]^{\mathcal{M}} \ = \ I(fct), \quad [\![Z]\!]^{\mathcal{M}} \ = \ I(Z) \\
[\![\ \lambda z_1,\ldots,z_n[expr]\ ]\!]^{\mathcal{M}}(d_1,\ldots,d_n) \ = \ [\![expr]\!]^{\mathcal{M}[z_1:=d_1,\ldots,z_n:=d_n]} \\
[\![\ lim\ Z\ [\ term \uparrow term_0\ ]\ ]\!]^{\mathcal{M}} \ = \ lim(f_0,f_1,f_2,\ldots) \\
[\![\ iterate\ Z\ [\ term \uparrow^k term_0\ ]\ ]\!]^{\mathcal{M}} \ = \ f_k \\
\qquad\qquad \text{where } f_0 = [\![term_0]\!]^{\mathcal{M}},\ f_{i+1} = [\![term]\!]^{\mathcal{M}[Z:=f_i]}.
\end{array}
$$

Figure 10.3: Semantics of the algebraic mu-calculus

Example 10.1.6 [Computing shortest paths] Let $G = (V, E, cost)$ be a finite directed graph with a positive cost function, i.e. V is a finite set of vertices, $E \subseteq V \times V$ a set of directed edges and $cost : E \to \mathbb{R}_{>0}$ a function that assigns to each edge (v, w) the cost $cost(v, w)$ for passing the edge from v to w. Let $mincost : V \times V \to \overline{\mathbb{R}}$ be the function that returns for any pair (v, w) of nodes in G the length of a shortest path from v to w. (We put $mincost(v, w) = \bot$ if there is no path from v to w.) It is easy to see that the function $mincost$ is the limit of the sequence $(f_i)_{i\geq 0}$ where the functions $f_i : V \times V \to \overline{\mathbb{R}}$ are given by

$$f_0(v,w) \ = \ \begin{cases} 0 & : \text{ if } v = w \\ \bot & : \text{ otherwise} \end{cases}$$

and

$$f_{i+1}(v,w) \ = \ \min\{\ f_i(v,w),\ \min\{f_i(v,u) + cost(u,w) : (u,w) \in E\}\ \}$$

where the limit is taken in $\overline{\mathbb{R}}$ (cf. Notation 10.1.2, page 262).[7] We use binary function symbols $cost$ and id and the model $\mathcal{M} = (V, I)$ where the underlying domain is the vertex set V and the interpretation I is given by $I(id) = f_0$ and

$$I(cost)(v,w) \ = \ \begin{cases} cost(v,w) & : \text{ if } (v,w) \in E \\ \bot & : \text{ otherwise.} \end{cases}$$

[7] Here, we assume that the natural order on the real line is extended by $q < \bot$ for all real numbers q. (This yields $\min Q = \min(Q \setminus \{\bot\})$ if $\emptyset \neq Q \subseteq \overline{\mathbb{R}}$, $Q \neq \{\bot\}$.) For the extension of $+$ to an operator on the extended reals, we require that $\bot + q = q + \bot = \bot$.

Then, the semantics of the algebraic term $lim\ Z\ [\ term \uparrow id\ ]$ is the function *mincost* where

$$term \ = \ \lambda v, w \ \left[\ \min \ \left\{ \ Z(v,w), \ \min_u [\ Z(v,u) + cost(u,w)\ ] \ \right\} \ \right].$$

This can be seen as follows. We have $f_0 = [\![id]\!]^{\mathcal{M}}$. By induction on i, we get that

$$f_{i+1} = [\![term]\!]^{\mathcal{M}[Z:=f_i]},\ i = 0, 1, 2, \ldots.$$

Thus, $[\![\ lim\ Z\ [term \uparrow id]\ ]\!]^{\mathcal{M}} \ = \ lim(f_0, f_1, f_2, \ldots) \ = \ mincost$. The semantics of the term

$$iterate\ Z \ \left[\ \lambda v, w \ \left[\ \min_u [\ Z(v,u) + cost(u,w)\ ] \ \right] \ \uparrow^k id \ \right]$$

with respect to $\mathcal{M}$ is the function f_k. The value $f_k(v,w)$ is the cost of a shortest path $v = v_0, v_1, \ldots, v_l = w$ where $l \leq k$. ■

Example 10.1.7 [Matrix multiplication] Let $\mathbf{A}$ be a $n \times m$-matrix, $\mathbf{B}$ a $m \times l$-matrix. $\mathbf{A}(i,j)$ denotes the element of $\mathbf{A}$ in the i-th row and j-th column; similarly, the element of $\mathbf{B}$ in the j-th row and k-th column is denoted by $\mathbf{B}(j,k)$. The following algebraic terms describes the matrix product $\mathbf{A} \cdot \mathbf{B}$.

$$term \ = \ \lambda i, k \ \left[\ \sum_j [\ A(i,j) * B(j,k)\ ] \ \right]$$

where A and B are binary function symbols that represent the matrices $\mathbf{A}$ and $\mathbf{B}$ respectively. More precisely, if $N = \max\{n, m, l\}$ and $\mathcal{M} = (\{1, \ldots, N\}, I)$ where

$$I(A)(i,j) \ = \ \begin{cases} \mathbf{A}(i,j) & : \text{ if } 1 \leq i \leq n \text{ and } 1 \leq j \leq m \\ \bot & : \text{ otherwise} \end{cases}$$

$$I(B)(j,k) \ = \ \begin{cases} \mathbf{B}(j,k) & : \text{ if } 1 \leq j \leq m \text{ and } 1 \leq k \leq l \\ \bot & : \text{ otherwise} \end{cases}$$

then $[\![term]\!]^{\mathcal{M}}$ represents $\mathbf{A} \cdot \mathbf{B}$ in the sense that $[\![term]\!]^{\mathcal{M}}(i,k)$ is the element of $\mathbf{A} \cdot \mathbf{B}$ in the i-th row and k-th column (provided that $1 \leq i \leq n$ and $1 \leq k \leq l$).[8] ■

Example 10.1.8 [Iterative squaring] In certain applications, one has to compute $\mathbf{A}^K$ for a quadratic matrix $\mathbf{A}$ and some large K.[9] For simplicity, we assume that $K = 2^k$. Instead of computing $\mathbf{A}^K$ by the iteration $\mathbf{A}^{i+1} = \mathbf{A}^i \cdot \mathbf{A}$, $i = 2, \ldots, K-1$, it is better to use iterative squaring which is based on the iteration $\mathbf{A}^{2^{i+1}} = \mathbf{A}^{2^i} \cdot \mathbf{A}^{2^i}$, $i = 0, 1, \ldots, k-1$. This can be described by the algebraic term

$$iterate\ Z \ \left[\ \lambda i, k \ \left[\ \sum_j [\ Z(i,j) * Z(j,k)\ ] \ \right] \ \uparrow^{k-1} \ A \ \right]$$

where A is a binary function symbol that represents $\mathbf{A}$. ■

[8] Here, we assume that $\bot * q = q * \bot = \bot + q = q + \bot = 0$ if $q \in \mathbb{R}$ and $\bot * \bot = \bot + \bot = \bot$.

[9] For example, one of the two methods for the handling of the bounded until operator $\mathcal{U}^{\leq K}$ in the *PCTL* model checking algorithm of [HaJo94] is based on the computation of $\mathbf{A}^K$ for some matrix $\mathbf{A}$.

In the literature about numerical analysis, a variety of iterative methods for matrix operations are proposed; see e.g. [Varg62, YoGr73]. Using the limit operator, most of them can be described as terms of the algebraic mu-calculus. In Example 10.1.9, we consider the "naive" method for solving linear equation systems of the type $\mathbf{z} = \mathbf{q} + \mathbf{A} \cdot \mathbf{z}$ that is based on the iteration $\mathbf{z}_{k+1} = \mathbf{q} + \mathbf{A} \cdot \mathbf{z}_k$. This method can be viewed as the basis of several iterative methods, e.g. the methods by Jacobi or Gauss-Seidel or the relaxation methods. Another possible application of the algebraic mu-calculus in the field of matrix operations is the computation of eigenvalues by well-known iterative methods, e.g. the methods by Mises or Wielandt (see Example 10.1.10, page 266).

Example 10.1.9 [Solving linear equation systems] Let $\mathbf{A}$ be a real $n \times n$-matrix, $\mathbf{I}$ the $n \times n$-identity matrix. We assume that $\|\mathbf{I} - \mathbf{A}\| < 1$ for some matrix norm $\| \cdot \|$. Then, $\mathbf{I} - \mathbf{A}$ is regular and, for each vector $\mathbf{q} \in \mathbb{R}^n$, the sequence $(\mathbf{z}_k)_{k \geq 0}$ converges to the unique solution of the equation system $(\mathbf{I} - \mathbf{A}) \cdot \mathbf{z} = \mathbf{q}$ where $\mathbf{z}_0$ is an arbitrary real vector with n components and $\mathbf{z}_{k+1} = \mathbf{q} + \mathbf{A} \cdot \mathbf{z}_k$. We use a binary function symbol A (that represents $\mathbf{A}$) and 1-ary function symbols q and z_0 (that represent the vector $\mathbf{q}$ and the starting vector $\mathbf{z}_0$) and consider the algebraic term

$$term \;=\; lim\ Z \left[\lambda i \left[q(i) \;+\; \sum_j \left[\ A(i,j) * Z(j)\ \right] \right] \;\uparrow\; z_0 \right].$$

Let $\mathcal{M} = (\{1, \ldots, n\}, I)$ where $I(A)(i,j) = \mathbf{A}(i,j)$, $I(q)(i)$ is the i-th component of $\mathbf{q}$ and $I(z_0)(i)$ the i-th component of $\mathbf{z}_0$. Then, $[\![term]\!]^{\mathcal{M}}$ represents the solution of $(\mathbf{I}-\mathbf{A})\cdot\mathbf{z} = \mathbf{q}$, i.e. the vector $([\![term]\!]^{\mathcal{M}}(i))_{1 \leq i \leq n}$ is the unique solution of $\mathbf{z} = \mathbf{q} + \mathbf{A} \cdot \mathbf{z}$.

The Jacobi-approach for solving linear equation systems of the type $\mathbf{B} \cdot \mathbf{z} = \mathbf{q}$ is a modification of this naive method. It is based on a decomposition of the matrix $\mathbf{B}$ into $\mathbf{D}+\mathbf{L}+\mathbf{U}$ where $\mathbf{D}$ is a diagonal matrix, $\mathbf{L}$ a lower triangular matrix and $\mathbf{U}$ an upper triangular matrix (where all entries in the diagonals of $\mathbf{L}$ and $\mathbf{U}$ are 0). We assume that $\mathbf{D}$ is regular (i.e. all elements in the diagonal of $\mathbf{B}$ are non-zero) and that $\mathbf{A} = \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$ satisfies the conditions of the naive method (i.e. $\|\mathbf{I} - \mathbf{A}\| < 1$ for some matrix norm $\| \cdot \|$). Then, the Jacobi-method is based on the observation that

$$\mathbf{B} \cdot \mathbf{z} = \mathbf{q} \quad \text{iff} \quad \mathbf{z} \;=\; \mathbf{D}^{-1} \cdot (\mathbf{q} \;-\; (\mathbf{L} + \mathbf{U}) \cdot \mathbf{z})$$

and uses the iteration $\mathbf{z}_{k+1} = \mathbf{D}^{-1} \cdot (\mathbf{q} - \ (\mathbf{L} + \mathbf{U}) \cdot \mathbf{z}_k)$. This corresponds to the algebraic term

$$lim\ Z \left[\lambda i \left[\left(q(i) \;-\; \sum_j \left[\ h(i,j) * B(i,j) * z(j)\ \right] \right) \;\%\; B(i,i) \right] \;\uparrow\; z_0 \right]$$

where B, z_0, q are function symbols with the obvious interpretations. h is a binary function symbol for which we assume the interpretation $I(h) : \{1, \ldots, n\}^2 \to \{0, 1\}$, $I(h)(i,j) = 1$ iff $i \neq j$. ■

Example 10.1.10 [Computing eigenvectors] We sketch how the iterative method by Mises for computing eigenvectors can be described by an algebraic term. Let $\mathbf{A}$ be a $n \times n$-matrix that is similar to a diagonal matrix and let $\mathbf{z}_0$ be a real vector with n components. Let $\| \cdot \|$ be a vector norm and let

$$\mathbf{z}_{k+1} \;=\; \frac{1}{\|\mathbf{y}_k\|} \cdot \mathbf{y}_k \quad \text{where} \quad \mathbf{y}_k \;=\; \mathbf{A} \cdot \mathbf{z}_k, \quad k = 0, 1, 2, \ldots.$$

Under certain conditions about $\mathbf{z}_0$, the sequence $(\mathbf{z}_k)_{k\geq 0}$ converges to an eigenvector of $\mathbf{A}$. Using a binary function symbol A (that represents $\mathbf{A}$) and a 1-ary function symbol z_0 (that represents the starting vector $\mathbf{z}_0$) this iterative method can be described in the algebraic mu-calculus by the term $\mathit{lim}\ Z\ [\mathit{term} \uparrow z_0]$ where

$$\mathit{term} \;=\; \lambda i \left[\sum_j [\ A(i,j) * Z(j)\ ] \;\%\; \max_j [\ |A(i,j) * Z(j)|\ ] \right].$$

The underlying vector norm that we use here is the maximum norm $\|\mathbf{y}\|_\infty = \max\{|y_i| : i = 1,\ldots,n\}$ if $\mathbf{y} = (y_i)_{1\leq i\leq n}$. ■

Having a fixed model $\mathcal{M} = (D, I)$ in mind, it is often useful to extend the syntax of the algebraic mu-calculus by expressions of the form $\mathit{term}(\kappa_1,\ldots,\kappa_n)$ where κ_i are either individual variables that do do occur free in term or values of the underlying domain D. Then, $\mathit{term}(\kappa_1,\ldots,\kappa_n)$ stands short for the algebraic expression $\mathit{term}'(z_1,\ldots,z_k)$ where $\{z_1,\ldots,z_k\} = \mathit{IndVar} \cap \{\kappa_1,\ldots,\kappa_n\}$,

$$\mathit{term}' \;=\; \lambda z_1,\ldots,z_k \left[\sum_{\zeta_1,\ldots,\zeta_n} [\ \mathit{term}(\zeta_1,\ldots,\zeta_n) * \mathit{bexpr}\ ] \right]$$

and

$$\mathit{bexpr} \;=\; \bigwedge_{d\in D} \bigwedge_{\substack{1\leq i\leq n\\ \kappa_i = d}} E_d(\zeta_i) \;\wedge\; \bigwedge_{1\leq j\leq k} \bigwedge_{\substack{1\leq i\leq n\\ \kappa_i = z_j}} E(\zeta_i, z_j).$$

Here, $\zeta_1,\ldots,\zeta_n$ are "fresh" pairwise distinct individual variables that do not occur free in term. E is a binary function symbol (that represents the equality predicate on D), E_d are 1-ary function symbols (that stand for the singleton set $\{d\}$). The intended meanings of E and E_d are formalized by the requirement that the interpretations for E and E_d are given by:

$$I(E)(d_1,d_2) \;=\; \begin{cases} 1 & : \text{ if } d_1 = d_2 \\ 0 & : \text{ otherwise} \end{cases} \qquad I(E_d)(d') \;=\; \begin{cases} 1 & : \text{ if } d = d' \\ 0 & : \text{ otherwise.} \end{cases}$$

For example, $\mathit{term}(d,z,z)$ stands short for $\mathit{term}'(z)$ where

$$\mathit{term}' \;=\; \lambda z \left[\sum_{\zeta_1,\zeta_2,\zeta_3} [\ \mathit{term}(\zeta_1,\zeta_2,\zeta_3) * \mathit{bexpr}\ ] \right]$$

and $\mathit{bexpr} \;=\; (\zeta_1 = d) \;\wedge\; (\zeta_2 = z) \;\wedge\; (\zeta_3 = z)$ where we write $\zeta_1 = d$ rather than $E_d(\zeta_1)$ and $\zeta_i = z$ rather than $E(\zeta_i, z)$, $i = 2, 3$.

Example 10.1.11 [Computing the probabilities $\mathit{Prob}(s,\tau^*,C)$] Let $\mathcal{S} = (S, \mathit{Act}, \mathbf{P})$ be a finite action-labelled fully probabilistic system and R an equivalence relation on S. We show how the probabilities $\mathit{Prob}(s,\tau^*,C)$ (where $s \in S$ and $C \in S/R$) can be described by a term of the algebraic mu-calculus.[10] For this, we use the following fact. The function $S \times S \to [0,1]$, $(s,t) \mapsto \mathit{Prob}(s,\tau^*,[t]_R)$, is the least fixed point of the

[10] Recall that $\mathit{Prob}(s,\tau^*,C)$ denotes the probability for s to reach a C-state via internal actions, see page 50.

operator $F : (S \times S \to [0,1]) \to (S \times S \to [0,1])$ that is given by $F(f)(s,t) = 1$ if $(s,t) \in R$ and, if $(s,t) \notin R$,

$$F(f)(s,t) \;=\; \mathbf{P}(s,\tau,[t]_R) \;+\; \sum_{u \in S \setminus [t]_R} \mathbf{P}(s,\tau,u) \cdot f(u,t)$$

(cf. Proposition 3.3.4, page 49). Here, $[t]_R = \{t' \in S : (t,t') \in R\}$. By Tarski's fixed point theorem, the least fixed point is obtained as limit of the function sequence $(F^i(f_0))_{i \geq 0}$ where $f_0(s,t) = 0$ for all $s,t \in S$. This iteration can be described by an algebraic term of the form $\mathit{lim}\ Z\ [\ldots \uparrow \lambda s,t[0]]$. For this, we rewrite the definition of F. Let f_R be the characteristic function of R. Then,

$$F(f)(s,t) \;=\; \max\ \{\ f_R(s,t),\ (1 - f_R(s,t)) * G(f)(s,t)\ \}$$

where

$$G(f)(s,t) \;=\; \sum_{t' \in S} \mathbf{P}(s,\tau,t') \cdot f_R(t,t') \;+\; \sum_{u \in S} \mathbf{P}(s,\tau,u) \cdot (1 - f_R(t,u)) \cdot f(u,t).$$

We use a ternary function symbol P (that represents $\mathbf{P}$) and a binary function symbol R (that represents R). We consider the model $\mathcal{M} = (D, I)$ where $D = S \uplus \mathit{Act}$ and

$$\begin{array}{rcl} I(P)(s,a,t) & = & \begin{cases} \mathbf{P}(s,a,t) & : \text{ if } s,\ t \in S \text{ and } a \in \mathit{Act} \\ 0 & : \text{ otherwise} \end{cases} \\ I(R)(s,t) & = & \begin{cases} 1 & : \text{ if } s,\ t \in S \text{ and } (s,t) \in R \\ 0 & : \text{ otherwise.} \end{cases} \end{array}$$

We use s, t, t' and u as individual variables and define

$$\mathit{term} \;=\; \mathit{lim}\ Z\ [\ \lambda s,t\ [\ \max\{\ R(s,t),\ (1 - R(s,t)) * \mathit{expr}\ \}\ ]\ \uparrow\ \lambda s,t[0]\ ]$$

where

$$\mathit{expr} \;=\; \sum_{t'} [\ P(s,\tau,t') * R(t,t')\ ] \;+\; \sum_{u} [\ P(s,\tau,u) * (1 - R(t,u)) * Z(u,t)\ ].$$

Here, in the expressions $P(s,\tau,t')$ and $P(s,\tau,u)$, we use the notation $\mathit{term}(\kappa_1,\ldots,\kappa_n)$ explained on page 267.[11] The meaning of *term* with respect to $\mathcal{M}$ is the function $[\![\mathit{term}]\!]^{\mathcal{M}} : D^2 \to \overline{\mathbb{R}}$ which is given by $[\![\mathit{term}]\!]^{\mathcal{M}}(s,t) = \mathit{Prob}(s,\tau^*,[t]_R)$ if $s,\ t \in S$. ■

Remark 10.1.12 There are several possible extensions of the algebraic mu-calculus that might be useful in certain applications.

- The algebraic mu-calculus could be extended by (total or partial) 1-ary arithmetic operators such as square root $\sqrt{\mathit{expr}}$, logarithms (such as $\log_2(\mathit{expr})$) or exponentiation (such as 2^{expr}) together with an appropriate semantics, e.g. in the case of square root

$$[\![\ \sqrt{\mathit{expr}}\ ]\!]^{\mathcal{M}} \;=\; \begin{cases} \sqrt{[\![\mathit{expr}]\!]^{\mathcal{M}}} & : \text{ if } [\![\mathit{expr}]\!]^{\mathcal{M}} \in \mathbb{R}_{\geq 0} \\ \bot & : \text{ otherwise.} \end{cases}$$

[11] Note that τ is an element of the domain D.

- Another possible extension is to deal with tuples of term variables in the limit or bounded iteration operator; i.e. to deal with an operator

$$lim_j\ \overline{Z}\ [\ \overline{term}\ \uparrow\ \overline{term}_0\ ]$$

where, for some natural number $l \geq 1$, $\overline{Z} = (Z_1, \ldots, Z_l)$ is a l-tuple of term variables, $\overline{term} = (term_1, \ldots, term_l)$ and $\overline{term}_0 = (term_{1,0}, \ldots, term_{l,0})$ are l-tuples of algebraic terms and $j \in \{1, \ldots, l\}$ such that the arity of Z_h, $term_h$ and $term_{h,0}$ is the same, $h = 1, \ldots, l$. The semantics of this limit operator with index j is given by $lim(f_{j,0}, f_{j,1}, f_{j,2}, \ldots)$ where $f_{h,0} = [\![term_{h,0}]\!]^{\mathcal{M}}$ and

$$f_{h,i+1} = [\![term_h]\!]^{\mathcal{M}[Z_1:=f_{1,i},\ldots,Z_l:=f_{l,i}]}, \quad h = 1, \ldots, l \text{ and } i = 0, 1, 2, \ldots.$$

Similarly, the bounded iteration operator could be extended for tuples of term variables.

- Instead of just using the symbol $\perp$ – that we use to handle all kinds of non-converging sequences – one might extend the real line by three symbols $-\infty$, $+\infty$ and $\perp$. This allows the distinction between sequences that diverge to $-\infty$ (e.g. $-1, -2, -3, \ldots$) and sequences that diverge to $+\infty$ (e.g. $1, 2, 3, \ldots$) and sequences that neither diverge to $+\infty$ or $-\infty$ nor converge to a real number (e.g. $-1, 1, -1, 1, \ldots$). ■

Notation 10.1.13 [**The range** $Range^{\mathcal{M}}(term)$] *Let term be an n-ary algebraic term and $\mathcal{M} = (D, I)$ a model. Then,*

$$Range^{\mathcal{M}}(term) \ = \ \left\{ [\![term]\!]^{\mathcal{M}}(\overline{d}) : \overline{d} \in D^n \right\}$$

denotes the range of (the semantics of) term with respect to $\mathcal{M}$.

Definition 10.1.14 [**$\Re$-models**] *Let $\Re$ be a subset of $\overline{\mathbb{R}}$. A model $\mathcal{M} = (D, I)$ is called a* $\Re$-model *iff $Range(I(Z))$, $Range(I(fct)) \subseteq \Re$ for all term variables Z and function symbols fct.*[12]

Definition 10.1.15 [**$\Re$-closedness**] *Let $\Re$ be a subset of $\overline{\mathbb{R}}$, $\mathcal{M} = (D, I)$ a model and Z an n-ary term variable. An algebraic term term is called* $\Re$-closed with respect to Z *and $\mathcal{M}$ iff $Range^{\mathcal{M}[Z:=f]}(term) \subseteq \Re$ for all functions $f : D^n \to \Re$. Similarly, an algebraic expression expr is said to be* $\Re$-closed with respect to Z *and $\mathcal{M}$ iff $[\![\text{expr}]\!]^{\mathcal{M}[Z:=f]} \in \Re$ for any function $f : D^n \to \Re$. An algebraic term or expression is called* $\Re$-closed *iff it is $\Re$-closed with respect to any term variable Z and any $\Re$-model (i.e. if $Range^{\mathcal{M}}(term) \subseteq \Re$ resp. $[\![\text{expr}]\!]^{\mathcal{M}} \in \Re$ for any $\Re$-model $\mathcal{M}$).*

Clearly, the subcalculus of $\Re$-closed expressions and terms contains any constant $q \in \Re \cap \mathbb{R}$ and is closed under λ-abstraction and term application and all those operators $op \in Op$ that are closed in $\Re$ (i.e. whenever q_1, $q_2 \in \Re$ then $q_1\ op\ q_2 \in \Re$).

Example 10.1.16 [**$\{0, 1, \perp\}$-closedness**] Given a $\{0, 1\}$-model $\mathcal{M} = (D, I)$, the semantics $[\![bexpr]\!]^{\mathcal{M}}$ for expressions of the boolean mu-calculus is either 0 or 1 or $\perp$. Similarly, the meaning $[\![bterm]\!]^{\mathcal{M}}$ of a term of the boolean mu-calculus is a partial boolean function, i.e. returns 0, 1 or $\perp$. Thus, each boolean expression or term is $\{0, 1, \perp\}$-closed. ■

[12] In the notations of Section 2.1, page 29, $Range(f)$ denotes the image of the function f, i.e. if $f : X \to Y$ then $Range(f) = f(X) = \{f(x) : x \in X\}$.

10.1.3 Fixed point operators

This section shows that under certain conditions the limit operator $lim\ Z\ [term \uparrow term_0]$ specializes to a fixed point operator in the sense that the semantics of $lim\ Z\ [term \uparrow term_0]$ is a certain fixed point of the higher-order function $f \mapsto [\![term]\!]^{\mathcal{M}[Z:=f]}$. The conditions that we present here are based on Banach's or Tarski's fixed point theorem which yield operators that describe *unique* or *least* or *greatest* fixed points.[13] To apply Banach's or Tarski's fixed point theorem we have to ensure that the higher-order operator $f \mapsto [\![term]\!]^{\mathcal{M}[Z:=f]}$ is a self-mapping of a complete metric space (in the case of Banach's fixed point theorem) or a complete lattice (in the case of Tarski's fixed point theorem). In both cases, we deal with function spaces of the form $D^n \to \Re$ where $\Re$ is a certain nonempty compact subset of $\mathbb{R}$, namely either a real interval $[a, b]$ or a finite nonempty set of reals. Then, $\Re$ – and hence, the function space $D^n \to \Re$ – is a complete metric space and a complete lattice.[14]

Proposition 10.1.17 *Let term be an n-ary algebraic term, Z an n-ary term variable and $\mathcal{M} = (D, I)$ a model for the algebraic mu-calculus. Let $\Re$ be a nonempty subset of $\mathbb{R}$ such that term is $\Re$-closed with respect to $\mathcal{M}$ and Z. Then, the operator*

$$F : (D^n \to \Re) \to (D^n \to \Re),\ F(f)\ =\ [\![term]\!]^{\mathcal{M}[Z:=f]},$$

is well-defined. Moreover:

(a) If F is contracting, $\Re$ is a compact interval $[a, b]$ and $term_0$ an n-ary algebraic term such that $Range^{\mathcal{M}}(term_0) \subseteq \Re$ then

$$[\![\ lim\ Z\ [term \uparrow term_0]\ ]\!]^{\mathcal{M}}\ =\ fix(F)$$

is the unique *fixed point of F.*

(b) Let $a = \min \Re$ and $b = \max \Re$ and $\Re$ either a compact interval or finite (i.e. either $\Re = [a, b]$ or $\Re = \{a_1, \ldots, a_k\}$ where $a = a_1 < a_2 < \ldots < a_k = b$). If F preserves suprema and infima then

$$[\![\ lim\ Z\ [\ term \uparrow\ \lambda z_1, \ldots, z_n[a]\ ]\ ]\!]^{\mathcal{M}}\ =\ lfp(F) \text{ is the least fixed point of } F,$$

$$[\![\ lim\ Z\ [\ term \uparrow\ \lambda z_1, \ldots, z_n[b]\ ]\ ]\!]^{\mathcal{M}}\ =\ gfp(F) \text{ is the greatest fixed point of } F.$$

Proof: follows immediately by Banach's and Tarski's fixed point theorem (see Section 12.1.2, page 310, and Section 12.1.1, page 308). ■

In the remainder of this section we present conditions that ensure that the conditions of part (b) of Proposition 10.1.17 are fulfilled. Clearly, if the operators *op* preserve suprema and infima for all subexpressions $expr_1\ op\ expr_2$ of *term* that contain a free occurrence of *Z* then the operator $f \mapsto [\![term]\!]^{\mathcal{M}[Z:=f]}$ preserves suprema and infima. But, in many applications, the requirement that only those operators *op* that preserve suprema and infima are allowed is not sufficient because the multiplication operator $*$ and the minus operator $-$ (and also the derived negation operator $\neg$) are not monotonic and hence do

[13] This observation justifies the name "mu-calculus" because usually (e.g. in the case of the relational mu-calculus) the greek letter "mu" denotes "least fixed point".

[14] We assume the natural metric $d(x, y) = |x - y|$ and the natural order $\leq$ on $\Re$. The function space $D^n \to \Re$ is equipped with the induced metric or partial order; see page 308 and page 310.

not preserve suprema and infima. In what follows, we shrink our attention to the compact interval $\Re = [0,1]$ and define a subcalculus of the algebraic mu-calculus which is $[0,1]$-closed and contains least and greatest fixed point operators. Similar conditions for other compact intervals $[a,b]$ can be derived using the bijection $[a,b] \to [0,1]$, $q \mapsto (q-a)/(b-a)$.

$[0,1]$-closedness requires the restriction to constants $q \in [0,1]$ and the use of such operators $op \in Op$ that are total operators on $[0,1]$ and always return values between 0 and 1 (i.e. operators that can be restricted to functions $[0,1] \times [0,1] \to [0,1]$). For this reason, we cannot use the addition operator $+$ or the minus operator $-$. Instead of the binary minus operator, we use the 1-ary operator $q \mapsto 1 - q$ and deal with expressions of the form $1 - expr$. One general possibility to handle summation is to combine the plus operator $+$ with the 1-ary minimum operator $q \mapsto \min\{1, q\}$ and to deal with the operator $(q_1, q_2) \mapsto \min\{1, q_1 + q_2\}$ which yields expressions of the form $\min\{1, expr_1 + expr_2\}$. Another possibility is to use boolean guards for the summation. For this, we deal with special function symbols *bfct* for which we assume an interpretation by a boolean-valued function and use expressions of the form

$$bfct(z_1, \ldots, z_n) * expr_1 \ + \ (1 - bfct(z_1, \ldots, z_n)) * expr_2.$$

Expressions of the above form can be read as "if $bfct(z_1, \ldots, z_n)$ then $expr_1$ else $expr_2$". Similarly, for the summation quantifier $\sum_z[expr]$ we could make the requirement that it can only be used in the context of the minimum operator, i.e. in the form $\min\{1, \sum_z[expr]\}$. Alternatively, we can generalize the idea of the boolean guards by non-negative weights that sum up to 1. This leads to a *weighted sum* e.g. of the form

$$\sum_{z_1,\ldots,z_k} [\ wfct(z_1, \ldots, z_k, y_1, \ldots, y_{n-k}) * expr \ ]$$

where $(z_1, \ldots, z_k, y_1, \ldots, y_{n-k})$ is a tuple of pairwise distinct individual variables and *wfct* a special function symbol that is interpreted in such a way that, whenever we fix interpretations $e_1, \ldots, e_{n-k}$ for $y_1, \ldots, y_{n-k}$ and sum up the values $I(wfct)(d_1, \ldots, d_k, e_1, \ldots, e_{n-k})$ where $d_1, \ldots, d_k$ range over all possible values for $z_1, \ldots, z_k$ then we obtain a value in $[0,1]$.

The algebraic $[0,1]$-mu-calculus: We assume a subset $Op^{[0,1]}$ of Op such that all operators $op \in Op^{[0,1]}$ can be restricted to an operator $[0,1] \times [0,1] \to [0,1]$. Moreover, we assume that $q \ op \perp$, $\perp \ op \ q$, $\perp \ op \perp \ \in [0,1] \cup \{\perp\}$ for any operator $op \in Op^{[0,1]}$. Let $BFct^n \subseteq Fct^n$ be a set of *boolean* function symbols. For $n \geq 1$ and $1 \leq i_1 < \ldots < i_k \leq n$, let $WFct^n(i_1, \ldots, i_k) \subseteq Fct^n$ be a set of n-ary function symbols. The algebraic $[0,1]$-mu-calculus is those subcalculus of the algebraic mu-calculus whose expressions and n-ary terms are built from the grammar shown in Figure 10.4 (page 272). Here, $q \in [0,1]$, op is an operator in $Op^{[0,1]}$, $wfct \in WFct^n(i_1, \ldots, i_k)$ and $bfct \in BFct^n$. The least and greatest fixed points operators are given by $lfp \ Z \ [term] \ = \ lim \ Z \ [term \uparrow \lambda z_1, \ldots, z_n[0]]$ and $gfp \ Z \ [term] \ = \ lim \ Z \ [term \uparrow \lambda z_1, \ldots, z_n[1]]$. The expression `if` $bfct(z_1, \ldots, z_n)$ `then` $expr_1$ `else` $expr_2$ stands short for

$$bfct(z_1, \ldots, z_n) * expr_1 \ + \ (1 - bfct(z_1, \ldots, z_n)) * expr_2.$$

A model for the algebraic $[0,1]$-mu-calculus is a $[0,1]$-model $\mathcal{M} = (D, I)$ for the algebraic mu-calculus such that

- $I(bfct)(\overline{d}) \in \{0,1\}$ for all $bfct \in BFct^n$ and $\overline{d} \in D^n$,

$$
\begin{array}{rcl}
expr & ::= & q \;\Big|\; expr_1\ op\ expr_2 \;\Big|\; 1 - expr \;\Big|\; term(z_1, \ldots, z_n) \;\Big| \\
 & & \texttt{if}\ bfct(z_1, \ldots, z_n)\ \texttt{then}\ expr_1\ \texttt{else}\ expr_2 \;\Big| \\
 & & \sum_{z_{i_1}, \ldots, z_{i_k}} [\ wfct(z_1, \ldots, z_n) * expr\ ] \;\Big|\; \min_z\ [expr] \;\Big|\; \max_z\ [expr] \\
term & ::= & fct \mid Z \;\Big|\; \lambda z_1, \ldots, z_n\, [expr] \;\Big|\; lfp\ Z\ [term] \;\Big|\; gfp\ Z\ [term] \;\Big| \\
 & & iterate\ Z\ \left[\ term \uparrow^k term_0\ \right]
\end{array}
$$

Figure 10.4: Syntax of the algebraic $[0,1]$-mu-calculus

- for all $wfct \in WFct^n(i_1, \ldots, i_k)$ and $d_i \in D$, $i \in \{1, \ldots, n\} \setminus \{i_1, \ldots, i_k\}$,
$$\sum_{d_{i_1}, \ldots, d_{i_k} \in D} I(wfct)(d_1, \ldots, d_n) \ \leq\ 1.$$

Example 10.1.18 The term in Example 10.1.11 (page 267) that describes the function $S \times S \to [0,1]$, $(s,t) \mapsto Prob(s, \tau^*, [t]_R)$ can be rewritten as

$$lfp\ Z\ [\ \lambda s,t\ [\ \texttt{if}\ R(s,t)\ \texttt{then}\ 1\ \texttt{else}\ expr\ ]\ ].$$

which is a term of the algebraic [0,1]-mu-calculus provided that $P \in WFct^3(2,3)$ and $R \in BFct^2$. The model $\mathcal{M} = (D, I)$ considered in Example 10.1.11 is a model for the algebraic [0,1]-mu-calculus. ■

It is easy to see that, for any expression *expr* of the algebraic $[0,1]$-mu-calculus and any model $\mathcal{M}$ for the algebraic $[0,1]$-mu-calculus, $[\![expr]\!]^{\mathcal{M}} \in [0,1] \cup \{\bot\}$. Similarly, if *term* is a term of the algeraic $[0,1]$-mu-calculus then $Range^{\mathcal{M}}(term) \subseteq [0,1] \cup \{\bot\}$ for any model $\mathcal{M}$ for the algebraic $[0,1]$-mu-calculus. We now present conditions that ensure that the semantics returns values in $[0,1]$ rather than the auxiliary symbol $\bot$. For this, we make some syntactic requirements about the occurrences of the term variable Z within subterms *lfp* Z [*term*] and *gfp* Z [*term*]. The first condition ("formal monotonicity") is taken from the relational mu-calculus where, for the least and greatest fixed point operators *lfp* Z [*bterm*] and *gfp* Z [*bterm*], the monotonicity of the induced operator $f \mapsto [\![bterm]\!]^{\mathcal{M}[Z:=f]}$ is ensured by the requirement that all free occurrences of Z in *bterm* fall under an even number of the negation operator $\neg bexpr$.

Definition 10.1.19 [Formal monotonicity] *Let term be an algebraic term, expr an algebraic expression and Z an n-ary term variable. Z is* formally monotone *in term (or* expr*) iff all free occurrences of Z in* term *(resp.* expr*) fall under an even number of minus operations.*[15]

[15] The number of minus operations under which an occurrence of a term variable Z falls in an expression or term is given by the number of subexpressions $1 - expr$ of that expression or term where the occurence of Z is contained in *expr*.

Example 10.1.20 Let Z be a 1-ary term variable, *fct* a 1-ary function symbol and z, y individual variables. Z is formally monotone in the expressions $\frac{1}{3} * Z(z) * (1 - fct(y))$ and $Z(z) * (1 - (1 - Z(z)))$ while it is not in $(1 - Z(z)) * (1 - Z(z))$. In the third expression, both occurrences of Z fall under an odd number of minus operations. In the expression $(1 - Z(z)) * Z(z)$, the first occurrence of Z falls under an odd number of minus operations (which yields that Z is not formally monotone in $(1 - Z(z)) * Z(z))$ while the second occurrence falls under an even number of minus operations. ■

Remark 10.1.21 Formal monotonicity of term variables in terms or expressions of the relational mu-calculus (as a subcalculus of the algebraic mu-calculus) in the sense of Definition 10.1.19 is the same as formal monotonicity à la Park [Park74]. ■

The second condition that we need to ensure that the function $f \mapsto [\![term]\!]^{\mathcal{M}[Z:=f]}$ preserves suprema and infima is that, for each subexpression $expr_1\ op\ expr_2$, if Z occurs free in $expr_i$ then the operator op preserves suprema and infima in the i-th argument. To formalize this condition, we define the operator sets $Op_1^{[0,1]}$ and $Op_2^{[0,1]}$ as follows. Let $Op_1^{[0,1]}$ be a set of operators $op \in Op^{[0,1]}$ such that the function $[0,1] \times [0,1] \to [0,1]$, $(q_1, q_2) \mapsto q_1\ op\ q_2$, preserves suprema and infima in its first argument, i.e. whenever Q is a nonempty subset of $[0,1]$ with $q^+ = \sup Q$ and $q^- = \inf Q$ and $q_2 \in [0,1]$ then

$$\sup \{q_1\ op\ q_2 : q_1 \in Q\} \ = \ q^+\ op\ q_2, \quad \inf \{q_1\ op\ q_2 : q_1 \in Q\} \ = \ q^-\ op\ q_2.$$

Similarly, $Op_2^{[0,1]}$ denotes a set of operators $op \in Op^{[0,1]}$ where the function $[0,1] \times [0,1] \to [0,1]$, $(q_1, q_2) \mapsto q_1\ op\ q_2$, preserves suprema and infima in its second argument. Clearly, any operator $op \in Op_i^{[0,1]}$ is monotonic in the i-th argument. For instance, multiplication $*$, minimum op_{min} and maximum op_{max} belong to $Op_1^{[0,1]} \cap Op_2^{[0,1]}$ while the operator $(q_1, q_2) \mapsto q_1/(1 + q_2)$ is contained in $Op_1^{[0,1]} \setminus Op_2^{[0,1]}$ (provided that it belongs to $Op^{[0,1]}$). The comparison operators $op_{\bowtie}$ are not contained in $Op_1^{[0,1]}$ or $Op_2^{[0,1]}$.[16]

Definition 10.1.22 [Formal continuity] *Let Z be an n-ary term variable. Z is called* formally continuous *in a term or expression of the algebraic $[0,1]$-mu-calculus iff, for any free occurrence of Z in that term or expression within a subexpression $\mathrm{expr}_1\ op\ \mathrm{expr}_2$:*

$$\textit{If } Z \textit{ occurs free in } \mathrm{expr}_i \textit{ then } op \in Op_i^{[0,1]}.$$

Example 10.1.23 Let Z, Y be 1-ary term variables. Z is formally continuous in the expressions $\frac{1}{2} * (1 - Z(z))$ and

$$\min \{Z(z), Z(z) * Z(z)\} \ * \ \left(\ (\tfrac{1}{3} * Y(y))\ op_< \ \tfrac{2}{3} \ \right)$$

while it is not in $Z(z)\ op_< \ \frac{2}{5}$. ■

Remark 10.1.24 Clearly, the algebraic $[0,1]$-mu-calculus subsumes the boolean mu-calculus. In the boolean mu-calculus, only the operators $\vee = op_{max}$, $\wedge = op_{min}$ and the comparison operators $op_{\bowtie}$ are allowed. Thus, if *bterm* is a term of the boolean mu-calculus then Z is formally continuous in *bterm* iff there is no free occurrence of Z within a subexpression of the form $expr_1\ op_{\bowtie}\ expr_2$. In particular, for the relational mu-calculus à

[16] Note that the comparison operators $op_{\bowtie}$ are not monotonic. For instance, $0.4\ op_>\ 0.3 \ = \ 1$ while $0.4\ op_>\ 0.5 \ = \ 0$ although $0.3 < 0.5$.

la Park (the boolean mu-calculus without the comparison operators $op_{\bowtie}$ and the bounded iteration operator), all expressions and terms are formally continuous. ■

Definition 10.1.25 [Formal divergence freedom] *A term or expression of the algebraic $[0,1]$-mu-calculus is called* formally divergence free *iff, for each subterm lfp Z [term] or gfp Z [term], Z is formally monotone and formally continuous in term.*

Example 10.1.26 The term in Example 10.1.11 (page 267) that describes the function $S \times S \to [0,1]$, $(s,t) \mapsto Prob(s, \tau^*, [t]_R)$ is formally divergence free when we deal with $P \in WFct^3(2,3)$ and $R \in BFct^2$. ■

Theorem 10.1.27 *Let term be an n-ary term of the algebraic $[0,1]$-mu-calculus that is formally divergence free. Let $\mathcal{M} = (D, I)$ be a model for the algebraic $[0,1]$-mu-calculus and Z an n-ary term variable. Then, the function*

$$F : (D^n \to [0,1]) \to (D^n \to [0,1]), \quad F(f) \;=\; [\![term]\!]^{\mathcal{M}[Z:=f]},$$

is well-defined. Moreover, if Z is formally monotone and formally continuous in term then F preserves suprema and infima and we have the following.

(a) $[\![\ lfp\ Z[term]\ ]\!]^{\mathcal{M}} \;=\;$ *$lfp(F)$ is the least fixed point of F,*

(b) $[\![\ gfp\ Z[term]\ ]\!]^{\mathcal{M}} \;=\;$ *$gfp(F)$ is the greatest fixed point of F.*

Proof: We say that an n-ary term variable Z is formally antitone in *expr* iff Z is formally monotone in $1 - expr$. Z is called formally antitone in a k-ary term *term* iff Z is formally antitone in the expression $term(z_1, \ldots, z_k)$. Let l, m be natural numbers with $l + m \geq 1$. We say that a function

$$F : (D^{n_1} \to [0,1]) \times \ldots \times (D^{n_{l+m}} \to [0,1]) \to [0,1]$$

is (l,m)-continuous iff, for all nonempty subsets Ξ_i of functions $f_i : D^{n_i} \to [0,1]$,

$$F\left(f_1^+, \ldots, f_l^+, f_{l+1}^-, \ldots, f_{l+m}^-\right) \;=\; \sup\ \{F(f_1, \ldots, f_{l+m}) : f_i \in \Xi_i, i = 1, \ldots, l+m\}\,,$$

$$F\left(f_1^-, \ldots, f_l^-, f_{l+1}^+, \ldots, f_{l+m}^+\right) \;=\; \inf\ \{F(f_1, \ldots, f_{l+m}) : f_i \in \Xi_i, i = 1, \ldots, l+m\}\,.$$

Here, $f_i^+ \;=\; \sup_{f \in \Xi_i} f$ and $f_i^- \;=\; \inf_{f \in \Xi_i} f$. Similarly, we define (l,m)-continuity for functions

$$F : (D^{n_1} \to [0,1]) \times \ldots \times (D^{n_{l+m}} \to [0,1]) \to (D^n \to [0,1]).$$

By structural induction on the expressions and terms of the algebraic $[0,1]$-mu-calculus we get the following. Whenever $Z_1, \ldots, Z_{l+m}$ are pairwise distinct term variables and a is an algebraic expression or term such that a is formally divergence free and

- $Z_1, \ldots, Z_{l+m}$ are formally continuous is a,
- $Z_1, \ldots, Z_l$ are formally monotone in a,
- $Z_{l+1}, \ldots, Z_{l+m}$ are formally antitone in a

then the function F_a is (l,m)-continuous. Here, F_a is given by

$$F_a(f_1, \ldots, f_{l+m}) \;=\; [\![a]\!]^{\mathcal{M}[Z_1:=f_1,\ldots,Z_{l+m}:=f_{l+m}]}.$$

With $a = term$, $l = 1$, $m = 0$, $Z_1 = Z$ we get that the operator $F = F_a$ preserves suprema and infima. Parts (a), (b) can be derived from Proposition 10.1.17(b) (page 270). ■

10.2 The algebraic mu-calculus as a specification language

In this section we show that the algebraic mu-calculus subsumes several temporal or modal logics that can serve as specification languages for (several types of) parallel systems. The relational mu-calculus with its standard semantics à la Park [Park74] and Kozen's modal mu-calculus [Koze83] can be viewed as subcalculi of the boolean mu-calculus (see Sections 10.2.1 and 10.2.2). In particular, the boolean mu-calculus (and hence, the algebraic mu-calculus) has the expressiveness of all formalisms – e.g. automata on infinite words and several kinds of temporal logics such as *CTL* or *LTL* – that are contained in the relational or modal mu-calculus; see e.g. [StEm84, EmLei86, Niwi88, BCM$^+$90, EmJu91, Dam94]. Moreover, the algebraic mu-calculus contains several temporal and modal logics for reasoning about *quantitative* properties of concurrent systems. For instance, the algebraic mu-calculus subsumes the extensions of Emerson [Emer92] and Seidl [Seidl96] of Kozen's modal mu-calculus for specifying (certain kind of) real time properties and several logics to reason about probabilistic systems such as the probabilistic mu-calculus à la [HuKw97, HuKw98] or *PCTL* [HaJo94, BidAl95]; see Sections 10.2.2 and 10.2.3. Moreover, the algebraic mu-calculus can serve as specification language for arithmetic circuits as it subsumes the temporal logic Word Level *CTL* [CCH$^+$96, CKZ96, Zhao96]; see Section 10.2.4. In all these cases, we have an *embedding* of the respective (temporal or modal) logic $\mathbf{L}$ into the algebraic mu-calculus of the following form. For each formula φ of $\mathbf{L}$, there is an "equivalent" algebraic term $term_\varphi$. Here, "equivalence" is in the following sense: For any model $\mathcal{N}$ for $\mathbf{L}$, there is a model $\mathcal{M}$ for the algebraic mu-calculus with

$$(*) \qquad [\![term_\varphi]\!]^{\mathcal{M}} \;=\; [\![\varphi]\!]^{\mathcal{N}}.$$

Hence, all properties that can be specified by a formula of $\mathbf{L}$ can also be expressed in the algebraic mu-calculus. Moreover, the definition of $term_\varphi$ is by structural induction on the syntax of φ; i.e. the definition of $term_\varphi$ is constructive. Thus, any method that automatically computes the semantics of the terms of the algebraic mu-calculus yields a *model checker* for $\mathbf{L}$.

10.2.1 The relational mu-calculus

The syntax of Park's relational mu-calculus is obtained from the boolean mu-calculus (see page 261) by removing the expressions built from the comparison operators $op_{\bowtie}$ and the bounded iteration operator *iterate* Z $[\ldots \uparrow^k \ldots]$. We now show that the semantics of the relational mu-calculus (as a subcalculus of the algebraic or boolean mu-calculus where the semantics of the least and greatest fixed point operators are defined as limits of certain function sequences) agrees with the standard semantics à la Park [Park74]. In the approach of Park, the use of the least and greatest fixed point *lfp* Z $[bterm]$ and *gfp* Z $[bterm]$ operators is restricted to those relational terms *bterm* and term variables Z where Z is formally monotone in *bterm*.[17] The meanings of *lfp* Z $[bterm]$ and *gfp* Z $[bterm]$ with respect to a $\{0,1\}$-model $\mathcal{M} = (D, I)$ (which can be viewed as a model for the

[17] Recall that formal monotonicity of a term variable Z in a relational term *bterm* means that all free occurrences of Z in *bterm* fall under an even number of the negation operator $\neg\, bexpr$.

relational mu-calculus in the sense of Park) are defined as the least and greatest fixed points of the higher-order operator $f \mapsto [\![bterm]\!]^{\mathcal{M}[Z:=f]}$ on the function space $D^n \to \{0,1\}$. To see that the operators *lfp* Z $[bterm]$ and *gfp* Z $[bterm]$ of the boolean mu-calculus are indeed least and greatest fixed point operators we apply Theorem 10.1.27 (page 274).[18]

Theorem 10.2.1 *Let bterm be a n-ary term of the boolean mu-calculus that is formally divergence free. Let Z be an n-ary term variable that is formally monotone and formally continuous in bterm. Then, for any $\{0,1\}$-model $\mathcal{M}$:*

(a) $[\![$ *lfp* Z $[bterm]$ $]\!]^{\mathcal{M}}$ $=$ *$lfp(F)$ is the least fixed point of F*

(b) $[\![$ *gfp* Z $[bterm]$ $]\!]^{\mathcal{M}}$ $=$ *$gfp(F)$ is the greatest fixed point of F*

where $F : (D^n \to \{0,1\}) \to (D^n \to \{0,1\})$ *is given by* $F(f) = [\![bterm]\!]^{\mathcal{M}[Z:=f]}$.

Proof: follows immediately by Theorem 10.1.27 (page 274).[19] ■

Since formal continuity in expressions and terms of the relational mu-calculus is always satisfied (see Remark 10.1.24, page 273), Theorem 10.2.1 yields that the standard semantics for the relational mu-calculus à la Park agrees with the semantics of the relational mu-calculus when viewed as a sublanguage of the algebraic mu-calculus. Thus, the relational mu-calculus can be viewed as a subcalculus of the algebraic mu-calculus.

10.2.2 The modal mu-calculus

The modal mu-calculus was introduced by Kozen [Koze83] as a language for analyzing the behaviour of possibly infinite computations. Formulas of the modal mu-calculus are built from the boolean connectives $\wedge$, $\vee$, $\neg$, modal next step operators $\langle\alpha\rangle$ and $[\alpha]$ (where α ranges over certain actions) and least or greatest fixed point operators. They are interpreted by *sets* of states of a finite action-labelled transition system and might express e.g. safety or liveness properties. The modal next step operator $\langle\alpha\rangle$ can be viewed as the modal counterpart to the boolean quantifier $\exists x$ and states that "there is an α-labelled transition" while $[\alpha]$ is its dual ("for all α-labelled transitions"). Using 2-ary function symbols R_α – where R_α represents the characteristic function of the transition relation for the action label α (i.e. we assume an interpretation I such that $I(R_\alpha)$ is a boolean-valued function where $I(R_\alpha)(s,t)$ is true iff $s \xrightarrow{\alpha} t$) – each modal mu-formula φ can be translated into an "equivalent" 1-ary boolean term $bterm_\varphi$. For instance, for mu-formulas with modal next step $\langle\alpha\rangle$ or least fixed points,

$$bterm_{\langle\alpha\rangle\varphi} \;=\; \lambda z\,[\,\exists z'\,[\,R_\alpha(z,z')\;\wedge\;bterm_\varphi(z')\,]\,],\quad bterm_{lfp\ Z\ [\varphi]} \;=\; lfp\ Z\ [bterm_\varphi].$$

[18] It should be observed that the boolean mu-calculus is a subcalculus of the algebraic $[0,1]$-mu-calculus. Note that the summation quantifier $\sum_z$ is not contained in the boolean mu-calculus, only the operators $\wedge = op_{min}$, $\vee = op_{max}$ and $op_{\bowtie}$ are allowed and the boolean negation operator $\neg bexpr$ is modelled by $1 - bexpr$.

[19] For this, we use the following simple fact. If $F : (D^n \to [0,1]) \to (D^n \to [0,1])$ is an operator that preserves suprema and infima and such that $F(f)(d_1,\ldots,d_n) \in \{0,1\}$ for any function $f : D^n \to \{0,1\}$ and $d_1,\ldots,d_n \in D$ then the least and greatest fixed points of F are functions with range $\{0,1\}$.

Here, "equivalence" is in the sense of condition (*) on page 275. Thus, Kozen's modal mu-calculus can be viewed as a sublanguage of the boolean mu-calculus.

In the literature, Kozen's modal mu-calculus has been extended to reason about quantitaive properties of timed systems [Emer92, Seidl96] or probabilistic systems [HuKw97, MoMcI97, HuKw98, McIv98]. In the approach of [Emer92], formulas are still interpreted over *sets* of states of a labelled transition system while [Seidl96, HuKw97, MoMcI97, HuKw98, McIv98] deal with an interpretation by *functions* from the states into the reals. Emerson [Emer92] extends the modal mu-calculus by bounded iteration operators that are used to formulate real time properties such as "the process will terminate within the next k time units". The above mentioned embedding of Kozen's modal mu-calculus in the boolean mu-calculus can be extended to an embedding of Emerson's modal mu-calculus where we describe Emerson's bounded iteration operator with the help of our bounded iteration operator *iterate* $Z\ [\ldots \uparrow^k \ \ldots]$. In the remainder of this section, we briefly explain how the modal mu-calculus with the interpretations by Seidl [Seidl96] over durational transition systems and Huth & Kwiatkowska [HuKw97, HuKw98] over probabilistic (reactive) systems can be embedded into the algebraic mu-calculus.

The durational mu-calculus à la Seidl: [Seidl96] deals with an interpretation of formulas by functions from the states of a finite action-labelled transition system into a discrete time domain *Time*. The transitions are endowed with an *duration* (i.e. the amount of time that is needed to perform the transitions). In some sense, the semantics of the formulas gives a measure for the time of how long a certain property holds.

The time domain *Time* is a subinterval of the non-negative integers extended by $\perp$ that we treat as ∞.[20] Moreover, *Time* is equipped with a set *Op* of binary operators such as maximim op_{max}, minimum op_{min}, addition $+$ and a sequence operator $(x,y) \mapsto x;y = y$. Formulas are given by the following grammar

$$\varphi \ ::= \ bta \mid Z \mid \varphi_1 \ op \ \varphi_2 \mid [\alpha]\varphi \mid \langle\alpha\rangle\varphi \mid lfp\ Z\ [\varphi] \mid gfp\ Z\ [\varphi]$$

where bta is a *basic time assignment*, $\alpha \in Act$, Z a variable and $op \in Op$. Here, *Act* is a fixed finite set of actions. Formulas are interpreted over the states of a *durational transition system*, i.e. a tuple $\mathcal{S} = (S, \rightarrow, dur)$ where S is a finite set of states, $\rightarrow \subseteq S \times Act \times S$ a transition relation and dur a function that assigns to each transition $s \stackrel{\alpha}{\rightarrow} t$ its duration $dur(s,\alpha,t)$. A model $\mathcal{N} = (\mathcal{S}, J)$ consists of a durational transition system $\mathcal{S} = (S, \rightarrow, dur)$ and an interpretation J for the variables and basic time assignments by functions $S \rightarrow Time$. The meaning $[\![\varphi]\!]^{\mathcal{N}} : S \rightarrow Time$ is defined as shown in Figure 10.5 (page 278). We associate with each formula φ an 1-ary term $term_\varphi$ as follows. Each basic time assignment bta is viewed as a 1-ary function symbol. For each action α, we use binary function symbols R_α and dur_α. Intuitively, R_α represents the the α-labelled transitions (i.e. R_α stands for a boolean-valued function $S \times S \rightarrow \{0,1\}$ where $(s,t) \mapsto 1$ is true iff $s \stackrel{\alpha}{\longrightarrow} t$) while dur_α stands for the duration of the α-labelled transitions (i.e. dur_α represents the partial function $S \times S \rightarrow \overline{I\!R}$ where $(s,t) \mapsto dur(s,\alpha,t)$ if $s \stackrel{\alpha}{\longrightarrow} t$). The variables Z of the durational mu-calculus are viewed as 1-ary term variables.

$$term_Z = Z, \quad term_{bta} = bta, \quad term_{\varphi_1 op \varphi_2} \ = \ \lambda s[term_{\varphi_1}(s) \ op \ term_{\varphi_2}(s)],$$

[20] To be precisely, [Seidl96] also uses the symbol $-\infty$ to express unaccessibility. Using an extension of the real line by the three symbols $\pm\infty$ and $\perp$ rather than just the single symbol $\perp$, we could also deal with $-\infty$.

$$[\![Z]\!]^{\mathcal{N}} = J(Z), \quad [\![bta]\!]^{\mathcal{N}} = J(bta), \quad [\![\varphi_1 \ op \ \varphi_2]\!]^{\mathcal{N}}(s) = [\![\varphi_1]\!]^{\mathcal{N}}(s) \ op \ [\![\varphi_2]\!]^{\mathcal{N}}(s)$$
$$[\![[\alpha]\varphi]\!]^{\mathcal{N}}(s) = \min \left\{ dur(s,\alpha,t) + [\![\varphi]\!]^{\mathcal{N}}(t) : s \xrightarrow{\alpha} t \right\}$$
$$[\![\langle\alpha\rangle\varphi]\!]^{\mathcal{N}}(s) = \max \left\{ dur(s,\alpha,t) + [\![\varphi]\!]^{\mathcal{N}}(t) : s \xrightarrow{\alpha} t \right\}$$
$$[\![lfp\ Z\ [\varphi]]\!]^{\mathcal{N}} = \text{least fixed point of } (S \to Time) \to (S \to Time),\ f \mapsto [\![\varphi]\!]^{\mathcal{N}[Z:=f]}$$
$$[\![gfp\ Z\ [\varphi]]\!]^{\mathcal{N}} = \text{greatest fixed point of } (S \to Time) \to (S \to Time),\ f \mapsto [\![\varphi]\!]^{\mathcal{N}[Z:=f]}$$

Figure 10.5: Semantics of the durational mu-calculus à la [Seidl96]

$$term_{[\alpha]\varphi} = \lambda s [\ \min_t \ [\ R_\alpha(s,t) * (dur_\alpha(s,t) + term_\varphi(t))\ ]\ ],$$
$$term_{\langle\alpha\rangle\varphi} = \lambda s [\ \max_t \ [\ R_\alpha(s,t) * (dur_\alpha(s,t) + term_\varphi(t))\ ]\ ],$$
$$term_{lfp\ Z\ [\varphi]} = lim\ Z [\ term_\varphi \uparrow \lambda s[time_{min}]\ ],$$
$$term_{gfp\ Z\ [\varphi]} = lim\ Z\ [\ term_\varphi \uparrow \lambda s[time_{max}]\ ].$$

Here, $time_{min} = \min\{t : t \in Time\}$ and $time_{max} = \max\{t : t \in Time\}$. Given a model $\mathcal{N} = (\mathcal{S}, J)$ where $\mathcal{S}$ is as before, we define a model $\mathcal{M} = (S, I)$ for the algebraic mu-calculus by $I(bta) = J(bta)$,

$$I(R_\alpha)(s,t) = \begin{cases} 1 & : \text{ if } s \xrightarrow{\alpha} t \\ 0 & : \text{ otherwise} \end{cases} \qquad I(dur_\alpha)(s,t) = \begin{cases} dur(s,\alpha,t) & : \text{ if } s \xrightarrow{\alpha} t \\ \bot & : \text{ otherwise} \end{cases}$$

and $I(Z) = J(Z)$ for all variables Z. By structural induction on φ, we get $[\![\varphi]\!]^{\mathcal{N}} = [\![term_\varphi]\!]^{\mathcal{M}}$.

The probabilistic mu-calculus à la Huth & Kwiatkowska: In the probabilistic mu-calculus of [HuKw97, HuKw98], formulas are given by the grammar

$$\varphi \ ::= \ ap \mid Z \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid [\alpha]\varphi \mid \langle\alpha\rangle\varphi \mid lfp\ Z\ [\varphi] \mid gfp\ Z\ [\varphi]$$

where ap is an atomic proposition, $\alpha \in Act$ an action and Z a variable. Formulas are interpreted with respect to a model $\mathcal{N} = (\mathcal{S}, J)$ consisting of a reactive system $\mathcal{S} = (S, Act, \mathbf{P})$ (cf. Definition 3.3.10 on page 51 and Notation 3.3.11 on page 52) and an interpretation J for the atomic propositions ap and the variables Z by functions $S \to [0,1]$. The meaning $[\![\varphi]\!]^{\mathcal{N}} : S \to [0,1]$ of a formula φ is defined as shown in Figure 10.6 (page 279). For disjunction $\vee$ and conjunction $\wedge$, several interpretations by binary operators are possible. To guarantee the existence of least/greatest fixed points for formulas with alternating depth ≤ 1 the following operators $op_\vee$ and $op_\wedge$ can be used.[21]

- $op_\vee$ is op_{max} or one of the operators $(q_1, q_2) \mapsto q_1 + q_2 - q_1 \cdot q_2$, $(q_1, q_2) \mapsto \min\{1, q_1 + q_2\}$
- $op_\wedge$ is op_{min}, $*$ or the operator $(q_1, q_2) \mapsto \max\{q_1 + q_2 - 1, 0\}$.

Similarly to the way in which we describe each formula φ of Seidl's durational mu-calculus by an "equivalent" algebraic term $term_\varphi$, we obtain a transformation from the positive

[21] Alternating depth ≤ 1 means that, for any subformula $lfpZ[\psi]$ or $lfpZ[\psi]$, at most the variable Z occurs free in ψ. This ensures the existence of least and greatest fixed points of the associated operator and that they can be computed by the standard iterations. See Proposition 1 and 2 in [HuKw98].

$$
\begin{array}{l}
[\![Z]\!]^{\mathcal{N}} = J(Z), \qquad [\![ap]\!]^{\mathcal{N}} = J(ap), \qquad [\![\neg\varphi]\!]^{\mathcal{N}}(s) \;=\; 1 - [\![\varphi]\!]^{\mathcal{N}}(s), \\
[\![\varphi_1 \vee \varphi_2]\!]^{\mathcal{N}} = [\![\varphi_1]\!]^{\mathcal{N}} \; op_{\vee} \; [\![\varphi_2]\!]^{\mathcal{N}}, \; [\![\varphi_1 \wedge \varphi_2]\!]^{\mathcal{N}} = [\![\varphi_1]\!]^{\mathcal{N}} \; op_{\wedge} \; [\![\varphi_2]\!]^{\mathcal{N}}, \\
[\![\; \langle\alpha\rangle\varphi \;]\!]^{\mathcal{N}}(s) \;=\; \sum_{t\in S} \mathbf{P}(s,\alpha,t) \cdot [\![\varphi]\!]^{\mathcal{N}}(t), \\
[\![\; [\alpha]\varphi \;]\!]^{\mathcal{N}}(s) \;=\; 1 - \sum_{t\in S} \mathbf{P}(s,\alpha,t) \cdot (1 - [\![\varphi]\!]^{\mathcal{N}}(t)), \\
[\![\; \mathit{lfp}\; Z\; [\varphi] \;]\!]^{\mathcal{N}} = \text{least fixed point of } (S \to [0,1]) \to (S \to [0,1]),\; f \mapsto [\![\varphi]\!]^{\mathcal{N}[Z:=f]}, \\
[\![\; \mathit{gfp}\; Z\; [\varphi] \;]\!]^{\mathcal{N}} = \text{greatest fixed point of } (S \to [0,1]) \to (S \to [0,1]),\; f \mapsto [\![\varphi]\!]^{\mathcal{N}[Z:=f]}.
\end{array}
$$

Figure 10.6: Semantics of the probabilistic mu-calculus à la [HuKw97]

modal mu-calculus with the Fuzzy interpretations of [HuKw97, HuKw98] to the algebraic mu-calculus. For each formula φ, we define an 1-ary algebraic term $term_\varphi$ by structural induction. The atomic propositions ap are viewed as 1-ary function symbols. For each action α, we use a binary function symbols P_α. Intuitively, $P_\alpha(s,t)$ stands for the probability for s to move to t via an α-labelled transition, i.e. P_α represents the function $S \times S \to [0,1]$, $(s,t) \mapsto \mathbf{P}(s,\alpha,t)$. The variables Z of the positive modal mu-calculus are viewed as 1-ary term variables. We suppose that $op_\vee$, $op_\wedge \in Op$ and define $term_Z = Z$, $term_{ap} = ap$, $term_{\neg\varphi} = 1 - term_\varphi$ and

$$
\begin{array}{l}
term_{\varphi_1\vee\varphi_2} = \lambda s[\; term_{\varphi_1}(s) \; op_\vee \; term_{\varphi_2}(s) \;], \\
term_{\varphi_1\wedge\varphi_2} = \lambda s[\; term_{\varphi_1}(s) \; op_\wedge \; term_{\varphi_2}(s) \;], \\
term_{\langle\alpha\rangle\varphi} \;=\; \lambda s [\; \sum_t \; [\; P_\alpha(s,t) * term_\varphi(t) \;] \;], \\
term_{[\alpha]\varphi} \;=\; \lambda s [\; 1 - \sum_t \; [\; P_\alpha(s,t) * (1 - term_\varphi(t)) \;] \;], \\
term_{\mathit{lfp}\; Z\; [\varphi]} \;=\; \mathit{lim}\; Z [\; term_\varphi \; \uparrow \lambda s[0] \;], \quad term_{\mathit{gfp}\; Z\; [\varphi]} \;=\; \mathit{lim}\; Z [\; term_\varphi \; \uparrow \lambda s[1] \;],
\end{array}
$$

Given a model $\mathcal{N} = (\mathcal{S}, J)$ where $\mathcal{S} = (S, Act, \mathbf{P})$, we define a model $\mathcal{M} = (S, I)$ for the algebraic mu-calculus as follows. For each variable Z, we define $I(Z) = J(Z)$; similarly, $I(ap) = J(ap)$ for all atomic propositions ap. For the binary function symbols P_α, we define $I(P_\alpha)(s,t) = \mathbf{P}(s,\alpha,t)$. By structural induction on φ it can be shown that $[\![\varphi]\!]^{\mathcal{N}} \;=\; [\![term_\varphi]\!]^{\mathcal{M}}$. Here, we assume that φ is a formula of alternating depth ≤ 1.

10.2.3 The logic *PCTL*

In Section 10.4 (page 295 ff) we describe how the MTBDD-based algorithm to evaluate the expressions and terms of the algebraic mu-calculus (presented in Section 10.3, page 285 ff) can be applied to obtain symbolic verification methods for probabilistic systems. Thus, the method described in Section 10.4 focusses on a fixed data structure (namely MTBDDs) for representing probabilistic systems. Here, we explain how – from a purely mathematical point of view – *PCTL* formulas (with the interpretations over fully probabilistic and concurrent probabilistic systems) can be reformulated as boolean terms. Hence, *any* method that automatically evaluates the expressions and terms of the algebraic mu-calculus, can be used as basis for a model checker for *PCTL*, independent of the chosen data structure of an implementation.

Recall the syntax and semantics of *PCTL* that is explained in Chapter 9 (page 212 ff). In the concurrent case, we shrink our attention to stratified systems (see Definition 3.2.3, page 39) and consider the standard interpretation $\models$ à la [BidAl95] and the satisfaction relation $\models_{fair}$ that involves fairness with respect to the non-deterministic choices.

We now explain how any *PCTL* formula Φ can be translated into an "equivalent" boolean term $bterm_\Phi$. For this, we use the following function symbols. For each atomic proposition $a \in AP$, we use an 1-ary boolean function symbol Sat_a that represents the (characteristic function of the) set $Sat(a)$ of states where a holds. Moreover, we use a binary term variable P that stands for the transition probability matrix of a fully probabilistic system or a stratified system. In addition, in the concurrent (stratified) case, we assume an 1-ary boolean function symbol S_{prob} that stands for the (characteristic function of the) set of probabilistic states.

The boolean terms $bterm_\Phi$ are defined by structural induction. We define $bterm_{tt} = \lambda s[1]$, $bterm_a = Sat_a$ and, for formulas whose outermost operator is $\neg$ or $\wedge$,

$$bterm_{\neg\Phi} = \lambda s\,[\,\neg bterm_\Phi(s)\,], \quad bterm_{\Phi_1\wedge\Phi_2} = \lambda s\,[\,bterm_{\Phi_1}(s) \wedge bterm_{\Phi_2}(s)\,].$$

The definition of the boolean terms for formulas built from the probabilistic operator $\mathsf{Prob}_{\bowtie p}$ depends on whether we assume an interpretation over fully probabilistic or stratified systems. In what follows, we briefly write $lfp\ Z[\ldots]$ rather than $lim\ Z\ [\ldots \uparrow \lambda s[0]]$ and use expressions of the form `if` $bexpr$ `then` $expr_1$ `else` $expr_2$ instead of $bexpr * expr_1 + (1 - bexpr) * expr_2$.

Fully probabilistic case: For formulas whose outermost operator is the probabilistic operator $\mathsf{Prob}_{\bowtie p}$ we define $bterm_{\mathsf{Prob}_{\bowtie p}(\varphi)} = \lambda s[\,term_\varphi(s) \bowtie p\,]$ where the algebraic terms $term_\varphi$ for the path formulas are defined as follows.

$$term_{X\Phi} = \lambda s\,[\,\textstyle\sum_t\,[\,P(s,t) * bterm_\Phi(t)\,]\,]$$

$$term_{\Phi_1\mathcal{U}^{\leq k}\Phi_2} = iterate\ Z\ \Big[\ \lambda s[expr]\ \uparrow^k\ \lambda s[0]\ \Big]$$

$$term_{\Phi_1\mathcal{U}\Phi_2} = lfp\ Z\ [\ \lambda s[expr]\ ]$$

with

$$expr = \texttt{if}\ bterm_{\Phi_2}(s)\ \texttt{then}\ 1\ \texttt{else}\ bterm_{\Phi_1}(s) * \left(\sum_t\ [\ P(s,t) * Z(t)\ ]\right).$$

Let $\mathcal{S} = (S, \mathbf{P}, AP, \mathcal{L})$ be a proposition-labelled fully probabilistic system. We define $\mathcal{M} = (S, I)$ where $I(P) = \mathbf{P}$ and $I(Sat_a)(s) = 1$ if $a \in \mathcal{L}(s)$, $I(Sat_a)(s) = 0$ otherwise. Then, $[\![bterm_\Phi]\!]^{\mathcal{M}}$ is the characteristic function of $Sat(\Phi) = \{s \in S : s \models \Phi\}$ while $[\![term_\varphi]\!]^{\mathcal{M}}$ agrees with the function $S \to [0,1]$, $s \mapsto p_s(\Phi_1\mathcal{U}\Phi_2)$ where

$$p_s(\Phi_1\mathcal{U}\Phi_2) = Prob\{\pi \in Path_{ful}(s) : \pi \models \Phi_1\mathcal{U}\Phi_2\}.$$

This can be seen as follows. Theorem 3.1.6 (page 36) yields that the function $S \to [0,1]$, $s \mapsto p_s(\Phi_1\mathcal{U}\Phi_2)$, is the least fixed point of the operator $F : (S \to [0,1]) \to (S \to [0,1])$ which is given by: $F(f)(s) = 1$ if $s \models \Phi_2$, $F(f)(s) = 0$ if $s \not\models \Phi_1 \vee \Phi_2$ and $F(f)(s) = \sum_{t\in S_1\cup S_2} \mathbf{P}(s,t) \cdot f(t)$ if $s \models \Phi_1 \wedge \neg\Phi_2$. Using structural induction and

Theorem 10.1.27 (page 274) we obtain that, for all states s, $s \models \Phi$ iff $[\![bterm_\Phi]\!]^{\mathcal{M}}(s) = 1$ and $p_s(\varphi) = [\![term_\varphi]\!]^{\mathcal{M}}(s)$.[22] Thus,

$$term_{\Phi_1 \mathcal{U} \Phi_2} = lfp\ Z[\ \lambda s[expr]\ ] \text{ and } [\![\ term_{\Phi_1 \mathcal{U} \Phi_2}\ ]\!]^{\mathcal{M}}(s) = lfp(F)(s) = p_s(\Phi_1 \mathcal{U} \Phi_2).$$

Concurrent case: Depending on the comparison operator $\bowtie$ and depending on the satisfaction relation we need the minimal or maximal probabilities under all (or under all fair) adversaries. For both satisfaction relations $\models$ and $\models_{fair}$, the boolean terms for the formulas involving the probability operator $\mathsf{Prob}_{\sqsubset p}$ are defined as follows.

$$bterm_{\mathsf{Prob}_{\sqsubset p}(\varphi)} = \lambda s \left[\ term_\varphi^{max}(s) \sqsubset p\ \right]$$

where $\sqsubset \in \{<, \leq\}$. For the path formulas φ, $term_\varphi^{max}$ is defined as follows.

$$term_{X\Phi}^{max} = \lambda s [\ expr_{X\Phi}^{max}\ ] \text{ where } expr_{X\Phi}^{max} \text{ is}$$

$$\texttt{if } S_{prob}(s) \texttt{ then } \sum_t [\ P(s,t) * bterm_\Phi(t)\ ] \texttt{ else } \max_t [\ P(s,t) * bterm_\Phi(t)\ ]$$

$$term_{\Phi_1 \mathcal{U}^{\leq k} \Phi_2}^{max} = iterate\ Z \left[\ \lambda s \left[\ expr_{\Phi_1 \mathcal{U} \Phi_2}^{max}\ \right] \uparrow^k \lambda s[0]\ \right],$$

$$term_{\Phi_1 \mathcal{U} \Phi_2}^{max} = lfp\ Z \left[\ \lambda s \left[\ expr_{\Phi_1 \mathcal{U} \Phi_2}^{max}\ \right]\ \right]$$

The expression $expr_{\Phi_1 \mathcal{U} \Phi_2}^{max}$ is defined as follows.

$$expr_{\Phi_1 \mathcal{U} \Phi_2}^{max} = \texttt{if } bterm_{\Phi_2}(s) \texttt{then } 1 \texttt{ else } bterm_{\Phi_1}(s) * expr'$$

where

$$expr' = \texttt{if } S_{prob}(s) \texttt{ then } \sum_t [\ P(s,t) * Z(t)\ ] \texttt{ else } \max_t [\ P(s,t) * Z(t)\ ].$$

Now we consider formulas of the form $\mathsf{Prob}_{\sqsupset p}(\varphi)$ where $\sqsupset \in \{>, \geq\}$. If the outermost operator of φ is the next step operator or the bounded until operator then the definition of the corresponding boolean term is the same for the interpretations $\models$ and $\models_{fair}$. For $\varphi \in \{X\Phi, \Phi_1 \mathcal{U}^{\leq k} \Phi_2\}$, we define $bterm_{\mathsf{Prob}_{\sqsupset p}(\varphi)} = \lambda s \left[\ term_\varphi^{min}(s) \sqsupset p\ \right]$ where the definition of the algebraic terms $term_\varphi^{min}$ is similar to the definition of $term_\varphi^{max}$.

$$term_{X\Phi}^{min} = \lambda s [\ expr_{X\Phi}^{min}\ ] \text{ where } expr_{X\Phi}^{min} \text{ is}$$

$$\texttt{if } S_{prob}(s) \texttt{ then } \sum_t [\ P(s,t) * bterm_\Phi(t)\ ] \texttt{ else } \min_t [\ P(s,t) * bterm_\Phi(t)\ ]$$

$$term_{\Phi_1 \mathcal{U}^{\leq k} \Phi_2}^{min} = iterate\ Z \left[\ \lambda s \left[\ expr_{\Phi_1 \mathcal{U} \Phi_2}^{min}\ \right] \uparrow^k \lambda s[0]\ \right]$$

with $expr_{\Phi_1 \mathcal{U} \Phi_2}^{min} = \texttt{if } bterm_{\Phi_2}(s) \texttt{then } 1 \texttt{ else } bterm_{\Phi_1}(s) * expr''$ where $expr''$ is

$$\texttt{if } S_{prob}(s) \texttt{ then } \sum_t [\ P(s,t) * Z(t)\ ] \texttt{ else } \min_t [\ P(s,t) * Z(t)\ ].$$

[22] Note that – with $P \in WFct^2(2)$ – the terms $term_\varphi$ are terms of the algebraic $[0,1]$-mu-calculus that are formally divergence free; the term variable Z is formally monotone and formally continuous in $\lambda s[expr]$.

Next we consider the until operator, i.e. formulas of the form $\mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U} \Phi_2)$. We define

$$bterm_{\mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U} \Phi_2)} \;=\; \lambda s \Big[\; term^{min}_{\Phi_1 \mathcal{U} \Phi_2}(s) \;\sqsupseteq\; p \;\Big]$$

where the definition of $term^{min}_{\Phi_1 \mathcal{U} \Phi_2}$ depends on whether we deal with $\models$ or $\models_{fair}$. Dealing with the standard interpretation $\models$, we define

$$term^{min}_{\Phi_1 \mathcal{U} \Phi_2} \;=\; lfp\ Z \;\Big[\; \lambda s \Big[\; expr^{min}_{\Phi_1 \mathcal{U} \Phi_2} \;\Big] \;\Big]$$

where $expr^{min}_{\Phi_1 \mathcal{U} \Phi_2}$ is as in the case of the bounded until operator. Dealing with the satisfaction relation $\models_{fair}$ we use the result of Theorem 9.3.23 (page 227) stating that

$$s \;\models_{fair}\; \mathsf{Prob}_{\sqsupseteq p}(\Phi_1 \mathcal{U} \Phi_2) \quad \text{iff} \quad 1 - p^{max}_s(a^? \mathcal{U} \neg a^+) \sqsupseteq p$$

where a^+ and $a^?$ are fresh atomic propositions representing the sets $S^+(\Phi_1, \Phi_2)$ and $S^?(\Phi_1, \Phi_2) = Sat(\Phi_1) \setminus S^+(\Phi_1, \Phi_2)$. Recall that $S^+(\Phi_1, \Phi_2)$ is the set of all states $s \in S$ that can reach a Φ_2-state via a path through Φ_1-states (Notation 9.3.11, page 223). Thus, $S^+(\Phi_1, \Phi_2) \;=\; lfp(F)$ where the monotonic operator $F : 2^S \to 2^S$ is given by

$$F(Z) \;=\; Sat(\Phi_2) \;\cup\; \{ s \in Sat(\Phi_1) \;:\; \exists t \in S \;[\; (\mathbf{P}(s,t) > 0) \;\wedge\; t \in Z \;] \}\,.$$

The definition of the term $term^{min}_{\Phi_1 \mathcal{U} \Phi_2}$ with respect to the satisfaction relation $\models_{fair}$ is as follows. We put

$$term^{min}_{\Phi_1 \mathcal{U} \Phi_2} \;=\; \lambda s \;[\; 1 - term^{max}_{a^? \mathcal{U} \neg a^+}(s) \;]\,.$$

The algebraic term $term^{max}_{a^? \mathcal{U} \neg a^+}$ is defined as described above with the only difference that we deal with the boolean terms $bterm^? \;=\; \lambda s \Big[bterm_{\Phi_1}(s) \wedge \neg bterm^+(s) \Big]$ and $bterm^+$ rather than the function symbols $Sat_{a^?}$ and Sat_{a^+}. Here, $bterm^+$ is given by

$$lfp\ Z \;[\; \lambda s \;[\; bterm_{\Phi_2}(s) \;\vee\; (\; bterm_{\Phi_1}(s) \;\wedge\; \exists t \;[\; (P(s,t) > 0) \;\wedge\; Z(t) \;]\;)\;]\;]\,.$$

As in the fully probabilistic case (and using Theorem 10.1.27 (page 274), Theorem 3.2.11 (page 43) and the results of Section 9.3) we obtain the following. Let $\mathcal{S} = (S, \mathbf{P}, AP, \mathcal{L})$ be a finite proposition-labelled stratified system. Let $\mathcal{M} = (S, I)$ where $I(Sat_a)$ and $I(S_{prob})$ are the boolean-valued functions $S \to \{0,1\}$ with $I(Sat_a)(s) = 1$ iff $a \in \mathcal{L}(s)$ and $I(S_{prob})(s) = 1$ iff s is a probabilistic state. The interpretation $I(P)$ for the binary function symbol P is given the transition probability function $\mathbf{P}$ (defined as in Notation 3.2.4, page 40); more precisely, we deal with

$$I(P)(s,t) \;=\; \begin{cases} \mathbf{P}(s,t) & : \text{ if } s \in S_{prob} \\ 1 & : \text{ if } s \notin S_{prob} \text{ and } s \longrightarrow t \\ \bot & : \text{ otherwise.} \end{cases}$$

Then, for all states $s \in S$, $s \models_{\mathcal{A}} \Phi$ iff $[\![bterm_\Phi]\!]^{\mathcal{M}}(s) = 1$ and

$$[\![term^{min}_{\varphi}]\!]^{\mathcal{M}}(s) \;=\; \inf_{A \in \mathcal{A}} \; Prob\Big\{ \pi \in Path^A_{ful}(s) : \pi \models_{\mathcal{A}} \varphi \Big\}\,,$$

$$[\![term^{max}_{\varphi}]\!]^{\mathcal{M}}(s) \;=\; \sup_{A \in \mathcal{A}} \; Prob\Big\{ \pi \in Path^A_{ful}(s) : \pi \models_{\mathcal{A}} \varphi \Big\}$$

where $\mathcal{A}$ stands for $\mathcal{A}dv$ or $\mathcal{A}dv_{fair}$ (depending on whether we deal with $\models$ or $\models_{fair}$).[23] In a similar way, we can deal with the satisfaction relations $\models_{sfair}$ (where strict fairness is supposed) or $\models^{W}_{fair}$ (where fairness in the W-states is supposed). For the definition of the boolean terms $bterm_{\mathsf{Prob}_{\bowtie p}(\Phi_1 \mathcal{U} \Phi_2)}$ for formulas involving the unbounded until operator, one has to describe the set $T^{max}(\Phi_1, \Phi_2)$ (see Notation 9.3.14, page 224) resp. the set S^0_W (see Notation 9.3.31, page 229) by a boolean term. In both cases, the corresponding boolean term is of the form $lfp\ Z[bterm]$. For instance, for the set $T^{max}(\Phi_1, \Phi_2)$, the definition of $bterm$ is given by

$$\lambda s \, [\ \neg bterm^{+}(s) \ \vee \ bterm_{\Phi_2}(s) \ \vee \ (\ S_{prob}(s) \ \wedge \ \forall t \, [\, (P(s,t) > 0) \rightarrow Z(t) \,] \ )$$

$$\vee \ (\ \neg S_{prob}(s) \ \wedge \ \exists t \, [\, (P(s,t) > 0) \ \wedge \ (term^{max}_{\Phi_1 \mathcal{U} \Phi_2}(s) = term^{max}_{\Phi_1 \mathcal{U} \Phi_2}(t)) \,] \ ) \ \ ].$$

10.2.4 Word level *CTL*

Word Level *CTL* [CCH+96, CKZ96, Zhao96] is an extension of *CTL* to reason about properties involving the relationships among data words. Such properties are needed for the verification of *arithmetic circuits*. Word Level *CTL* distinguishes between several types of formulas: *atomic formulas* Φ^{AF} that are built from atomic propositions and equations or inequalities for expressions, *static formulas* Φ^{SF} that are built from atomic formulas and the boolean connectives $\wedge$ and $\neg$ and *temporal formulas* Φ^{TF} that are given by static formulas, the boolean connectives and the *CTL* path quantifiers combined with the temporal operators X and $\mathcal{U}$.

$$\begin{array}{lcl}
\Phi^{AF} & ::= & a \ \mid \ \forall(e_1 \bowtie e_2) \ \mid \ \exists(e_1 \bowtie e_2) \\
\Phi^{SF} & ::= & \Phi^{AF} \ \mid \ \Phi^{SF}_1 \wedge \Phi^{SF}_2 \ \mid \ \neg \Phi^{SF} \\
\Phi^{TF} & ::= & \Phi^{SF} \ \mid \ \Phi^{TF}_1 \wedge \Phi^{TF}_2 \ \mid \ \neg \Phi^{TF} \ \mid \ \forall X \Phi^{TF} \ \mid \ \exists(\Phi^{TF}_1 \mathcal{U} \Phi^{TF}_2) \ \mid \ \exists \Box \Phi^{TF}
\end{array}$$

where $a \in AP$ and $\bowtie \in \{=, <, \leq, >, \geq\}$. The words are tuples of propositional formulas, i.e. they are of the form $word = \langle \Phi^{PF}_1, \ldots, \Phi^{PF}_n \rangle$ where the propositional formulas Φ^{PF} are built from the atomic propositions $a \in AP$ and the boolen connectives $\wedge$ and $\neg$. The expressions are given by:

$$e \ ::= \ const \ \mid \ word \ \mid \ next(word) \ \mid \ e_1 \ op \ e_2 \ \mid \ \texttt{if} \ \Phi^{SF} \ \texttt{then} \ e_1 \ \texttt{else} \ e_2$$

Formulas, expressions and words are interpreted over finite proposition-labelled transition systems $(S, \mathbf{R}, AP, \mathcal{L})$ where S is a set of states, $\mathbf{R} \subseteq S \times S$ the transition relation and $\mathcal{L} : S \rightarrow 2^{AP}$ the labelling function for the states by atomic propositions. Propositional, atomic, static and temporal formulas are interpreted by sets of the states:

$$[\![a]\!] = \{s \in S : a \in \mathcal{L}(s)\}, \quad [\![\Phi_1 \wedge \Phi_2]\!] = [\![\Phi_1]\!] \cap [\![\Phi_2]\!], \quad [\![\neg \Phi]\!] = S \setminus [\![\Phi]\!],$$

$$[\![\ \forall(e_1 \bowtie e_2) \ ]\!] \ = \ \{ \ s \in S \ : \ \forall s' \in S \, [\ \mathbf{R}(s,s') \ \rightarrow \ [\![e_1]\!](s,s') \bowtie [\![e_2]\!](s,s') \ ] \ \},$$

$$[\![\ \exists(e_1 \bowtie e_2) \ ]\!] \ = \ \{ \ s \in S \ : \ \exists s' \in S \, [\ \mathbf{R}(s,s') \ \rightarrow \ [\![e_1]\!](s,s') \bowtie [\![e_2]\!](s,s') \ ] \ \},$$

[23] Here, for the extension of multiplication *, the minimum/maximum operators and the comparison operators $op_{\bowtie}$ on the extended reals, we assume that, if $q \in \mathbb{R}$ and $\emptyset \neq Q \subseteq \mathbb{R}$ then $q * \bot = \bot * q = \bot$, $q \, op_{\bowtie} \bot = \bot \, op_{\bowtie} q = 1$, $\min Q = \min(Q \setminus \{\tau\})$, $\max Q = \max(Q \setminus \{\tau\})$ and $\min\{\bot\} = \max\{\bot\} = \bot$.

$$[\![\ \forall X \Phi_{TF}\ ]\!] \ = \ \{\ s \in S\ :\ \forall s' \in S\ [\ \mathbf{R}(s,s') \rightarrow s' \in [\![\Phi_{TF}]\!]\ ]\ \}$$

$$[\![\ \exists(\Phi_1^{TF} \mathcal{U} \Phi_2^{TF})\ ]\!] \ = \ \{\ s \in S\ :\ \exists k \geq 0\ \exists s_0, \ldots, s_k \in S\ [\ (s_0 = s)\,, \\ \wedge\ \left(s_k \in [\![\Phi_2^{TF}]\!]\right)\ \wedge\ \bigwedge_{0 \leq i < k}\ \left(\mathbf{R}(s_i, s_{i+1}) \wedge (s_i \in [\![\Phi_1^{TF}]\!])\right)\ ]\ \},$$

$$[\![\ \exists \Box \Phi^{TF}\ ]\!] \ = \ \{\ s \in S\ :\ \exists s_0, s_1, s_2 \ldots \in S\ [\ (s_0 = s) \\ \wedge\ \ \bigwedge_{i \geq 0}(\mathbf{R}(s_i, s_{i+1}) \wedge (s_i \in [\![\Phi^{TF}]\!])\ \ ]\ \}.$$

The interpretation $[\![word]\!] : S \rightarrow I\!N$ of a word is given by:

$$[\![\ \langle \Phi_1^{PF}, \ldots, \Phi_n^{PF} \rangle\ ]\!](s) \ = \ \sum_{i=0}^{n} [\![\Phi_i^{PF}]\!](s) \cdot 2^i$$

where the set $[\![\Phi_i^{PF}]\!]$ is identified with their characteristic function $S \rightarrow \{0,1\}$. The interpretation $[\![e]\!] : S \times S \rightarrow I\!N$ of an expression e is given by:

$$[\![const]\!](s,s') = const, \quad [\![e_1\ op\ e_2]\!](s,s') \ = \ [\![e_1]\!](s,s')\ op\ [\![e_2]\!](s,s'),$$
$$[\![word]\!](s,s') = [\![word]\!](s), \quad [\![next(word)]\!](s,s') = [\![word]\!](s')$$

and

$$[\![\texttt{if}\ \Phi^{SF}\ \texttt{then}\ e_1\ \texttt{else}\ e_2]\!](s,s') \ = \ \begin{cases} [\![e_1]\!](s,s') & : \text{ if } s \in [\![\Phi^{SF}]\!] \\ [\![e_2]\!](s,s') & : \text{ otherwise.} \end{cases}$$

We associate with each formula Φ of the word level CTL an 1-ary boolean term $bterm_\Phi$ while the words and the expressions are associated with algebraic terms $term_*^W$ and $term_*^E$. The terms for the expressions are binary while the terms for the words have the arity 1. We use a binary function symbol R (that represents the transition relation $\mathbf{R}$), 1-ary term variables Sat_a for all atomic propositions a (that represent the sets $[\![a]\!]$) and individual variables s and s'. The 1-ary algebraic terms for the words are given by:

$$term^W_{\langle \Phi_1^{PF}, \ldots, \Phi_n^{PF} \rangle} \ = \ \lambda s \left[\sum_{1 \leq i \leq n} bterm_{\Phi_i^{PF}}(s) * 2^i \right]$$

The binary terms for the expressions are given by $term^E_{const} = \lambda s, s'[const]$ and

$$term^E_{word} = \lambda s, s'[term^W_{word}(s)], \quad term^E_{next(word)} = \lambda s, s' \left[\ term^W_{word}(s')\ \right]$$
$$term^E_{e_1\ op\ e_2} \ = \ \lambda s, s' \left[\ term^E_{e_1}(s,s')\ op\ term^E_{e_2}(s,s')\ \right]$$
$$term^E_{\texttt{if}\ \Phi^{SF}\ \texttt{then}\ e_1\ \texttt{else}\ e_2}$$
$$= \ \lambda s, s' \ \left[\ \texttt{if}\ bterm_{\Phi^{SF}}(s)\ \texttt{then}\ term^E_{e_1}(s,s')\ \texttt{else}\ term^E_{e_2}(s,s')\ \right]$$

The boolean terms for formulas built from the boolean connectives are clear:

$$bterm_a = Sat_a, \ \ bterm_{\neg\Phi} = \lambda s[\neg bterm_\Phi(s)], \ \ bterm_{\Phi_1 \wedge \Phi_2} = \lambda s[bterm_{\Phi_1}(s) \wedge bterm_{\Phi_2}(s)]$$

For the atomic formulas we define:

$$bterm_{\forall(e_1 \bowtie e_2)} \ = \ \lambda s \left[\ \forall s' \left[\ R(s,s') \ \rightarrow \ \left(\ term^E_{e_1}(s,s') \ \bowtie \ term^E_{e_2}(s,s')\ \right)\ \right]\ \right],$$
$$bterm_{\exists(e_1 \bowtie e_2)} \ = \ \lambda s \left[\ \exists s' \left[\ R(s,s') \ \rightarrow \ \left(\ term^E_{e_1}(s,s') \ \bowtie \ term^E_{e_2}(s,s')\ \right)\ \right]\ \right].$$

For the temporal operators we use the following boolean terms:

$$bterm_{\forall X \Phi^{TF}} \;=\; \lambda s\,[\;\forall s'\;[\;R(s,s')\;\rightarrow\; bterm_{\Phi^{TF}}(s')\;]\;]$$

$$bterm_{\exists(\Phi_1^{TF}\mathcal{U}\Phi_2^{TF})}$$

$$= \; lfp\; Z\;\Big[\;\lambda s\Big[\;bterm_{\Phi_2^{TF}}(s)\;\vee\;\Big(\;bterm_{\Phi_1^{TF}}(s)\wedge\exists s'\,[R(s,s')\wedge Z(s')]\;\Big)\;\Big]\;\Big]$$

$$bterm_{\exists\Box\Phi^{TF}} \;=\; lfp\; Z\;[\;\lambda s[\;bterm_{\Phi^{TF}}(s)\;\wedge\;(\;\exists s'\,[R(s,s')\wedge Z(s')]\;)\;]\;]$$

The connection between the word level *CTL* formulas and the associated algebraic terms is as follows. Let $\mathcal{M} = (S, I)$ where $I(R) = \mathbf{R}$ and

$$I(Sat_a)(s) \;=\; \begin{cases} 1 & : \text{ if } a \in \mathcal{L}(s) \\ 0 & : \text{ otherwise.} \end{cases}$$

Using structural induction and Theorem 10.2.1 (page 276) we obtain: $[\![term^W_{word}]\!]^{\mathcal{M}} = [\![word]\!]$, $[\![term^E_e]\!]^{\mathcal{M}} = [\![e]\!]$ and

$$[\![bterm_\Phi]\!]^{\mathcal{M}}(s) \;=\; \begin{cases} 1 & : \text{ if } s \in [\![\Phi]\!] \\ 0 & : \text{ otherwise} \end{cases}$$

for all formulas Φ, words *word* and expressions e.

10.3 A "compiler" for the algebraic mu-calculus

The algebraic mu-calculus can be viewed as a *language* for manipulating real-valued functions. Any closed algebraic term *term* yields an operator $\psi(f_1, \ldots, f_k)$ that takes the interpretations $f_i = I(fct_i)$ for the function symbols as its input and describes how to combine these functions $f_1, \ldots, f_k$ via arithmetic operators and iteration. The semantics $[\![term]\!]^{\mathcal{M}}$ (where $\mathcal{M} = (D, I)$) stands for the composed function. For instance, in the example for solving linear equation systems of the type $\mathbf{z} = \mathbf{q} + \mathbf{A} \cdot \mathbf{z}$ with the "naive" method (Example 10.1.9, page 266), the term

$$term \;=\; lim\; Z\;\left[\;\lambda i\left[\;q(i)\;+\;\sum_j\;[\;A(i,j) * Z(j)\;]\;\right]\;\uparrow\;z_0\;\right]$$

can be viewed as the operator that takes as its arguments the functions $I(A)$ for the matrix $\mathbf{A}$, $I(q)$ for the vector $\mathbf{q}$ and $I(z_0)$ for the starting vector $\mathbf{z}_0$ and "returns" the function that represents the unique solution $\mathbf{z}$.

In this section, we turn to the question how the terms (and expressions) can be evaluated automatically and present an algorithm that takes as its input an algebraic term (or expression) and a model $\mathcal{M} = (D, I)$ and returns the semantics $[\![\ldots]\!]^{\mathcal{M}}$ of that term (or expression) with respect to $\mathcal{M}$. In some sense, this algorithm can be viewed as a *compiler* for the algebraic mu-calculus. Such an algorithm requires an adequate data structure for the functions $D^n \rightarrow \overline{I\!R}$. Of course, "adequacy" of the chosen data structure depends on the concrete application. In that thesis where we concentrate on the verification of probabilistic systems we shrink our attention to the use of MTBDDs as chosen data structure since MTBDDs are known to be efficient for representing probabilistic systems

[HMP+94, HarG98].[24] Clearly, in other applications, the use of MTBDDs might be not efficient. For instance, in Section 10.2.4 (page 283 ff), we saw that the algebraic mu-calculus can also serve as specification language for arithmetic circuits. In that case, it is known that the use of MTBDDs is not efficient (the resulting MTBDDs might have exponential size) and the use of other decision diagrams like HDDs is preferable. See [CCH+96, CKZ96, Zhao96].

In a first step, we introduce the *mixed calculus* which is a variant of the algebraic mu-calculus that is based on a fixed interpretation for the function symbols. For this, we assume the domain $D = \{0,1\}$ of the underlying model $\mathcal{M} = (D, I)$ and a representation of the functions $I(fct) : \{0,1\}^n \to \overline{\mathbb{R}}$ for the n-ary function symbols by MTBDDs. The expressions and terms of the mixed calculus are interpreted by partial functions from the individual variables into the reals. These functions are represented by MTBDDs whose nonterminal vertices are labelled by individual variables.[25] For the mixed calculus, we describe an algorithm that takes a term or expression of the mixed calculus as its input and generates the corresponding MTBDD. Given a model $\mathcal{M} = (D, I)$ for the algebraic mu-calculus, we use an encoding of the domain D in $\{0,1\}^k$ and transform the algebraic expressions and terms into "equivalent" expressions and terms of the mixed calculus. The MTBDD for that expression or term of the mixed calculus can be viewed as a representation for the semantics $[\![\ldots]\!]^{\mathcal{M}}$ with respect to $\mathcal{M}$.[26] Thus, the transformation algorithm from the algebraic to the mixed calculus and the algorithm for computing the MTBDDs for the expressions and terms of the mixed calculus can be composed to a method that automatically computes the semantics of the algebraic expressions and terms. On the other hand, the mixed calculus in its own can be viewed as a *language* for manipulating MTBDDs where our algorithm acts as a *compiler* that automatically generates the MTBDD described by an expression or term of the mixed calculus.

10.3.1 The mixed calculus

We present the syntax and semantics of the *mixed calculus*. In essential, the syntax of the mixed calculus arises from the syntax of the algebraic mu-calculus where the function symbols are replaced by MTBDDs. The expressions of the mixed calculus are interpreted by partial functions from the individual variables into the reals; the semantics of the n-ary terms are partial functions that take as their arguments the individual variables and an n-bit vector.

Syntax of the mixed calculus: We fix sets IndVar of individual variables, TermVar of term variables where each term variable Z is associated with an arity (a natural number ≥ 1) and a set $\{\vartheta_1, \vartheta_2, \ldots\}$ of *dummy variables*. As before, *Op* denotes a set of total binary operators on the extended reals. The syntax of *mixed* expressions and n-ary terms is given by the production system shown in Figure 10.7 on page 287. Here, $q \in \overline{\mathbb{R}}$ and

[24] The reader not familiar with MTBDDs should recall the definition of MTBDDs which is presented in Section 12.3 (page 315 ff).

[25] To be precisely, the n-ary terms also take a n-bit vector as input. Thus, the MTBDDs for them also contain nonterminal vertices labelled by other variables.

[26] For this, we surpress the interpretation $I(z)$ for the individual variables of the algebraic mu-calculus and consider $[\![expr]\!]^{\mathcal{M}}$ as a function $(IndVar \to D) \to \overline{\mathbb{R}}$ and $[\![term]\!]^{\mathcal{M}}$ as a function $(IndVar \to D) \times \{0,1\}^n \to \overline{\mathbb{R}}$.

$$
\begin{array}{rcl}
\mathsf{expr} & ::= & q \mid \mathsf{z} \mid \mathsf{expr}_1\ op\ \mathsf{expr}_2 \mid \mathsf{term}(\mathsf{z}_1, \ldots, \mathsf{z}_n) \mid \\
 & & \sum_{\mathsf{z}} [\mathsf{expr}] \mid \min_{\mathsf{z}} [\mathsf{expr}] \mid \max_{\mathsf{z}} [\mathsf{expr}] \\
\mathsf{term} & ::= & \mathrm{Q} \mid \mathsf{Z} \mid \lambda \mathsf{z}_1, \ldots, \mathsf{z}_n\ [\mathsf{expr}] \mid lim\ \mathsf{Z}\ [\ \mathsf{term} \uparrow \mathsf{term}_0\ ] \\
 & & iterate\ \mathsf{Z}\ [\ \mathsf{term} \uparrow^k \mathsf{term}_0\ ]
\end{array}
$$

Figure 10.7: Syntax of the mixed calculus

$op \in Op$. Q is a MTBDD over $(\vartheta_1, \ldots, \vartheta_n)$. $\mathsf{z}, \mathsf{z}_1, \ldots, \mathsf{z}_n \in \mathsf{IndVar}$ such that $\mathsf{z}_1, \ldots, \mathsf{z}_n$ are pairwise distinct. $\mathsf{Z} \in \mathsf{TermVar}$ is an n-ary term variable. Free and bounded occurrences of individual or term variables in mixed expressions or terms are defined in the obvious way. For the expressions $\mathsf{term}(\mathsf{z}_1, \ldots, \mathsf{z}_n)$, we require that there are no free occurrences of the individual variables z_i in term. A mixed expression or term is called *closed* iff it does not contain free occurrences of individual or term variables. For $\overline{\mathsf{z}} = (\mathsf{z}_1, \ldots, \mathsf{z}_n)$, we briefly write $\sum_{\overline{\mathsf{z}}}$ or $\sum_{\mathsf{z}_1,\ldots,\mathsf{z}_n}$ rather than $\sum_{\mathsf{z}_1} \ldots \sum_{\mathsf{z}_n}$. Similarly, $\min_{\overline{\mathsf{z}}}$, $\max_{\overline{\mathsf{z}}}$ or $\lambda\overline{\mathsf{z}}$ have the obvious meanings. The mixed boolean calculus is defined in analogy to the boolean mu-calculus (see page 261).

Semantics of the mixed calculus: Intuitively, the mixed expressions and terms are interpreted by functions with values in the extended reals and whose arguments are the individual variables. Moreover, the functions for the n-ary mixed terms depend on an n-bit vector (that represents the values of the dummy variables $\vartheta_1, \ldots, \vartheta_n$). Formally, the semantics of the mixed calculus is defined with respect to an interpretation $\mathcal{J}$ for the n-ary term variables by functions $\{0,1\}^n \to \overline{I\!R}$. The semantics

$$[\![\mathsf{expr}]\!]^{\mathcal{J}} : (\mathsf{IndVar} \to \{0,1\}) \to \overline{I\!R}, \quad [\![\mathsf{term}]\!]^{\mathcal{J}} : (\mathsf{IndVar} \to \{0,1\}) \times \{0,1\}^n \to \overline{I\!R}$$

of the mixed expressions and terms with respect to $\mathcal{J}$ is defined by structural induction as shown in Figure 10.8 (page 288). Here, ι is a function $\mathsf{IndVar} \to \{0,1\}$ and $\langle b_1, \ldots, b_n \rangle \in \{0,1\}^n$. $\iota[\mathsf{z}_1 := c_1, \ldots, \mathsf{z}_k := c_k]$ denotes those function $\mathsf{IndVar} \to \{0,1\}$ that agrees with ι on all individual variables $\mathsf{z} \in \mathsf{IndVar} \setminus \{\mathsf{z}_1, \ldots, \mathsf{z}_k\}$ and returns the value c_i for the variable z_i.[27] The interpretation $\mathcal{J}[\mathsf{Z} := f]$ is defined in the obvious way.

10.3.2 Inference from the algebraic to the mixed calculus

Given an expression or term of the algebraic mu-calculus and a model $\mathcal{M} = (D, I)$ for the algebraic mu-calculus, we define an "equivalent" mixed expression or term. This inference from the algebraic mu-calculus to the mixed calculus is based on an encoding of the elements of D by k-bit vectors. The individual variables z of the algebraic mu-calculus are replaced by k-tuples $(\mathsf{z}_1^z, \ldots, \mathsf{z}_k^z)$ of individual variables of the mixed calculus.

[27] Here, we assume that $\mathsf{z}_1, \ldots, \mathsf{z}_k \in \mathsf{IndVar}$ are pairwise distinct and that $c_1, \ldots, c_k \in \{0,1\}$.

$$[\![q]\!]^{\mathcal{J}}(\iota) = q \qquad [\![\mathsf{z}]\!]^{\mathcal{J}}(\iota) = \iota(\mathsf{z})$$

$$[\![\ \mathsf{expr}_1\ op\ \mathsf{expr}_2\ ]\!]^{\mathcal{J}}(\iota) \ = \ [\![\mathsf{expr}_1]\!]^{\mathcal{J}}(\iota)\ op\ [\![\mathsf{expr}_2]\!]^{\mathcal{J}}(\iota)$$

$$[\![\ \mathsf{term}(\mathsf{z}_1,\ldots,\mathsf{z}_n)\ ]\!]^{\mathcal{J}}(\iota) \ = \ [\![\mathsf{term}]\!]^{\mathcal{J}}\ (\ \iota,\ \langle \iota(\mathsf{z}_1),\ldots,\iota(\mathsf{z}_n)\rangle\ )$$

$$[\![\ \textstyle\sum_{\mathsf{z}}[\mathsf{expr}]\ ]\!]^{\mathcal{J}}(\iota) \ = \ [\![\mathsf{expr}]\!]^{\mathcal{J}}(\ \iota[\mathsf{z} := 0]\ ) \ + \ [\![\mathsf{expr}]\!]^{\mathcal{J}}(\ \iota[\mathsf{z} := 1]\ )$$

$$[\![\ \min_{\mathsf{z}}[\mathsf{expr}]\ ]\!]^{\mathcal{J}}(\iota) \ = \ \min\ \left\{\ [\![\mathsf{expr}]\!]^{\mathcal{J}}(\ \iota[\mathsf{z} := 0]\ ),\ [\![\mathsf{expr}]\!]^{\mathcal{J}}(\ \iota[\mathsf{z} := 1]\ )\ \right\}$$

$$[\![\ \max_{\mathsf{z}}[\mathsf{expr}]\ ]\!]^{\mathcal{J}}(\iota) \ = \ \max\ \left\{\ [\![\mathsf{expr}]\!]^{\mathcal{J}}(\ \iota[\mathsf{z} := 0]\ ),\ [\![\mathsf{expr}]\!]^{\mathcal{J}}(\ \iota[\mathsf{z} := 1]\ )\ \right\}$$

$$[\![\mathrm{Q}]\!]^{\mathcal{J}}\ (\ \iota,\ \langle b_1,\ldots,b_n\rangle\ ) \ = \ f_{\mathrm{Q}}(b_1,\ldots,b_n) \qquad [\![\mathsf{Z}]\!]^{\mathcal{J}} \ = \ \mathcal{J}(\mathsf{Z})$$

$$[\![\ \lambda\mathsf{z}_1,\ldots,\mathsf{z}_n[\mathsf{expr}]\ ]\!]^{\mathcal{J}}(\iota,\langle b_1,\ldots,b_n\rangle) \ = \ [\![\mathsf{expr}]\!]^{\mathcal{J}}\ (\ \iota[\mathsf{z}_1 := b_1,\ldots,\mathsf{z}_n := b_n]\ )$$

$$[\![\ \mathit{lim}\ \mathsf{Z}\ [\ \mathsf{term} \uparrow \mathsf{term}_0\ ]\ ]\!]^{\mathcal{J}} \ = \ \mathit{lim}(f_0, f_1, f_2, \ldots)$$

$$[\![\ \mathit{iterate}\ \mathsf{Z}\ [\ \mathsf{term} \uparrow^k \mathsf{term}_0\ ]\ ]\!]^{\mathcal{J}} \ = \ f_k$$

$$\text{where } f_0 = [\![\mathsf{term}_0]\!]^{\mathcal{J}},\ f_{i+1} = [\![\mathsf{term}]\!]^{\mathcal{J}[\mathsf{Z}:=f_i]}.$$

Figure 10.8: Semantics of the mixed calculus

While the individual variable z of the algebraic mu-calculus is interpreted by an element $I(z)$ of the domain D, the individual variable z_i^z of the mixed calculus stands for the i-th component of the bit vector that encodes $I(z)$. Moreover, we represent the interpretations $I(fct) : D^n \to \overline{I\!R}$ for the n-ary function symbols by functions $\{0,1\}^{n\cdot k} \to \overline{I\!R}$ and replace fct by the corresponding MTBDD. Thus, n-ary algebraic terms are translated into $(n \cdot k)$-ary mixed terms.

We fix a model $\mathcal{M} = (D, I)$ for the algebraic mu-calculus and choose an encoding of D in $\{0,1\}^k$, i.e. an injection $code : D \to \{0,1\}^k$ (where $k = \lceil \log |D| \rceil$). For each n-ary function symbol fct, we assume a representation of the function $I(fct) : D^n \to \overline{I\!R}$ by a function $\widehat{I(fct)} : \{0,1\}^{n\cdot k} \to \overline{I\!R}$. For instance, we may put

$$\widehat{I(fct)}(code(d_1),\ldots,code(d_n)) \ = \ I(fct)(d_1,\ldots,d_n)$$

for all $d_1,\ldots,d_n \in D$. If $\overline{b}_i \in \{0,1\}^k$, $i = 1,\ldots,n$, such that at least one k-bit tupe $\overline{b}_i$ is not of the form $code(d)$ for some $d \in D$ then we put $\widehat{I(fct)}(\overline{b}_1,\ldots,\overline{b}_n) = \bot$.[28] If fct is an n-ary function symbol in the algebraic mu-calculus then we associate with fct those MTBDD Q_{fct} over $(\vartheta_1,\ldots,\vartheta_{n\cdot k})$ where the induced function $f_{\mathrm{Q}_{fct}}$ is $\widehat{I(fct)}$. Let $IndVar$ be the individual variables used in the algebraic mu-calculus. Then, in the mixed calculus

[28] Note that also other representations of $I(fct)$ by a function $\{0,1\}^{n\cdot k} \to \overline{I\!R}$ are possible. E.g. if $n = 2$ then $\widehat{I(fct)}$ might be defined by $\widehat{I(fct)}(b_1, c_1, \ldots, b_k, c_k) = f(d_1, d_2)$ where $\langle b_1,\ldots,b_k\rangle = code(d_1)$ and $\langle c_1,\ldots,c_k\rangle = code(d_2)$.

we use the individual variables

$$\mathsf{IndVar} \;=\; \{\mathsf{z}_i^z : z \in \mathit{IndVar}, i = 1, \ldots, k\}.$$

Each n-ary term variable Z of the algebraic mu-calculus is viewed as $(n \cdot k)$-ary term variable of the mixed calculus. I.e., in the mixed calculus, we deal with set $\mathsf{TermVar} = \mathit{TermVar}$ of term variables where the arity of each term variable of the algebraic mu-calculus is multiplied by the factor k. For each algebraic expression *expr* (or term *term*), let $\mathsf{mixed}(\mathit{expr})$ (resp. $\mathsf{mixed}(\mathit{term})$) be those mixed expression (or term) that results from *expr* (or *term*) by replacing

- each individual variable $z \in \mathit{IndVar}$ by the individual variables $\mathsf{z}_1^z, \ldots, \mathsf{z}_k^z$,[29]
- each function symbol *fct* by the MTBDD $\mathrm{Q}_{\mathit{fct}}$.

We get the "equivalence" of the algebraic expressions/terms and the resulting mixed expressions/terms in the following sense. Let $\iota : \mathsf{IndVar} \to \{0,1\}$ be given by

$$\iota(\mathsf{z}_i^z) \;=\; i\text{-th component of } \mathit{code}(I(z)).$$

The interpretation $\mathcal{J}$ for the term variables of the mixed calculus is given by $\mathcal{J}(Z) = \widehat{I(Z)}$. Here, as for the interpretation of the function variables, we assume a suitable representation of the function $I(Z) : D^n \to \overline{\mathbb{R}}$ by a function $\widehat{I(Z)} : \{0,1\}^{n \cdot k} \to \overline{\mathbb{R}}$. Then,

$$[\![\mathit{expr}]\!]^{\mathcal{M}} \;=\; [\![\, \mathsf{mixed}(\mathit{expr}) \,]\!]^{\mathcal{J}}(\iota)$$

for any algebraic expression *expr*. For any n-ary algebraic term *term*, we have

$$[\![\mathit{term}]\!]^{\mathcal{M}}(d_1, \ldots, d_n) \;=\; [\![\, \mathsf{mixed}(\mathit{term}) \,]\!]^{\mathcal{J}} \left(\iota,\ \langle \mathit{code}(d_1), \ldots, \mathit{code}(d_n) \rangle \right).$$

It is known that the efficiency of the MTBDD-based approach crucially depends on the chosen variable ordering. Having obtained a MTBDD representation for $f = I(\mathit{fct}) : D^n \to \overline{\mathbb{R}}$ (resp. the associated function $\widehat{f} : \{0,1\}^{n \cdot k} \to \overline{\mathbb{R}}$), well-known techniques (e.g. Rudell's sifting algorithm [Rude93]) can be applied to improve the representation. Changing the variable ordering in the MTBDD corresponds to a permutation of the arguments of the function $\widehat{f} : \{0,1\}^{n \cdot k} \to \overline{\mathbb{R}}$. In the final MTBDD, the variables have to be renamed resulting in a MTBDD over $(\vartheta_1, \ldots, \vartheta_{n \cdot k})$.

Example 10.3.1 In Example 10.1.9 (page 266) we presented an algebraic term that describes the "naive" iteration for solving linear equation systems of the form $\mathbf{z} = \mathbf{q} + \mathbf{A} \cdot \mathbf{z}$. The reformulation of that algebraic term as a mixed term is obtained as follows. For simplicity, we assume that $\mathbf{A}$ is a $n \times n$-matrix where $n = 2^k$. We use an encoding for the indices of the rows and columns of the matrix $\mathbf{A}$ by k-bit vectors and describe $\mathbf{A}$ by a function $\{0,1\}^{2k} \to \mathbb{R}$. Let A be the MTBDD over $(\vartheta_1, \ldots, \vartheta_{2k})$ for that function. Similarly, the vectors $\mathbf{q}$, $\mathbf{z}_0 \in \mathbb{R}^n$ can be described by functions $\{0,1\}^k \to \mathbb{R}$ and represented by MTBDDs Q, z_0 over $(\vartheta_1, \ldots, \vartheta_k)$. The size of the so obtained MTBDD representations of $\mathbf{A}$, $\mathbf{q}$ and $\mathbf{z}_0$ depends on the way in which we represent $\mathbf{A}$, $\mathbf{q}$ and $\mathbf{z}_0$ by functions from bit vectors into the reals. Often the standard encoding of integers

[29] The replacement of an individual variable z in a quantifier requires multiple use of that quantifier in the mixed calculus, e.g. the summation quantifier $\sum_z$ in the algebraic mu-calculus has to be replaced by $\sum_{\mathsf{z}_1^z} \cdots \sum_{\mathsf{z}_k^z}$.

$i \in \{1,\ldots,n\}$ by k-bit vectors $\langle b_1,\ldots,b_k\rangle \in \{0,1\}^k$ ordered most significant to least significant (i.e. $i = 1 + \sum_{l=1}^{k} b_l \cdot 2^{k-l}$) and an interleaving of the encodings for the rows and columns of quadratic matrices is used which leads to a representation of $\mathbf{A}$ by a function

$$f : \{0,1\}^{2k} \to I\!R, \quad f(b_1,c_1,\ldots,b_k,c_k) \;=\; \mathbf{A}(i,j)$$

where $\langle b_1,\ldots,b_k\rangle$ is the standard encoding of i (the index for the rows) and $\langle c_1,\ldots,c_k\rangle$ the standard encoding for j (the index for the columns).[30] The vectors $\mathbf{q}$ and $\mathbf{z}_0$ might be represented by functions $\{0,1\}^k \to I\!R$ where $\langle b_1,\ldots,b_k\rangle$ is mapped to the i-th component of $\mathbf{q}$ resp. $\mathbf{z}_0$ (if $\langle b_1,\ldots,b_k\rangle$ is the standard encoding of i). Using these MTBDD representations of $\mathbf{A}$, $\mathbf{q}$ and $\mathbf{z}_0$, the algebraic term *term* of Example 10.1.9 (page 266) corresponds to the mixed term *lim* Z [term $\uparrow$ z_0] where

$$\mathsf{term} \;=\; \lambda \mathsf{i}_1,\ldots,\mathsf{i}_k \left[\mathrm{Q}(\mathsf{i}_1,\ldots,\mathsf{i}_k) \;+\; \sum_{\mathsf{j}_1,\ldots,\mathsf{j}_k} [\, \mathrm{A}(\mathsf{i}_1,\mathsf{j}_1,\ldots,\mathsf{i}_k,\mathsf{j}_k) * \mathsf{Z}(\mathsf{j}_1,\ldots,\mathsf{j}_k) \,] \right].$$

Note that other representations of $\mathbf{A}$, $\mathbf{q}$, $\mathbf{z}_0$ by functions from bit vectors to the reals lead to different MTBDD representations, in which case the individual variables i_l, j_h in the above mixed term have to be permutated. For instance, if we represent $\mathbf{A}$ by the function $f' : \{0,1\}^{2k} \to I\!R$,

$$f'(b_1,\ldots,b_k,c_1,\ldots,c_k) \;=\; \mathbf{A}(i,j) \;\text{ where }\; i = 1 + \sum_{l=1}^{k} b_l \cdot 2^{k-l}, \quad j = 1 + \sum_{l=1}^{k} c_l \cdot 2^{k-l}$$

(where the first k arguments of f' stand for the row while the last k arguments stand for the column) then we have to deal with the mixed term *lim* Z [term' $\uparrow$ z_0] where term' is the mixed term

$$\lambda \mathsf{i}_1,\ldots,\mathsf{i}_k \left[\mathrm{Q}(\mathsf{i}_1,\ldots,\mathsf{i}_k) \;+\; \sum_{\mathsf{j}_1,\ldots,\mathsf{j}_k} [\, \mathrm{A}'(\mathsf{i}_1,\ldots,\mathsf{i}_k,\mathsf{j}_1,\ldots,\mathsf{j}_k) * \mathsf{Z}(\mathsf{j}_1,\ldots,\mathsf{j}_k) \,] \right].$$

Here, Q and z_0 are as before. A' is the MTBDD over $(\vartheta_1,\ldots,\vartheta_{2k})$ for the function f' of above. ■

10.3.3 Computing the semantics of the mixed calculus

We present an algorithm to compute the semantics $[\![\ldots]\!]^{\mathcal{J}}$ of the mixed expressions and terms where we use MTBDDs as data structure for the functions associated with the mixed expressions and terms.[31] In essential, the algorithm works similar to the algorithm of [BCM+90] to compute the BDD representations for the formulas and terms of the relational mu-calculus. The individual variables and the dummy variables serve as variables (i.e. as labellings for the nonterminal nodes) in the MTBDDs.

[30] This convention imposes a recursive structure on the matrix from which efficient recursive algorithms for all standard matrix operations are derived [CFM+93].

[31] Clearly, the correctness of our method is up to the errors that arise from the approximations for the limit operator. An implementation of our method might suffer from rounding errors. Thus, the resulting MTBDD for a mixed expression or term can be viewed as an *approximation* for the function $[\![\ldots]\!]^{\mathcal{J}}$.

In what follows, we assume that IndVar is the set of individual variables used in the mixed calculus. We fix a total ordering $<$ on IndVar that we extend to a total ordering (also called $<$) on $\mathsf{IndVar} \cup \{\vartheta_1, \vartheta_2, \ldots\}$ where we define $\mathsf{z} < \vartheta_1 < \vartheta_2 < \ldots$ for all $\mathsf{z} \in \mathsf{IndVar}$. Then, $[\![\mathsf{expr}]\!]^{\mathcal{J}}$ is represented by a MTBDD over $\langle \mathsf{IndVar}, < \rangle$ while the functions $[\![\mathsf{term}]\!]^{\mathcal{J}}$ for the n-ary mixed terms are represented by MTBDDs over $\langle \mathsf{IndVar} \cup \{\vartheta_1, \ldots, \vartheta_n\}, < \rangle$.

Since the variables in the MTBDDs are ordered, expressions of the form $\mathsf{term}(\mathsf{z}_1, \ldots, \mathsf{z}_n)$ might cause problems, namely when it is *not* the case that $\mathsf{z}_1 < \ldots < \mathsf{z}_n$. Given the MTBDD for term, the replacement of the dummy variable ϑ_i by the individual variable z_i yields a MTBDD that represents the function associated with the expression $\mathsf{term}(\mathsf{z}_1, \ldots, \mathsf{z}_n)$ but the variable ordering in the resulting MTBDD is a new ordering $<'$ with $\mathsf{z}_1 <' \ldots <' \mathsf{z}_n$. On the other hand, the idea not to fix a variable ordering for the MTBDDs leads to the problem that the operators for composing two MTBDDs via an arithmetic operation (that we need to compute the MTBDDs for expressions of the form expr_1 *op* expr_2) would be much more expensive; we would have to adjust the variable orderings of the two MTBDDs. Similar problems occur with terms built by λ-abstraction. For this reason, we fix a total ordering $<$ on IndVar and shrink our attention to *well-formed* terms and expressions.

Definition 10.3.2 [Well-formed mixed expressions and term] *A mixed expression and term is called* well-formed *(with respect to the fixed ordering $<$) iff,*

- *for each subexpression of the form* $\mathsf{term}(\mathsf{z}_1, \ldots, \mathsf{z}_n)$,
- *for each subterm of the form* $\lambda \mathsf{z}_1, \ldots, \mathsf{z}_n[\mathsf{expr}]$,

we have $\mathsf{z}_1 < \ldots < \mathsf{z}_n$.

Any mixed expression or term can be rewritten as a well-formed mixed expression or term. For this, we replace

- each subexpression $\mathsf{term}(\mathsf{z}_1, \ldots, \mathsf{z}_n)$ by $\sum_{\zeta_1,\ldots,\zeta_n} [\ \mathsf{term}(\zeta_1, \ldots, \zeta_n) * \mathsf{bexpr}\ ]$
- each subterm $\lambda \mathsf{z}_1, \ldots, \mathsf{z}_n[\mathsf{expr}]$ by $\lambda \zeta_1, \ldots, \zeta_n \left[\sum_{\mathsf{z}_1,\ldots,\mathsf{z}_n} [\ \mathsf{expr} * \mathsf{bexpr}\ ] \right]$

where

$$\mathsf{bexpr} \ = \ \bigwedge_{1 \le i \le n} (\zeta_i \leftrightarrow \mathsf{z}_i).$$

Here, $\zeta_1, \ldots, \zeta_n$ are auxiliary individual variables ordered by $\zeta_1 < \ldots < \zeta_n$. For instance, if $\mathsf{z}_2 < \mathsf{z}_1$ then $\mathsf{term}(\mathsf{z}_1, \mathsf{z}_2)$ is replaced by

$$\sum_{\zeta_1,\zeta_2} [\ \mathsf{term}(\zeta_1, \zeta_2) * \mathsf{bexpr}\ ]$$

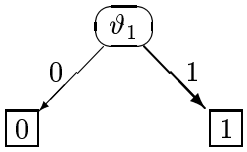
where bexpr is $(\mathsf{z}_1 \leftrightarrow \zeta_1) \wedge (\mathsf{z}_2 \leftrightarrow \zeta_2)$. The semantics of the mixed expressions/terms and the so obtained well-formed expressions/terms are the same.[32]

Computing the MTBDDs for the well-formed mixed expressions and terms: In what follows, we fix an interpretation $\mathcal{J}$ for the term variables. More precisely, we assume

[32] Note that the fresh individual variables $\zeta_1, \zeta_2, \ldots$ does not occur free in the resulting expressions/terms. Hence, the semantics for the resulting well-formed expressions (or n-ary terms) can be viewed as a function $(\mathsf{IndVar} \times \{0,1\}) \to \overline{I\!R}$ (or $(\mathsf{IndVar} \times \{0,1\}) \times \{0,1\}^n \to \overline{I\!R}$) rather than a function $(\mathsf{IndVar} \cup \{\zeta_1, \zeta_2, \ldots\} \to \{0,1\}) \to \overline{I\!R}$ (or $(\mathsf{IndVar} \cup \{\zeta_1, \zeta_2, \ldots\} \to \{0,1\}) \times \{0,1\}^n \to \overline{I\!R}$).

$\textsc{Mtbdd}^{\mathcal{J}}[\![q]\!]$ denotes the MTBDD that consists of a terminal vertex labelled by q.

$\textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{z}]\!]$ denotes the BDD



$$\textsc{Mtbdd}^{\mathcal{J}}[\![\ \mathsf{expr}_1\ op\ \mathsf{expr}_2\ ]\!] = \textsc{Apply}\left(\textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}_1]\!], \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}_2]\!], op\right)$$

$$\textsc{Mtbdd}^{\mathcal{J}}[\![\ \mathsf{term}(\mathsf{z}_1,\ldots,\mathsf{z}_n)\ ]\!] = \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{term}]\!]\{\vartheta_1 \leftarrow \mathsf{z}_1,\ldots,\vartheta_n \leftarrow \mathsf{z}_n\}$$

$$\textsc{Mtbdd}^{\mathcal{J}}[\![\ \textstyle\sum_{\mathsf{z}}\ [\mathsf{expr}]\ ]\!] = \textsc{Apply}\left(\ \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]|_{\mathsf{z}=0},\ \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]|_{\mathsf{z}=1},\ +\ \right)$$

$$\textsc{Mtbdd}^{\mathcal{J}}[\![\ \min_{\mathsf{z}}\ [\mathsf{expr}]\ ]\!] = \textsc{Apply}\left(\ \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]|_{\mathsf{z}=0},\ \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]|_{\mathsf{z}=1},\ op_{min}\ \right)$$

$$\textsc{Mtbdd}^{\mathcal{J}}[\![\ \max_{\mathsf{z}}\ [\mathsf{expr}]\ ]\!] = \textsc{Apply}\left(\ \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]|_{\mathsf{z}=0},\ \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]|_{\mathsf{z}=1},\ op_{max}\ \right)$$

$$\textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{Z}]\!] = \mathcal{J}(\mathsf{Z})$$

$$\textsc{Mtbdd}^{\mathcal{J}}[\![\ \lambda\mathsf{z}_1,\ldots,\mathsf{z}_n[\mathsf{expr}]\ ]\!] = \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]\{\mathsf{z}_1 \leftarrow \vartheta_1,\ldots,\mathsf{z}_n \leftarrow \vartheta_n\}$$

$\textsc{Mtbdd}^{\mathcal{J}}[\![\ lim\ \mathsf{Z}[\ \mathsf{term} \uparrow \mathsf{term}_0\ ]\ ]\!]$ denotes the MTBDD that is returned by

$$\textsc{Iterate}_{\epsilon}(i_{max}, \mathsf{Z}, \mathsf{term}, \mathsf{term}_0).$$

$\textsc{Mtbdd}^{\mathcal{J}}[\![\ iterate\ \mathsf{Z}[\ \mathsf{term} \uparrow^k \mathsf{term}_0\ ]\ ]\!]$ denotes the MTBDD that is returned by

$$\textsc{Iterate}(k, \mathsf{Z}, \mathsf{term}, \mathsf{term}_0).$$

Figure 10.9: Computing the MTBDDs for the mixed expressions and terms

that, for each n-ary term variable Z, the function $\mathcal{J}(\mathsf{Z}) : \{0,1\}^n \to \overline{I\!R}$ is represented by a MTBDD (also called $\mathcal{J}(\mathsf{Z})$) over $(\vartheta_1,\ldots,\vartheta_n)$. Each well-formed mixed expression expr is associated with a MTBDD $\textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]$ over $\langle \mathsf{IndVar}, < \rangle$, each n-ary well-formed algebraic term term with a MTBDD $\textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{term}]\!]$ over $\langle \mathit{IndVar} \cup \{\vartheta_1,\ldots,\vartheta_n\}, < \rangle$. We compute $\textsc{Mtbdd}^{\mathcal{J}}[\![\ldots]\!]$ for well-formed algebraic expressions and terms by structural induction as shown in Figure 10.9 on page 292. Here, we use the operators of [Brya86, CFM+93]; see Section 12.3 (page 317). For the computation of the MTBDD for $\mathsf{expr}_1\ op\ \mathsf{expr}_2$ we have to combine the MTBDDs for expr_1 and expr_2 via the binary operator op. For this, we use the well-known Apply-operator that combines two MTBDDs via an arbitrary binary operator (see page 317). The MTBDD for $\mathsf{term}(\mathsf{z}_1,\ldots,\mathsf{z}_n)$ is obtained from the MTBDD for *term* by renaming the dummy variables $\vartheta_1,\ldots,\vartheta_n$ into $\mathsf{z}_1,\ldots,\mathsf{z}_n$. This renaming respects the variable ordering $<$ as we have $\mathsf{z}_1 < \ldots < \mathsf{z}_n$. λ-abstraction requires the converse operation where individual variables are replaced by dummy variables. For the expressions built e.g. by the quantifier $\sum_{\mathsf{z}}[\mathsf{expr}]$, we have to sum up the values for expr when z ranges over all possible values of the underlying domain. In

$i := 0$; $\mathrm{Q}_0 := \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{term}_0]\!]$;
Repeat
  $i := i + 1$; $\mathrm{Q}_i := \textsc{Mtbdd}^{\mathcal{J}[\mathsf{Z}:=\mathrm{Q}_{i-1}]}[\![\mathsf{term}]\!]$;
  $\mathrm{B} := \textsc{Mtbdd}^{\mathcal{J}}[\![\ \lambda\overline{\mathsf{z}}[\ |\mathrm{Q}_{i-1}(\overline{\mathsf{z}}) - \mathrm{Q}_i(\overline{\mathsf{z}})| < \epsilon\ ]\ ]\!]$
until $i = i_{max}$ or $\mathrm{B} = \lambda\overline{\mathsf{z}}[1]$;
$\mathrm{Q} := \textsc{Mtbdd}^{\mathcal{J}}[\![\ \lambda\overline{\mathsf{z}}\ [\ (1 - \mathrm{B}(\overline{\mathsf{z}})) * \bot\ +\ \mathrm{B}(\overline{\mathsf{z}}) * \mathrm{Q}_i(\overline{\mathsf{z}})\ ]\ ]\!]$;
Return Q.

Figure 10.10: The procedure $\textsc{Iterate}_\epsilon(i_{max}, \mathsf{Z}, \mathsf{term}, \mathsf{term}_0)$

our case, we just have the values 0 and 1. Thus, we first compute the MTBDD for expr. Then, by removing the z-labelled vertices in that MTBDD, we obtain the restrictions to the cases where $\mathsf{z} = 0$ and $\mathsf{z} = 1$. This yields the MTBDDs $\textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]|_{\mathsf{z}=0}$ and $\textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]|_{\mathsf{z}=1}$ for the functions $(\mathsf{IndVar} \to \{0,1\}) \to \overline{I\!R}$,

$$\iota \mapsto [\![\mathsf{expr}]\!]^{\mathcal{J}}(\iota[\mathsf{z} := 0]) \quad \text{and} \quad \iota \mapsto [\![\mathsf{expr}]\!]^{\mathcal{J}}(\iota[\mathsf{z} := 1]).$$

Finally, we combine these MTBDDs via the operator $\textsc{Apply}(\ldots, +)$. Similar ideas are used for the quantifiers $\min_{\mathsf{z}}$ and $\max_{\mathsf{z}}$. The only difference is that in the last step the MTBDDs $\textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]|_{\mathsf{z}=0}$ and $\textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{expr}]\!]|_{\mathsf{z}=1}$ have to be combined via $\textsc{Apply}(\ldots, op_{min})$ and $\textsc{Apply}(\ldots, op_{max})$ respectively.

The limit operator: To approximate the semantics of *lim* Z [term $\uparrow$ term_0] we use iteration on MTBDDs where we stop latest after a fixed number of iteration steps. Let n be the arity of Z (hence, n is at the same time the arity of term and term_0). Let $\overline{\mathsf{z}} = (\mathsf{z}_1, \ldots, \mathsf{z}_n)$ be an n-tuple of individual variables with $\mathsf{z}_1 < \ldots < \mathsf{z}_n$. We fix some some "sufficiently small" value $\epsilon > 0$ and some natural number i_{max} (the maximal number of iterations). The procedure $\textsc{Iterate}_\epsilon(i_{max}, \mathsf{Z}, \mathsf{term}, \mathsf{term}_0)$ (shown in Figure 10.10 on page 293) successively computes the MTBDDs $\mathrm{Q}_0, \mathrm{Q}_1, \ldots$ where

$$\mathrm{Q}_0 = \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{term}_0]\!],\ \mathrm{Q}_i = \textsc{Mtbdd}^{\mathcal{J}[\mathsf{Z}:=\mathrm{Q}_{i-1}]}[\![\mathsf{term}]\!],\ i = 1, 2, \ldots.$$

Here, we suppose an extension of $+$ and $*$ to operators on the extended reals where $\bot * q = q * \bot = \bot$ for all $q \in \overline{I\!R} \setminus \{0\}$, $0 * \bot = \bot * 0 = 0$ and $\bot + q = q + \bot = \bot$ for all $q \in \overline{I\!R}$. The iteration terminates if the maximal difference between the function values of f_{Q_i} and $f_{\mathrm{Q}_{i-1}}$ is less than ϵ, i.e. if $|f_{\mathrm{Q}_{i-1}}(\overline{b}) - f_{\mathrm{Q}_i}(\overline{b})| < \epsilon$ for all $\overline{b} \in \{0,1\}^n$. If this condition is *not* satisfied after i_{max} iterations then,

- for those bit vectors $\overline{b}$ where $|f_{\mathrm{Q}_{i-1}}(\overline{b}) - f_{\mathrm{Q}_i}(\overline{b})| \geq \epsilon$: convergence of the sequence $(f_{\mathrm{Q}_i}(\overline{b}))_{i\geq 0}$ is not "detected" and we assume that the limit operator on the extended reals returns $\bot$,
- for those bit vectors $\overline{b}$ where $|f_{\mathrm{Q}_{i-1}}(\overline{b}) - f_{\mathrm{Q}_i}(\overline{b})| < \epsilon$: we assume convergence of the sequence $(f_{\mathrm{Q}_i}(\overline{b}))_{i\geq 0}$ and return $f_{\mathrm{Q}_{i_{max}}}(\overline{b})$ as an approximation for $\lim f_{\mathrm{Q}_i}(\overline{b})$.

Note that the BDD B represents the (characteristic function of the) set

$$B = \left\{ \overline{b} \in \{0,1\}^n : |f_{\mathrm{Q}_{i-1}}(\overline{b}) - f_{\mathrm{Q}_i}(\overline{b})| < \epsilon \right\}.$$

Thus, the condition "B $=$ $\lambda\overline{\mathsf{z}}[1]$" is fulfilled iff $B = \{0,1\}^n$ iff $|f_{\mathrm{Q}_{i-1}}(\overline{b}) - f_{\mathrm{Q}_i}(\overline{b})| < \epsilon$ for all $\overline{b} \in \{0,1\}^n$.

The bounded iteration operator: The procedure ITERATE$(k, \mathsf{Z}, \mathsf{term}, \mathsf{term}_0)$ (shown in Figure 10.11 on page 294) that we use to compute the semantics of the bounded iteration operator is almost the same as ITERATE$_\epsilon(\cdot)$; the only difference being that we do not care about convergence and just halt after exactly k iteration steps.

$i := 0$; $\mathrm{Q}_0 := \textsc{Mtbdd}^{\mathcal{J}}[\![\mathsf{term}_0]\!]$;
Repeat
  $i := i + 1$; $\mathrm{Q}_i := \textsc{Mtbdd}^{\mathcal{J}[\mathsf{Z}:=\mathrm{Q}_{i-1}]}[\![\mathsf{term}]\!]$;
until $i = k$;
Return Q_k.

Figure 10.11: The procedure ITERATE$(k, \mathsf{Z}, \mathsf{term}, \mathsf{term}_0)$

The mixed calculus as a language for MTBDDs: The algorithm of Figure 10.9 (page 292) yields the theoretical foundations for a tool that takes as its input certain MTBDDs $\mathrm{Q}_1, \ldots, \mathrm{Q}_l$ and a closed mixed term *term* built from these MTBDDs and that automatically generates the MTBDD representation of (the semantics of) that term. In this sense, the mixed calculus can be viewed as a language for manipulating MTBDDs where the closed mixed terms just describe which operations should be performed on the MTBDDs that occur in that term. Thus, the closed mixed terms can be viewed as *procedures* whose parameters are the MTBDDs occurring in that term and that outputs the MTBDD associated with that term.

Applications: In Example 10.1.9 (page 266) we presented an algebraic term that describes the "naive" iteration for solving linear equation systems of the form $\mathbf{z} = \mathbf{q} + \mathbf{A} \cdot \mathbf{z}$. The corresponding mixed term (see Example 10.3.1, page 289) stands for a procedure that computes the MTBDD for (an approximation of) the solution $\mathbf{z}$. Similarly, the corresponding mixed terms of the algebraic terms presented in Example 10.1.6 (page 264) and Example 10.1.10 (page 266) yield MTBDD-based methods for computing shortest paths or eigenvectors.[33] In Section 10.2, we saw that the algebraic mu-calculus subsumes a wide range of temporal and modal logics that can serve as specification languages for parallel systems. Our MTBDD-based algorithm applied to the mixed terms obtained from the algebraic terms $term_\varphi$ for a formula φ (or boolean terms $bterm_\Phi$ for a state formula Φ) yields a *symbolic model checker* for these logics. Of course, we cannot expect that the obtained MTBDD-based methods are efficient in any of the above mentioned possible applications. For example, the MTBDD approach is known to be efficient for verifying probabilistic systems [HarG98] while it is not for arithmetic circuits [Zhao96]. Thus, we might expect that the resulting symbolic model checking algorithm for *PCTL* is efficient while the obtained MTBDD-based method for word level *CTL* is not.

[33] For further discussions about the use of MTBDDs for computing shortest paths or MTBDD-based methods for matrix operations, see [CFM+93, BFG+93, HMP+94, FMY97].

10.4 Symbolic model checking for probabilistic processes

At the end of the previous section, we mentioned that the algebraic mu-calculus (with the MTBDD-based method for computing the semantics) yields a symbolic model checker for all logics that are contained in the algebraic mu-calculus; in particular, we obtain a symbolic model checker for *PCTL*. In this section we have a more detailed look of how to use the algebraic mu-calculus (or the mixed calculus) to obtain MTBDD-based verification methods for probabilistic processes. Section 10.4.2 is concerned with *PCTL* model checking. In Section 10.4.3 we briefly sketch how the MTBDD-based approach can be applied for a symbolic method to decide strong or weak bisimulation equivalence for fully probabilistic processes.

10.4.1 Representing probabilistic systems by MTBDDs

The basic idea behind the MTBDD-approach is the use a symbolic representation of a probabilistic system by MTBDDs as in [HarG98] (see also [BCH$^+$97]). In the fully probabilistic case, the ideas of the non-probabilistic case [BCM$^+$90, McMil92, CGL93] where transition systems are described in terms of BDDs that represent boolean functions (i.e. functions from bit vectors into $\{0,1\}$) can be adapted. Using an encoding of the states by bit vectors of length k, the transition probability function $\mathbf{P} : S \times S \to [0,1]$ can be viewed as a function $\{0,1\}^{2k} \to [0,1]$ and described by a MTBDD. Of course, the size of the MTBDD representation of the system depends on the encoding of the states and the chosen ordering of the variables in the MTBDD. In most cases, an interleaving of the components of the bit vectors for the starting state and the end state of the transitions yields an efficient representation. This corresponds to the replacement of the transition probability function $\mathbf{P}$ by the function $\widehat{\mathbf{P}} : \{0,1\}^{2k} \to [0,1]$,

$$\widehat{\mathbf{P}}(b_1, c_1, \ldots, b_k, c_k) \;=\; \mathbf{P}(s,t)$$

where $\langle b_1, \ldots, b_k \rangle$ is the encoding of state s and $\langle c_1, \ldots, c_k \rangle$ the encoding of state t. The resulting MTBDD representation can be improved using well-known techniques like Rudell's sifting algorithm [Rude93] or other heuristics, see e.g. [FMK91, MKR92, BMS95].

Example 10.4.1 [MTBDD representation of the communication protocol] We consider a variant of the simple communication protocol of Example 1.2.1 (page 19). The sender sends a message to the medium, which in turn tries to deliver the message to the receiver. With probability $\frac{1}{100}$, the messages get lost, in which case the medium tries again to deliver the message. With probability $\frac{1}{100}$, the message is corrupted (but delivered); with probability $\frac{98}{100}$, the correct message is delivered. When the (correct or faulty) message is delivered the receiver acknowledges the receipt of the message. For simplicity, we assume that the acknowledgement cannot be corrupted or lost. We describe the system in a simplified way where we omit all irrelevant states (e.g. the state where the receiver acknowledges the receipt of the correct message). We use the following four states:

s_{init}: the state in which the sender passes the message to the medium,

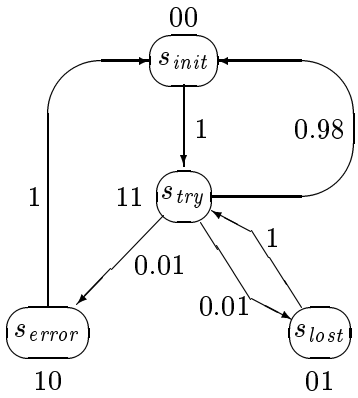


Figure 10.12: The simple communication protocol

s_{try}: the state in which the medium tries to deliver the message,
s_{lost}: the state reached when the message is lost,
s_{error}: the state reached when the message is corrupted

and the encoding $code(s_{init}) = 00$, $code(s_{try}) = 11$, $code(s_{lost}) = 01$, $code(s_{error}) = 10$. Then, the associated function $\widehat{\mathbf{P}} : \{0,1\}^4 \to [0,1]$ is given by:

$$(b_1, c_1, b_2, c_2) \mapsto \begin{cases} 1 & : \text{ if } b_1c_1b_2c_2 \in \{0101, 0111, 1000\} \\ \frac{1}{100} & : \text{ if } b_1c_1b_2c_2 \in \{1011, 1110\} \\ \frac{98}{100} & : \text{ if } b_1c_1b_2c_2 = 1010 \\ 0 & : \text{ otherwise.} \end{cases}$$

The system and the encodings are shown in Figure 10.12 (page 296); the MTBDD representation in Figure 10.13 (page 296). The thick lines stand for the "right" edges, the thin

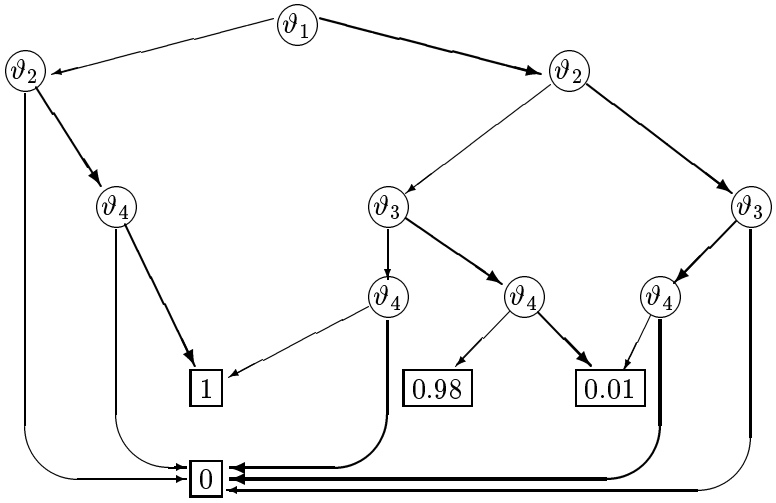


Figure 10.13: The MTBDD representation of the simple communication protocol

lines for the "left" edges. ■

Similarly, we can deal with action-labelled fully probabilistic systems. Let $(S, \mathbf{P}, Act)$ be a finite action-labelled fully probabilistic system. We use an encoding of the actions

in $\{0,1\}^h$ (where $h = \lceil \log |Act| \rceil$) and states in $\{0,1\}^k$ and replace $\mathbf{P}$ by a function $\widehat{\mathbf{P}} : \{0,1\}^{2k+h} \to [0,1]$ that we represent by a MTBDD.

Dealing with a concurrent probabilistic system $\mathcal{S} = (S, Steps)$, the situation is more complicate since the outgoing transitions of a state s are given by $Steps(s)$ which is a *set* of distributions on the state space S. One possibility to get a MTBDD-representation of $\mathcal{S}$ is to fix an enumeration $\nu_1^s, \nu_2^s, \ldots, \nu_{m_s}^s$ of the outgoing transitions of s. Then, we extend the i-th transition of s by its "identification number" i and deal with a transition probability function

$$S \times Id\# \times S \to [0,1], \; (s,i,t) \mapsto \nu_i^s(t)$$

where $Id\#$ stands for the set of identification numbers (e.g. $Id\# = \{1, \ldots, m_{max}\}$ where $m_{max} = \max_{s \in S} |Steps(s)|$) and ν_i^s is the i-th distribution in $Steps(s)$ according to the fixed enumeration of $Steps(s)$. (Here, we put $\nu_i^s(t) = 0$ for all $t \in S$ if $i > |Steps(s)|$.) Then, using an encoding for the states and identification numbers by bit vectors, the above function $S \times Id\# \times S \to [0,1]$ can be viewed as a function from bit vectors into the reals and represented by a MTBDD. As far as the author knows, whether or not such a MTBDD-representation of a concurrent probabilistic system is efficient for verification purposes is not yet investigated.[34] However, it seems to be much simpler to require a concurrent probabilistic system whose transitions can be described by a function $\mathbf{P} : S \times S \to [0,1]$ (or $\mathbf{P} : S \times Act \times S \to [0,1]$ in the action-labelled case) in a more natural way. This is the case for stratified systems (cf. Notation 3.2.4, page 40) or reactive systems (cf. Notation 3.3.11, page 52).[35] In that case, we can use the same ideas as for non-probabilistic or fully probabilistic systems and deal with an encoding of the states (and actions) by bit vectors which turns the above transition probability function $\mathbf{P}$ into a function from bit vectors into the reals and allows for a natural symbolic representation by a MTBDD.

10.4.2 Symbolic model checking for *PCTL*

In Section 10.2.3 (page 279 ff) we saw that any *PCTL* formula Φ can be transformed into an "equivalent" boolean term $bterm_\Phi$. For this transformation, we used 1-ary function symbols Sat_a (that represent the sets $Sat(a) = \{s \in S : a \in \mathcal{L}(s)\}$) and a binary function symbol P (that represents the transition probability matrix $\mathbf{P}$). To obtain a symbolic model checking algorithm for *PCTL*, we translate $bterm_\Phi$ into the mixed calculus and apply the algorithm to compute the BDD representation for $\mathsf{mixed}(bterm_\Phi)$. For this, we need the MTBDD representation P for the transition probability matrix $\mathbf{P}$ as described above. Moreover, for each atomic proposition a, we need a BDD representation SAT_a for the (characteristic functions of) the set $Sat(a) = \{s \in S : a \in \mathcal{L}(s)\}$. Dealing with stratified systems, we also need a BDD S_{prob} that represents the set S_{prob} of probabilistic states. Then, the mixed term $\mathsf{mixed}(bterm_\Phi)$ is obtained from $bterm_\Phi$ by replacing the function symbols Sat_a, S_{prob} and P by the corresponding (MT)BDDs SAT_a, S_{prob} and P; the individual variables s, t (that we used in the algebraic terms to range over the states)

[34] The author doubts whether it is. The operators on these MTBDDs that we would have to perform seem to be quite complicate because of the auxiliary (meaningless) components for the identification numbers.

[35] Recall that in Section 3.2, page 40, we argued that stratified systems have the same expressiveness as concurrent probabilistic systems.

by k-tuples $(\mathsf{s}_1,\ldots,\mathsf{s}_k)$, $(\mathsf{t}_1,\ldots,\mathsf{t}_k)$ of individual variables (where e.g. s_i stands for the i-th component of the encoding of state s). For example, if the MTBDD representation P is based on a description of $\mathbf{P}$ by the function $\widehat{\mathbf{P}}$ that interleaves the bits for the starting and end state of the transitions then any subexpression $P(s,t)$ of $bterm_\Phi$ has to be replaced by $\mathrm{P}(\mathsf{s}_1,\mathsf{t}_1,\ldots,\mathsf{s}_k,\mathsf{t}_k)$. We obtain a closed mixed term $\mathsf{mixed}(bterm_\Phi)$ where the associated BDD – that we get by applying the algorithm for computing the semantics of the mixed calculus – represents the characteristic function of $Sat(\Phi)=\{s\in S : s\models\Phi\}$.[36]

Example 10.4.2 We consider the system in Example 10.4.1 (page 295). We use two atomic propositions a_1, a_2 and the labelling function

$$\mathcal{L}(s_{init})=\emptyset,\ \mathcal{L}(s_{try})=\{a_1,a_2\},\ \mathcal{L}(s_{lost})=\{a_2\},\ \mathcal{L}(s_{error})=\{a_1\}.$$

We regard the *PCTL* formula $\Phi=\mathsf{Prob}_{>0.989898}(\varphi)$ where $\varphi = \neg error\ \mathcal{U}\ del$ and

- $error = a_1\wedge\neg a_2$ (i.e. $Sat(error)=\{s_{error}\}$),
- $del=\neg a_1\wedge\neg a_2$ (i.e. $Sat(del)=\{s_{init}\}$).

Intuitively, Φ states that the message will eventually be delivered with some probability >0.989898 when interpreted over the state s_{try}. We describe how our method works to get the BDD for the *PCTL* formula Φ. For this, we first construct the MTBDD for the path formula φ. The algebraic term $term_\varphi$ is (more precisely, can be reformulated to)

$$lfp\ Z\ \left[\ \lambda s\ \left[\ \max\left\{\ Sat_{del}(s),\ (1-Sat_{error}(s))*\left(\sum_t [P(s,t)*Z(t)]\right)\ \right\}\ \right]\ \right]$$

Hence, we get the mixed term $lfp\ \mathsf{Z}\ [\ \lambda\mathsf{s}_1,\mathsf{s}_2\ [\mathsf{expr}]\ ]$ where expr is

$$\max\left\{\ \mathrm{SAT}_{del}(\mathsf{s}_1,\mathsf{s}_2),\ (1-\mathrm{SAT}_{error}(\mathsf{s}_1,\mathsf{s}_2))*\left(\sum_{\mathsf{t}_1,\mathsf{t}_2}[\mathrm{P}(\mathsf{s}_1,\mathsf{t}_1,\mathsf{s}_2,\mathsf{t}_2)*\mathsf{Z}(\mathsf{t}_1,\mathsf{t}_2)]\right)\ \right\}.$$

Then, our algorithm applied to that mixed term uses the procedure $\textsc{Iterate}_\epsilon(\cdot)$ (see Figure 10.10, page 293) which successively computes the MTBDDs $\mathrm{Q}_0,\mathrm{Q}_1,\mathrm{Q}_2,\ldots$ for the mixed terms $\mathsf{term}_0=\lambda\mathsf{s}_1,\mathsf{s}_2[0]$, $\mathsf{term}_1,\mathsf{term}_2,\ldots$ where term_{i+1} is given by

$$\lambda\mathsf{s}_1,\mathsf{s}_2\ \left[\ \max\left\{\mathrm{SAT}_{del}(\mathsf{s}_1,\mathsf{s}_2),\mathrm{C}(\mathsf{s}_1,\mathsf{s}_2)*\left(\sum_{\mathsf{t}_1,\mathsf{t}_2}[\mathrm{P}(\mathsf{s}_1,\mathsf{t}_1,\mathsf{s}_2,\mathsf{t}_2)*\mathrm{Q}_i(\mathsf{t}_1,\mathsf{t}_2)]\right)\ \right\}\ \right].$$

The BDD SAT_{del} represents the sets $Sat(del)=\{s_{del}\}$ (which corresponds to be boolean function $(b_1,b_2)\mapsto\neg b_1\wedge\neg b_2$); C is a BDD for $\{s\in S : s\not\models error\}=\{s_{init},s_{try},s_{lost}\}$ (which corresponds to the boolean function $(b_1,b_2)\mapsto\neg b_1\vee b_2$). SAT_{del} and C are shown in Figure 10.14 (page 299). The MTBDDs $\mathrm{Q}_0,\mathrm{Q}_1,\ldots$ for the mixed terms $\mathsf{term}_0,\mathsf{term}_1,\ldots$ represent the functions

$$\begin{pmatrix}0\\0\\0\\0\end{pmatrix},\ \begin{pmatrix}1\\0\\0\\0\end{pmatrix},\ \begin{pmatrix}1\\0\\0\\0.98\end{pmatrix},\ \begin{pmatrix}1\\0.98\\0\\0.98\end{pmatrix},\ \begin{pmatrix}1\\0.98\\0\\0.9898\end{pmatrix},\ \begin{pmatrix}1\\0.9898\\0\\0.989898\end{pmatrix},\ldots$$

[36] This symbolic model checking procedure uses an iterative method (that approximates the least fixed point of a certain operator) for the handling of unbounded until $\mathcal{U}$. This is unlike the *PCTL* model checking algorithms of Hansson & Jonsson [HaJo94] or Bianco & de Alfaro [BidAl95] that work with linear equation systems or linear optimization problems respectively.

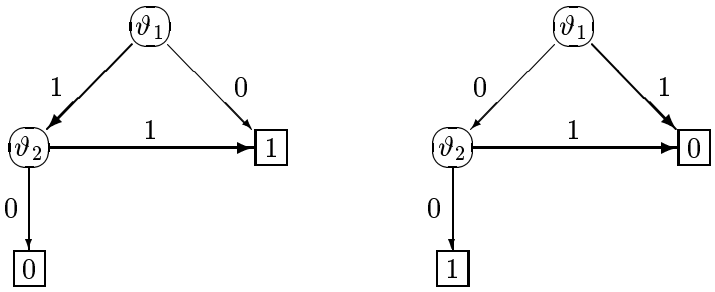


Figure 10.14: The BDDs C and SAT_{del}

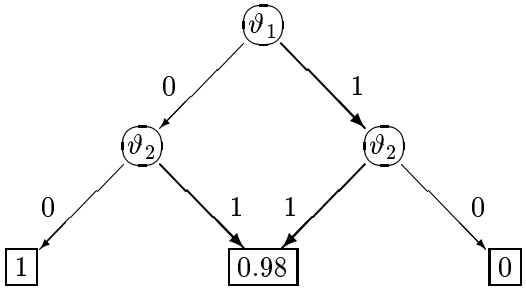


Figure 10.15: The MTBDD Q_3 for the mixed term term_3

where we use the vector notation $(f(0,0), f(0,1), f(1,0), f(1,1))$ (written as a column) to denote a function $f : \{0,1\}^2 \to I\!R$. Note that

- $f_{\mathrm{Q}_{i+1}}(1,0) = 0$ as $f_{\mathrm{B}}(1,0) = 0$ and $f_{\text{SAT}_{del}}(1,0) = 0$,
- $f_{\mathrm{Q}_{i+1}}(0,0) = 1$ as $f_{\text{SAT}_{del}}(0,0) = 1$,
- $f_{\mathrm{Q}_{i+1}}(0,1) = f_{\mathrm{Q}_i}(1,1)$ as $f_{\mathrm{B}}(0,1) = 1$, $f_{\text{SAT}_{del}}(0,1) = 0$ and

$$f_{\mathrm{P}}(0,c_1,1,c_2) \;=\; \begin{cases} 1 & : \text{ if } c_1c_2 = 11 \\ 0 & : \text{ otherwise} \end{cases}$$

- $f_{\mathrm{Q}_{i+1}}(1,1) \;=\; \frac{98}{100} \cdot f_{\mathrm{Q}_i}(0,0) \;+\; \frac{1}{100} \cdot f_{\mathrm{Q}_i}(0,1) \;+\; \frac{1}{100} \cdot f_{\mathrm{Q}_i}(1,0)$
  as $f_{\mathrm{B}}(1,1) = 1$, $f_{\text{SAT}_{del}}(1,1) = 0$ and

$$f_{\mathrm{P}}(1,c_1,1,c_2) \;=\; \begin{cases} \frac{98}{100} & : \text{ if } c_1c_2 = 00 \\ \frac{1}{100} & : \text{ if } c_1c_2 \in \{01,10\} \\ 0 & : \text{ if } c_1c_2 = 11. \end{cases}$$

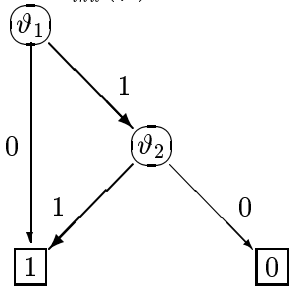
For instance, the MTBDD Q_3 is shown in Figure 10.15 (page 299). Our algorithm returns some MTBDD Q_i as an approximation for the function $s \mapsto p_s(\varphi)$ which is given by

$$p_{s_{lost}}(\varphi) = p_{s_{try}}(\varphi) = \tfrac{98}{99}, \quad p_{s_{error}}(\varphi) = 0 \;\text{ and }\; p_{s_{init}}(\varphi) = 1.$$

For the *PCTL* formula $\Phi = \mathsf{Prob}_{>0.989898}(\varphi)$, our algorithm computes the MTBDD Q for φ (as explained above) and then evaluates the mixed term

$$\lambda \mathsf{s}_1, \mathsf{s}_2[\; \mathrm{Q}(s) > 0.989898 \;]$$

which yields the BDD shown on the right. ■



10.4.3 Deciding bisimulation equivalence

In the non-probabilistic case, the set of bisimulation equivalence of a labelled transition system can be characterized as the greatest fixed point of a monotonic set-valued operator [Miln89]. If S is the state space then the set of bisimulation equivalence classes is the greatest fixed point of $F : 2^{S\times S} \to 2^{S\times S}$ where $F(Z)$ is the set of pairs $(s,s') \in Z$ such that, for all $a \in Act$:

(1) $s \xrightarrow{a} t$ implies $s' \xrightarrow{a} t'$ for some $t' \in S$ with $(t,t') \in Z$
(2) $s' \xrightarrow{a} t'$ implies $s \xrightarrow{a} t$ for some $t \in S$ with $(t,t') \in Z$.

On the basis of Tarski's fixed point theorem, this observation leads to an iterative method for computing the bisimulation equivalence relation $\sim$:

$$\sim \;=\; gfp(F) \;=\; \bigcap_{i\geq 0} F^i(S\times S).$$

Clearly, each of the relations $F^i(S\times S)$ is an equivalence. For Z to be an equivalence on S, the set $F(Z)$ consists of all pairs $(s,s') \in Z$ such that, for all $a \in Act$ and $t \in S$:

$$s \xrightarrow{a} t' \text{ for some } t' \in S \text{ with } (t,t') \in Z \text{ iff } s' \xrightarrow{a} t' \text{ for some } t' \in S \text{ with } (t,t') \in Z.$$

Burch et al [BCM$^+$90] (see also [EFT93]) take up this characterization of $\sim$ and describe $\sim$ by the following term of the relational mu-calculus

$$bterm \;=\; gfp\ Z\ [\ \lambda s,s'[\ \forall a\ \forall t\ [bexpr]\ ]\ ]$$

where $bexpr$ is $\exists t'\ [\ Z(t,t')\ \wedge\ R(s,a,t')\ ] \quad\leftrightarrow\quad \exists t'\ [\ Z(t,t')\ \wedge\ R(s',a,t')\ ]$.[37] Thus, the BDD-based method of [BCM$^+$90] to evaluate the terms of the relational mu-calculus yields a symbolic method for computing bisimulation equivalence classes.

We now explain how this idea can be adapted for the probabilistic case. Recall the definition of bisimulation equivalence for probabilistic systems.[38] We consider a finite action-labelled fully probabilistic or reactive system $\mathcal{S} = (S, Act, \mathbf{P})$ and use a ternary function symbol P to represent $\mathbf{P}$. As in the non-probabilistic case, bisimulation equivalence can be described as the greatest fixed point of an operator on $2^{S\times S}$. We consider the operator $F_\sim : 2^{S\times S} \to 2^{S\times S}$ where $F_\sim(Z)$ is the set of all pairs $(s,s') \in Z$ such that, for all $a \in Act$ and $t \in S$:

$$\sum_{\substack{t'\in S\\(t,t')\in Z}} \mathbf{P}(s,a,t') \;=\; \sum_{\substack{t'\in S\\(t,t')\in Z}} \mathbf{P}(s',a,t').$$

First, we observe that – in contrast to the non-probabilistic case – this operator is *not* monotonic. For instance, consider the system shown in Figure 10.16 (page 301). Let Z be the smallest equivalence relation on S that contains (s,s') and that identifies the states v_1, v_2, v' and $Z' = Z \cup \{(v',u')\}$. Then, we have $Z \subseteq Z'$ while $F_\sim(Z) \not\subseteq F_\sim(Z')$. For instance, $(s,s') \in F_\sim(Z) \setminus F_\sim(Z')$. To see why $(s,s') \notin F_\sim(Z')$ consider the state $t = v'$ and the set

$$T \;=\; \{t' \in S : (t,t') \in Z'\} \;=\; \{t' \in S : (v',t') \in Z'\} \;=\; \{v_1, v_2, v, u'\}.$$

[37] Here, R is a ternary predicate (function) symbol that represents the underlying transition relation $\longrightarrow \subseteq S \times Act \times S$.

[38] See Section 3.4.1, Definition 3.4.1 (page 54) and Definition 3.4.2 (page 54).

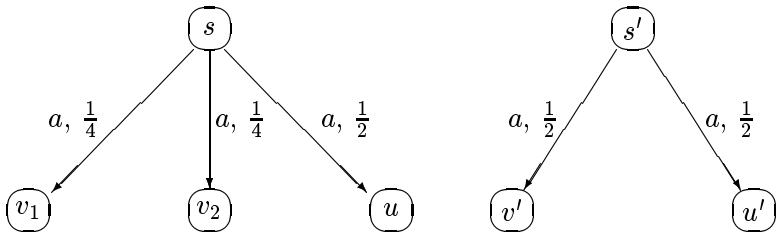


Figure 10.16:

Then, $\mathbf{P}(s,a,T) = \frac{1}{2} < 1 = \mathbf{P}(s',a,T)$. Nevertheless, $\sim = \bigcap_{i\geq 0} F_{\sim}^{i}(S\times S)$ is the greatest fixed point of $F_{\sim}$ and the iteration $Z_0 = S\times S$, $Z_{i+1} = F_{\sim}(Z_i)$, $i = 0,1,2,\ldots$, "converges" to $\sim$ (i.e. $Z_i = \sim$ for some i).[39] Thus, $\sim$ is the set associated with the boolean term[40] *gfp* Z [$\lambda s,s'$ [$\forall a$ $\forall$ t [*bexpr*]]] where *bexpr* is

$$\left(\sum_{t'} [\ Z(t,t') * P(s,a,t')\ ]\right) \quad = \quad \left(\sum_{t'} [\ Z(t,t') * P(s',a,t')\ ]\right).$$

In particular, the corresponding mixed term yields the BDD representation of $\sim$. Similar ideas can be used to compute the weak bisimulation equivalence classes of an action-labelled fully probabilistic system (cf. Chapter 7). For this, we can use the fact

$$\approx \ = \ gfp(F_{\approx}) \ = \ \bigcap_{i\geq 0} F_{\approx}^{i}(S\times S)$$

where the operator $F_{\approx} : 2^{S\times S} \to 2^{S\times S}$ is defined as follows. $F_{\approx}(Z)$ is the set of all pairs $(s,s') \in Z$ such that, for all $a \in Act$ and $t \in S$:

$$Prob(s, \tau^*\widehat{a}\tau^*, [t]_Z) \ = \ Prob(s', \tau^*\widehat{a}\tau^*, [t]_Z)$$

where $[t]_Z = \{t' \in S : (t,t') \in Z\}$. Using similar ideas as in Example 10.1.11 (page 267), we obtain algebraic terms $term^{Z}_{prob}$ that represent the function

$$S \times Act \times S \to [0,1], \quad (s,a,t) \mapsto Prob(s, \tau^*\widehat{a}\tau^*, [t]_Z).$$

Then, the semantics of the boolean term

$$lfp\ Z\ \left[\ \lambda s,s'\ \left[\ \forall a\ \forall t\ \left[\ term^{Z}_{prob}(s,a,t) = term^{Z}_{prob}(s',a,t)\ \right]\ \right]\ \right]$$

is $\approx$. Our algorithm applied to the corresponding mixed term successively generates the BDDs B_i for the relations $F_{\approx}^{i}(S\times S)$, $i = 0,1,2,\ldots$. For the computation of B_{i+1}, the algorithm calculates the MTBDD Q_i for $\mathsf{mixed}(term^{Z}_{prob})$ where the term variable Z is interpreted by (the characteristic function of) $F^i(S\times S)$ which is represented by the BDD B_i. The MTBDDs Q_i are computed by an iterative method (the procedure $\textsc{Iterate}_{\epsilon}(\cdot)$ of Figure 10.10, page 293). Thus, this method relies on repeated iterations to approximate the *real-valued* functions $(s,a,t) \mapsto Prob(s,\tau^*\widehat{a}\tau^*,[t]_i)$.[41] Alternatively,

[39] This is because $F_{\sim}$ restricted to an operator on equivalence relations on S is monotonic.

[40] Note that this term *gfp* Z [...] is *not* formally divergence free since the term variable Z is not formally continuous in $\lambda s,s'$[$\forall a$ $\forall t$[*bexpr*]] as there are free occurrences of Z within a subexpression of the form $expr_1\ op_{\bowtie}\ expr_2$. However, because of the above observation, the meaning of *gfp* Z[...] = *lim* Z[... $\uparrow$ $\lambda s,t$[1]] is the characteristic function of $\sim$.

[41] $[t]_i$ denotes the equivalence class of t with respect to $F^i(S\times S)$).

we can use the algebraic mu-calculus to describe (a variant of) the algorithm proposed in Section 7.2 (page 164 ff). The resulting MTBDD-based method just uses iterations on *boolean-valued* functions to compute least fixed points of monotonic set-based operators (rather than iterations on real-valued functions).[42] For this, we use a variant of the results presented in Chapter 7 where weak and branching bisimulation are shown to be the same and where an alternative characterization of branching bisimulation is used as basis for an algorithm to compute the branching (or weak) bisimulation equivalence classes. We define a monotonic operator G on the equivalence relations on S as follows. Let Z be an equivalence relation on S. Then,

$$G(Z) \;=\; \bigcap_{a,C} \{(s,s') : s \text{ and } s' \text{ are contained in the same block of } \mathit{Refine}(S/Z,a,C)\}$$

where (a,C) ranges over all pairs $(a,C) \in \mathit{Act} \times S/Z$ and where the operator $\mathit{Refine}(\cdot)$ is defined as in Notation 7.2.13 (page 168). The initial relation Z_{init} is those equivalence relation on S that identifies all divergent states (states that cannot reach a state where a visible action can be performed) and all non-divergent states, i.e.

$$Z_{init} \;=\; \mathit{Div} \times \mathit{Div} \;\cup\; (S \setminus \mathit{Div}) \times (S \setminus \mathit{Div})$$

where Div is the set of divergent states (see Definition 7.2.14, page 168). Then,

$$Z_{init} \;\supseteq G(Z_{init}) \;\supseteq\; G(G(Z_{init})) \;\supseteq\; \ldots \text{ and } \approx \;=\; G^i(Z_{init}) \text{ for some } i.$$

Thus, the BDD representation for $\approx$ can be obtained by evaluating the mixed term corresponding to the following boolean term.

$$\mathit{lim}\ Z\ [\ \lambda s,s'\ [\ G_Z(s,s')\ \uparrow\ \mathit{Init}\ ]\ ]$$

Init and G_Z are boolean terms that represent the sets Z_{init} and $G(Z)$. For the definition of *Init* and G_Z we use the notations (i.e. expressions of the form $\mathit{term}(\kappa_1,\ldots,\kappa_n)$) explained on page 267. Moreover, we use expressions like "$a \neq \tau$" (where a is an individual variable and $\tau \in \mathit{Act}$ the symbol for the internal action) to denote the boolean term $\neg E_\tau(a)$ where E_τ is an 1-ary function symbol that represents the singleton set $\{\tau\}$. For the definition of *Init* we use the following fact. $S \setminus \mathit{Div}$ is the least set $Y \subseteq S$ that contains

$$\mathit{Vis} \;=\; \{s \in S : \mathbf{P}(s,\alpha) > 0 \text{ for some } \alpha \in \mathit{Act} \setminus \{\tau\}\ \}$$

(the set of states where a visible action can be performed) and, whenever $t \in Y$, $a \in \mathit{Act}$ and $\mathbf{P}(s,a,t) > 0$ then $s \in Y$. We define *Init* by

$$\mathit{Init} \;=\; \lambda s,s'\ [\ \mathit{Vis}^*(s) \;\leftrightarrow\; \mathit{Vis}^*(s')\ ]$$

where $\mathit{Vis} \;=\; \lambda s[\ \exists a\ \exists t\ [(a \neq \tau)\ \wedge\ (P(s,a,t) > 0)\ ]\ ]$ and

$$\mathit{Vis}^* \;=\; \mathit{lfp}\ Y\ [\ \lambda s\ [\ \mathit{Vis}(s)\ \vee\ \exists a\ \exists t\ [\ Y(t) \wedge (P(s,a,t) > 0)\ ]\ ].$$

The definition of G_Z relies on the following observation.[43] Let Z be an equivalence relation on S. We define $\mathbf{P}_Z(s,a,t) = \mathbf{P}(s,a,[t]_Z)$ and $S_Z = \{s \in S : \mathbf{P}(s,\tau,[s]_Z) < 1\}$.[44] Let

[42] We might expect that the latter return the correct results (sets) and are much faster than the former (which return approximations rather than the exact values).

[43] The correctness of this observation can be easily derived from the results of Section 7.2 (page 164 ff).

[44] $[t]_Z$ is the equivalence class of t with respect to Z, i.e. $[t]_Z = \{t' : (t,t') \in Z\}$.

$Split_Z$ be the set of pairs $(s,s') \in Z$ where $s,\ s' \in S_Z$ and, for all $a \in Act$ and $t \in S$: if $a \neq \tau$ or $(s,t) \notin Z$ then

$$\frac{\mathbf{P}_Z(s,a,t)}{1-\mathbf{P}_Z(s,\tau,s)} \;=\; \frac{\mathbf{P}_Z(s',a,t)}{1-\mathbf{P}_Z(s',\tau,s')}.$$

A_Z denotes the relation consisting of those pairs $(s,t) \in Z$ where either $(s,t) \in Split_Z$ or there exists a finite path $\sigma = s_0 \stackrel{\tau}{\to} s_1 \stackrel{\tau}{\to} \ldots \stackrel{\tau}{\to} s_k$ such that $k \geq 1$, $s_0 = s$, $s_i \in S \setminus S_Z$, $i = 0,\ldots,k-1$ and $(s_k,t) \in Split_Z$. Alternatively, A_Z can be described as the least fixed point of the operator $H : 2^{S\times S} \to 2^{S\times S}$,

$$H(X) \;=\; Split_Z \cup \{\ (s,t)\ :\ s \notin S_Z\ \wedge\ \exists u \in S[(\mathbf{P}(s,\tau,u) > 0) \wedge (u,t) \in X]\ \}.$$

B_Z is the set of pairs $(s,t) \in A_Z$ such that $(s,t') \in A_Z$ implies $(t,t') \in Split_Z$. It is easy to see that $(s,t) \in B_Z$ iff $t \in S_Z$ and s, t belong to the same block of $Refine(S/Z,a,C)$ for any $(a,C) \in Act \times S/Z$. Thus, if Res_Z is the set of all states $s \in S$ where there is no $t \in S$ with $(s,t) \in B_Z$, then

$$G(Z) \;=\; \{\ (s,s') \in Z\ :\ s,s' \in Res_Z\ \vee\ \exists t\ [\ (s,t),\ (s',t) \in B_Z\ ]\ \}.$$

From this, we derive the definition of the algebraic term G_Z. We define

$$G_Z \;=\; \lambda s,s'\ [\ Z(s,s') \wedge (\ (Res_Z(s) \wedge Res_Z(s'))\ \vee\ \exists t\ [\ B_Z(s,t) \vee B_Z(s',t)\ ]\ )\ ]$$

where we use the following auxiliary algebraic (or boolean) terms.

$$\begin{aligned}
P_Z &= \lambda s,a,t\ [\ \textstyle\sum_u [\ Z(t,u) * P(s,a,u)\ ]\ ],\\
P'_Z &= \lambda s,a,t\ [\ P_Z(s,a,t)\ \%\ (1-P_Z(s,\tau,s))\ ],\\
S_Z &= \lambda s[\ P_Z(s,\tau,s) < 1\ ],\\
Split_Z &= \lambda s,s'\ [\ Z(s,s')\ \wedge\ S_Z(s)\ \wedge\ S_Z(s')\ \wedge\\
&\qquad\qquad \forall a\ \forall t\ [\ ((a \neq \tau) \vee \neg Z(s,t))\ \to\ (P'_Z(s,a,t) = P'_Z(s',a,t))\ ]\ ],\\
A_Z &= lfp\ X\ [\ \lambda s,t\ [\ \ Split_Z(s,t)\ \vee\ (\neg S_Z(s) \wedge\ \exists u[\ (P(s,\tau,u) > 0) \wedge X(u,t)\ ]\ ]\ ],\\
B_Z &= \lambda s,t\ [\ A_Z(s,t)\ \wedge\ \forall t'[A_Z(s,t') \to Split_Z(t,t')]\ ],\\
Res_Z &= \lambda s\ [\ \neg\exists t[\ B_Z(s,t)\ ]\ ].
\end{aligned}$$

Chapter 11

Concluding remarks

In summary, when the author started to work on probabilistic systems in the end of 1995, a lot of excellent research had already been done in this field. In this thesis, she tried to fill a few gaps but there are still a variety of interesting open questions. Among those that are closely related to the topics of this thesis we mention just a few.

Only a few research has been done so far in the development of algorithmic methods for establishing an implementation relation between two systems; in particular, the literature (still) lacks for algorithms that can check a *weak* equivalence for concurrent probabilistic systems (e.g. weak or branching bisimulation à la Segala & Lynch [SeLy94]). This is most important for the mechanised design and the system analysis since the weak relations are those that are needed to compare a "high-level" system (the specification) and a "low-level" system (the implementation) and that play a crucial role to reduce the state space by abstraction. Recent work by Philippou, Sokolsky & Lee [PSS98] and by Stoelinga, Vaandrager and the author [BSV98] are first attempts in this direction. [PSS98] present an algorithm for deciding weak bisimulation equivalence for stratified systems which uses an alternative characterization of weak bisimulation equivalence by means of the minimal/maximal probabilities of certain events under all adversaries. Stoelinga & Vaandrager [StVa98] propose to adapt the concept of *normed simulations* [GriVa98] for the probabilistic setting thus yielding a quite simple characterization of branching simulations. It seems to be the case that this characterization can serve as the basis of an algorithm for checking whether two concurrent probabilistic systems are branching (bi-)similar [BSV98].

An open problem that concerns the design of probabilistic systems is the question whether *satisfiability* of *PCTL* (or the full logic $PCTL^*$) is decidable with respect to any of the satisfaction relations. Possibly, a decision procedure for satisfiability might serve as a basis for an *automatic synthesis* of probabilistic processes fulfilling a given specification in the form of a satisfiable *PCTL* formula (as it is the case for non-probabilistic systems [EmCl82, MaWo84, AtEm89, PnRo89]).

Even though algorithmic verification for establishing qualitative or quantitive properties for concurrent systems are known, for realistic applications, the completeness for double exponential time for *LTL* model checking in the concurrent probabilistic case (shown by Courcoubetis & Yannakakis [CoYa95]) seems to be fatal. In [BKN98], we propose a method for computing lower and upper bounds for the values $p_s^{max}(\varphi)$ and $p_s^{min}(\varphi)$. This method is based on a Greedy algorithm and runs in single exponential time and needs

polynomial space. In the worst case, the obtained bounds might be far away from the precise values; e.g. it is possible to obtain 0 as lower and 1 as upper bound. The quality of this method has still to be worked out on the basis of experimental results. It would be desirable to have efficient methods (PTASs) that *approximate* the values $p_s^{max}(\varphi)$ and $p_s^{min}(\varphi)$.

Most research that has been done so far in the field of methods that attack the state explosion problem for probabilistic systems concerns the MTBDD approach [CFM+93, HMP+94, BCH+97, HarG98]. It would be interesting whether the *partial order reduction* techniques [Pele93, Valm94, Gode94] can be modified for (concurrent) probabilistic systems. Recent work by Biere et al [BCC+98] shows that, for non-probabilistic systems, symbolic *LTL model checking* is also possible *without* BDDs, using a reduction to the satisfiability problem for propositional logic. It would be very interesting whether similar ideas (e.g. using arithmetic equations rather than propositional formulas) are applicable for probabilistic systems.

Another interesting point is the investigation of probabilistic systems with an *infinite state space*. First attempts in this direction are the investigation of *probabilistic lossy channel systems* (PLCSs); see [IyNa97] where an appoximative method for verifying quantitative properties is proposed and [BaEn98] where it is shown that qualitative *LTL* model checking for PLCSs can be reduced to a reachability problem in the underlying non-probabilistic LCS (and is solvable with the methods of [AbJo93]).

Chapter 12

Appendix

In this chapter we recall some definitions and methods of the literature. Section 12.1 summarizes the necessary background that we need for the denotational semantics in Chapter 5. Section 12.2 briefly explains our notations concerning ordered balanced trees, Section 12.3 recalls the definition of MTBDDs. The notations introduced in one of the Sections 12.1, 12.2 or 12.3 are not used without a references to the relevant parts of this chapter.

12.1 Mathematical preliminaries for the denotational models

In Chapter 5 we use the standard procedure to give denotational semantics in the metric and partial order approach and the *probabilistic powerdomain of evaluations*. Here, we need some basic notions of domain theory, the theory of metric spaces and categorical methods for solving recursive domain equations. These are briefly summarized in Sections 12.1.1, 12.1.2 and 12.1.3. In Section 12.1.4, we briefly recall the notion of an "evaluation" on a topological space as introduced by Jones & Plotkin [JoPl89, Jone90].

12.1.1 Basic notions of domain theory

We briefly recall some basic notions of domain theory and explain our notations. Further details can be found e.g. in [GHK$^+$80, AbJu94, SLG94].

Preorders and partial orders: A *preorder* on a set D is a binary relation on D which is reflexive and transitive. A *poset* is a pair $(D, \sqsubseteq)$ consisting of a set D and a *partial order* $\sqsubseteq$ on D (i.e. $\sqsubseteq$ is an antisymmetric preorder on D). We often write D instead of $(D, \sqsubseteq)$. If nothing else is said then the underlying partial order of a poset D is denoted by $\sqsubseteq_D$ or shortly $\sqsubseteq$. A *pointed poset* is a poset D which has a *bottom element* (denoted by $\bot_D$ or shortly $\bot$), i.e. $\bot \sqsubseteq x$ for all $x \in D$. If D is a poset and $x \in D$ then we put $x\!\downarrow = \{y \in D : y \sqsubseteq x\}$ and $x\!\uparrow = \{y \in D : x \sqsubseteq y\}$. Let X be a subset of a poset D. We put $X\!\downarrow = \bigcup_{x\in X} x\!\downarrow$ and $X\!\uparrow = \bigcup_{x\in X} x\!\uparrow$. X is called *leftclosed* or *downwardclosed* iff X is nonempty and $X\!\downarrow = X$. Similarly, X is called *rightclosed* or *upwardclosed* iff X is

nonempty and $X\uparrow = X$. An element $x_0 \in D$ is called an *upper bound* of X iff $x \sqsubseteq x_0$ for all $x \in X$. x_0 is called the *least upper bound* of X iff x_0 is an upper bound of X such that $x_0 \sqsubseteq y$ for each upper bound y of X. The least upper bound of X (if it exists) is denoted by $\bigsqcup X$ or $lub(X)$. X is called *directed* iff every pair of elements in X has an upper bound.

Dcpo's: A pointed poset in which each directed subset X has a least upper bound is called a *directed-complete* partial order (shortly *dcpo*).[1] A *ω-chain* in a dcpo D is an infinite monotone sequence in D, i.e. a sequence $(x_n)_{n\geq 0}$ in D such that $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$. For $(x_n)_{n\geq 0}$ to be a ω-chain, we write $\bigsqcup_{n\geq 0} x_n$ or briefly $\bigsqcup x_n$ to denote the least upper bound of $\{x_n : n \geq 0\}$.

d-continuity and strictness: Let D, D' be dcpo's and $f : D \to D'$ a function. f is called *monotone* iff $x \sqsubseteq_D y$ implies $f(x) \sqsubseteq_{D'} f(y)$. f is called *d-continuous* iff, for each directed subset X of D, $f(\bigsqcup X) = \bigsqcup f(X)$. (In particular, if f is d-continuous then f is monotone.) f is called *strict* iff $f(\bot_D) = \bot_{D'}$.

Tarski's fixed point theorem:[2] Whenever D is a dcpo and $f : D \to D$ a d-continuous function then f has a least fixed point $lfp(f)$. Moreover, $lfp(f) = \bigsqcup f^n(\bot)$.

Scott-Topology: A subset A of a dcpo D is called *lub-closed* iff for every directed subset X of A we have $\bigsqcup X \in A$. We always suppose a dcpo D to be equipped with the *Scott-topology* whose closed sets are the downward-closed and lub-closed subsets of D. For A to be a nonempty subset of D, A^{cl} denotes the Scott-closure of A, i.e. the smallest Scott-closed subset containing A. We define $\emptyset^{cl} = \{\bot\}$. Then, for A to be finite and nonempty, $A^{cl} = A\downarrow$.

Hoare powerdomain: If D is a dcpo then $Pow_{Hoare}(D)$ is the dcpo of nonempty and Scott-closed subsets of D ordered by inclusion.

Continuous domains: Let x, y be elements of a dcpo D. We say y *approximates* x iff for all directed subsets X of D, $x \sqsubseteq \bigsqcup X$ implies $y \sqsubseteq z$ for some $z \in X$. $Approx(x)$ denotes the set of elements $y \in D$ such that y approximates x. A *basis* of a dcpo D is a subset B of D such that for each $x \in D$ the set $B \cap Approx(x)$ contains a directed subset with least upper bound x. A *continuous domain* is a dcpo which has a basis.

Function spaces: If X is a set and D a dcpo then the function space $X \to D$ (of all functions $f : X \to D$) is supposed to be equipped with the partial order $f_1 \sqsubseteq f_2$ iff $f_1(x) \sqsubseteq_D f_2(x)$ for all $x \in D$. Note that $X \to D$ is again a dcpo whose bottom element is the function $X \to D$, $x \mapsto \bot_D$ and where, for each directed set Ξ of functions $f : X \to D$, the least upper bound $\bigsqcup \Xi$ is given by: $(\bigsqcup \Xi)(x) = \bigsqcup\{f(x) : f \in \Xi\}$. In particular, if (f_n) is a ω-chain in $X \to D$ then $(\bigsqcup f_n)(x) = \bigsqcup f_n(x)$.

The function space $X \to [a,b]$: Clearly, any compact interval $[a,b]$ of real numbers (where $a < b$) equipped with the natural order $\leq$ is a dcpo. We use the symbols "sup" or "inf" to denote least upper bounds (suprema) or greatest lower bounds (infima) in $[a,b]$ which exist for all nonempty subsets of $[a,b]$ or sequences in $[a,b]$. We consider the function

[1] Note that, in contrast to the notions used in [AbJu94] and several other authors, we require a dcpo to have a bottom element.

[2] Several authors use different names for this theorem. For the history of this theorem and the question by whom it should be named (Tarski, Kleene or Knaster) see [LNS82].

space $X \to [a,b]$ (where X is an arbitrary nonempty set) equipped with the induced order, also denoted by $\leq$, as explained before, i.e. $f \leq f'$ iff $f(x) \leq f'(x)$ for all $x \in [a,b]$. If $\Xi = \{f_i : i \in I\}$ is a nonempty family of functions $f : X \to [a,b]$ then $\sup_{f \in \Xi} f$ (or $\sup_{i \in I} f_i$ or briefly $\sup f_i$) denotes the function $X \to [a,b]$, $x \mapsto \sup_{f \in \Xi} f(x)$. Similarly, $\inf_{f \in \Xi} f$ (or $\inf_{i \in I} f_i$ or briefly $\inf f_i$) denotes the function $X \to [a,b]$, $x \mapsto \inf_{f \in \Xi} f(x)$. We say that a function $F : (X \to [a,b]) \to (X \to [a,b])$ preserves suprema iff, for all nonempty sets Ξ of functions $f : X \to [a,b]$,

$$F\left(\sup_{f \in \Xi} f\right) = \sup_{f \in \Xi} F(f).$$

Similarly, F preserves infima iff, for all nonempty sets Ξ of functions $f : X \to [a,b]$, $F(\inf_{f \in \Xi} f) = \inf_{f \in \Xi} F(f)$.

Proposition 12.1.1 *Let $F : (X \to [a,b]) \to (X \to [a,b])$ be a monotone operator. Then, F has a greatest fixed point $gfp(F)$ and a least fixed point $lfp(F)$. $gfp(F)$ and $lfp(F)$ are given by*

$$gfp(F) = \sup_{f \in \Xi^F_{\leq}} f, \quad lfp(F) = \inf_{f \in \Xi^F_{\geq}} f$$

where $\Xi^F_{\leq} = \{f : X \to [a,b] : f \leq F(f)\}$, $\Xi^F_{\geq} = \{f : X \to [a,b] : f \geq F(f)\}$. If F preserves infima then

$$gfp(F) = \inf_{n \geq 0} F^n(f_b)$$

where $f_b(x) = b$ for all $x \in X$. Similarly, if F preserves suprema then $lfp(F) = \sup_{n \geq 0} F^n(f_a)$ where $f_a(x) = a$ for all $x \in X$.

Proof: easy verification. ■

Remark 12.1.2 For higher-order operators with more than one (function) arguments, e.g. operators whose arguments are pairs $\langle f, g \rangle$ where $f : X \to [a,b]$ and $g : Y \to [c,d]$ are functions, monotonicity is not a sufficient condition for the existence of least/greatest fixed points.[3] Nevertheless, if the operator preserves infima (resp. suprema) then greatest (resp. least) fixed points exists and an analogue to the second part of Proposition 12.1.1 holds. Formally, let $k \geq 2$, $X_1, \ldots, X_k$ nonempty sets and $a_1, \ldots, a_k$, $b_1, \ldots, b_k$ real numbers such that $a_i < b_i$. Let D the set of k-tuples $\langle f_1, \ldots, f_k \rangle$ where $f_i : X_i \to [a_i, b_i]$ is a function. The partial order $\leq$ on D is given by

$$\langle f_1, \ldots, f_k \rangle \leq \langle g_1, \ldots, g_k \rangle \text{ iff } f_i \leq g_i, i = 1, \ldots, k.$$

Let $F : D \to D$ be an operator that preserves suprema, i.e. whenever Ξ_i are nonempty families of functions $X_i \to [a_i, b_i]$ and $f_i^+ = \sup_{f \in \Xi_i} f$ then

$$\sup \{F(f_1, \ldots, f_k) : f_i \in \Xi_i, i = 1, \ldots, k\} = F(f_1^+, \ldots, f_k^+).$$

Then, $lfp(F)$ exists and equals $\sup_{n \geq 0} F^n(f_1, \ldots, f_k)$ where f_i denotes the function $f_i : X_i \to [a_i, b_i]$, $f_i(x) = a_i$ for all $x \in X_i$. Similarly, if F preserves infima then $gfp(F)$ exists and can be obtained by iteration. ■

[3] This is because suprema or infima of arbitrary sets of tuples of function might not exist.

12.1.2 Metric spaces

Basic notions concerning metric spaces can be found in any standard book about topology, see e.g. [Dugu66, Suth77, Enge89]. We briefly recall the definitions that we are used in that thesis and explain our notations.

Metric and ultrametric spaces: A *metric* on a set M is a function $d : M \times M \to [0,1]$ such that, for all x, y, $z \in M$,

$$d(x,y) = d(y,x),\ d(x,y) = 0 \text{ iff } x = y,\ d(x,z) \leq d(x,y) + d(y,z).$$

A metric d is called an *ultrametric* iff, for all x, y, $z \in M$, $d(x,z) \leq \max\{d(x,y), d(y,z)\}$. An *(ultra-)metric space* is a pair (M,d) consisting of a set M and an (ultra-)metric d on M. We often write M rather than (M,d) and refer to d as the *distance* on M. We always suppose that the underlying distance on a metric space M – which we always denote by d_M or shortly d – satisfies $d \leq 1$. In what follows, let M, M' be metric spaces.

Non-expansive and contracting functions and embeddings: Let $f : M \to M'$ be a function. f is called *non-expansive* iff $d_{M'}(f(x), f(y)) \leq d_M(x,y)$ for all x, $y \in M$. f is called *contracting* iff there exists a real number C with $0 < C < 1$ such that $d_{M'}(f(x), f(y)) \leq C \cdot d_M(x,y)$ for all x, $y \in M$. In that case, C is called a *contraction coefficient* of f. f is called an *embedding* iff $d_{M'}(f(x), f(y)) = d_M(x,y)$ for all x, $y \in M$.

Topology of open balls: The topology on M is given by taking the open balls $B(x,\rho)$, $x \in M$, $\rho > 0$, as its basic opens. Here, the *open ball* $B(x,\rho)$ with center x and radius ρ is defined by $B(x,\rho) = \{y \in M : d(x,y) < \rho\}$. $\overline{B}(x,\rho) = \{y \in M : d(x,y) \leq \rho\}$ is called the *closed ball* with center x and radius ρ. $Balls(M)$ denotes the set of all open balls, $Balls_\rho(M)$ the set of open balls with radius $\geq \rho$, i.e. open balls of the form $B(x,r)$ where $r \geq \rho$. By a *ρ-set*, we mean an open set $U \subseteq M$ such that $B(x,\rho) \subseteq U$ for all $x \in U$.

Cauchy sequences, limits and density: A *Cauchy sequence* in M is an infinite sequence $(x_n)_{n\geq 0}$ in M such that for each $\epsilon > 0$ there exists $N \geq 0$ with $d(x_n, x_m) \leq \epsilon$ for all n, $m \geq N$. If (x_n) is a sequence in M and $x \in M$ then x is called the limit of (x_n) (denoted by $\lim_{n\to\infty} x_n$ or shortly $\lim x_n$) iff for each $\epsilon > 0$ there exists $N \geq 0$ with $d(x_n, x) \leq \epsilon$ for all $n \geq N$. As in standard analysis, if $\lim x_n$ exists then we say (x_n) is converging or (x_n) converges to x. A subset X of M is called *dense* in M iff, for each $x \in X$, there is a converging sequence $(x_n)_{n\geq 0}$ in X such that $x = \lim x_n$.

Completeness: M is called *complete* iff each Cauchy sequence in M has a limit. A *completion* of a metric space M is a pair (M', e) consisting of a complete metric space M' and an embedding $e : M \to M'$ such that $e(M)$ is dense in M'. If e is understood from the context (or not of interest) then we briefly say that M' is a completion of M.

Banach's fixed point theorem: Each contracting function $f : M \to M$ on a complete metric space M has a unique fixed point $\mathit{fix}(f)$. Moreover, for each $x \in M$, $(f^n(x))$ is a Cauchy sequence with $\mathit{fix}(f) = \lim f^n(x)$.

Function spaces: If X is a set and M complete then the function space $X \to M$ equipped with the distance

$$d(f_1, f_2) \;=\; \sup_{x\in M} d_M(f_1(x), f_2(x))$$

is also a complete metric space. For each Cauchy sequence (f_n) in $X \to M$, the limit $\lim f_n : X \to M$ is given by $(\lim f_n)(x) = \lim f_n(x)$.

The powerdomain $Pow_{comp}(M)$: A subset X of M is called *compact* iff each infinite sequence in X contains a convergent subsequence whose limit belongs to X. $Pow_{comp}(M)$ denotes the collection of compact subsets of M. If M is complete then $Pow_{comp}(M)$ equipped with the *Hausdorff metric*

$$d(X,Y) = \max \left\{ \sup_{x \in X} \inf_{y \in Y} d(x,y), \sup_{y \in Y} \inf_{x \in X} d(x,y) \right\}$$

is a complete metric space (see [Kura56]).

12.1.3 Categorical methods for solving domain equations

We briefly explain the methods of Rutten & Turi [RuTu93] and Abramsky & Jung [AbJu94] for solving recursive domain equations for metric spaces or dcpo's. We refer the interested reader to [SmPl82, MajC88, AmRu89, MajC89, MaZe91, EdSm92, Barr93] for more informations about how to solve recursive domain equations. For the definition of categories and functors (and other related notions) see e.g. [McLan71, AHS90, BaWe90].

Coalgebras and fixed points of functors: Let *Cat* be a category and $\mathcal{F} : Cat \to Cat$ a functor. A *coalgebra* of $\mathcal{F}$ is a pair (X, e) consisting of an object X of *Cat* and a morphism $e : X \to \mathcal{F}(X)$ in *Cat*. A coalgebra (X, e) of $\mathcal{F}$ is called *final* iff it is a final object in the category of all coalgebras, i.e. iff for each coalgebra (X', e') of $\mathcal{F}$ there exists a unique morphism $f : X' \to X$ in *Cat* with $\mathcal{F}(f) \circ e' = e \circ f$. A *fixed point* of $\mathcal{F}$ is a coalgebra (X, e) of $\mathcal{F}$ such that e is an isomorphism in *Cat*. A fixed point (X, e) of $\mathcal{F}$ is called *final* iff for each fixed point (X', e') of $\mathcal{F}$ there exists a unique morphism $f : X' \to X$ in *Cat* with $\mathcal{F}(f) \circ e' = e \circ f$. Final coalgebras of $\mathcal{F}$ are always final fixed points (see e.g. [RuTu93]). A fixed point (X, e) of $\mathcal{F}$ is called *initial* iff for each fixed point (X', e') of $\mathcal{F}$ there exists a unique morphism $f : X \to X'$ in *Cat* with $\mathcal{F}(f) \circ e = e' \circ f$. We say (X, e) is the *unique* fixed point of $\mathcal{F}$ iff (X, e) is a fixed of $\mathcal{F}$ and for each fixed point (X', e') of $\mathcal{F}$ there exists a unique isomorphism $f : X \to X'$ in *Cat* with $\mathcal{F}(f) \circ e = e' \circ f$. If the underlying (iso-)morphism e of a coalgebra or fixed point is clear from the context or not of interest then we shortly write X instead of (X, e).

Categories used in that thesis:

SET denotes category of sets and functions,
CUM the category of complete ultrametric spaces and non-expansive functions,
$CONT_\perp$ the category of continuous domains and strict, d-continuous functions.

Categorical methods for solving recursive domain equations: A functor $\mathcal{F} : CONT_\perp \to CONT_\perp$ is called *locally d-continuous* if, for all continuous domains D, D', the function $(D \longrightarrow_{\text{strict \& dcont}} D') \to (\mathcal{F}(D) \longrightarrow_{\text{strict \& dcont}} \mathcal{F}(D'))$, $f \mapsto \mathcal{F}(f)$, is d-continuous. Here, $D \longrightarrow_{\text{strict \& dcont}} D'$ denotes the set of strict and d-continuous functions from D to D' (i.e. the set of $CONT_\perp$-morphism from D to D'). Clearly, the composition of locally d-continuous functors $CONT_\perp \to CONT_\perp$ is locally d-continuous. As shown in [AbJu94], each locally d-continuous functor $\mathcal{F} : CONT_\perp \to CONT_\perp$ has an initial fixed point.

Let $\mathcal{F} : CUM \to CUM$ be a functor. For M, M' to be complete ultrametric spaces, $M \longrightarrow_{\text{nexp}} M'$ denotes the set of non-expansive functions $M \to M'$, i.e. the set of CUM-morphism from M to M'. $\mathcal{F}$ is called *locally contracting* iff there exists a real number C with $0 \leq C < 1$ such that $d_{\mathcal{F}(M')}(\mathcal{F}(f_1), \mathcal{F}(f_2)) \leq C \cdot d_{M'}(f_1, f_2)$ for all complete ultrametric spaces M, M' and all non-expansive functions $f_i : M \to M'$, $i = 1, 2$, i.e. iff the function $(M \longrightarrow_{\text{nexp}} M') \to (\mathcal{F}(M) \longrightarrow_{\text{nexp}} \mathcal{F}(M'))$, $f \mapsto \mathcal{F}(f)$, is contracting with contracting constant C. Similarly, $\mathcal{F}$ is called *locally non-expansive* iff $d_{\mathcal{F}(M')}(\mathcal{F}(f_1), \mathcal{F}(f_2)) \leq d_{M'}(f_1, f_2)$ for all complete ultrametric spaces M, M' and all non-expansive functions $f_i : M \to M'$, $i = 1, 2$. Clearly, if $\mathcal{F}_i : CUM \to CUM$ are locally non-expansive functors, $i = 1, 2, 3$, such that at least one of them is locally contracting then the composition $\mathcal{F}_1 \circ \mathcal{F}_2 \circ \mathcal{F}_3$ is locally contracting. As shown in [RuTu93], each locally contracting functor $\mathcal{F} : CUM \to CUM$ has a unique fixed point.[4]

Functors used in that thesis:

- **Powerdomain functors:** The functors $Pow_{fin} : SET \to SET$, $Pow_{Hoare} : CONT_\perp \to CONT_\perp$ and $Pow_{comp} : CUM \to CUM$ are defined as follows. If X is a set then $Pow_{fin}(X)$ denotes the set of finite subsets of X. If $f : X \to Y$ is a function then we define $Pow_{fin}(f) : Pow_{fin}(X) \to Pow_{fin}(Y)$ by $Pow_{fin}(f)(U) = f(U)$. The definitions of the Hoare powerdomain $Pow_{Hoare}(D)$ for a dcpo D and and the powerdomain $Pow_{comp}(M)$ of compact subsets of an ultrametric space M are given on page 308 and page 311 respectively. If D, D' are continuous domains and $f : D \to D'$ is strict and d-continuous then $Pow_{Hoare}(f) : Pow_{Hoare}(D) \to Pow_{Hoare}(D')$ is given by $Pow_{Hoare}(f)(A) = f(A)^{cl}$. If M, M' are complete ultrametric spaces and $f : M \to M'$ is a non-expansive function then $Pow_{comp}(f) : Pow_{comp}(M) \to Pow_{comp}(M')$ is given by $Pow_{comp}(f)(X) = f(X)$.
- **The functors $\mathcal{F}_A^*$:** If A is a set then $\mathcal{F}_A : SET \to SET$ is defined as follows: $\mathcal{F}_A(X) = A \times X$ and, for $f : X \to Y$ to be a function, $\mathcal{F}_A(f)(a, x) = (a, f(x))$ for all $a \in A$ and $x \in X$. We extend $\mathcal{F}_A$ to endofunctors of CUM and $CONT_\perp$ (called $\mathcal{F}_A^{cum}$ and $\mathcal{F}_A^{cont}$ respectively) as follows.
  - $\mathcal{F}_A^{cum}(M) = A \times M$ where the ultrametric on $A \times M$ is given by:

$$d((a,x),(b,y)) = \begin{cases} 1 & : \text{ if } a \neq b \\ \frac{1}{2} \cdot d_M(x,y) & : \text{ otherwise (i.e. if } a = b). \end{cases}$$

    For $f : M \to M'$ to be a morphism in CUM, we define $\mathcal{F}_A^{cum}(f) = \mathcal{F}_A(f)$.
  - $\mathcal{F}_A^{cont}(D) = \{\perp\} \uplus A \times D$ where $\uplus$ denotes disjoint union and where the partial order on $\{\perp\} \uplus A \times D$ is given by: $\perp \sqsubseteq (a, x)$ for all $(a, x) \in A \times D$ and $(a, x) \sqsubseteq (b, y)$ iff $a = b$ and $x \sqsubseteq_D y$. For $f : D \to D'$ to be a morphism in $CONT_\perp$, we define $\mathcal{F}_A^{cont}(f)(\perp) = \perp$ and $\mathcal{F}_A^{cont}(f)(a, x) = (a, f(x))$ for all $(a, x) \in A \times D$.
- **The distribution functor:** $Distr$ can be viewed as an endofunctor of SET where, for $f : X \to Y$ to be a function, the function $Distr(f) : Distr(X) \to Distr(Y)$ is given by $Distr(f)(\mu)(y) = \mu[\ f^{-1}(y)\ ]$ (cf. Section 2.2, page 31).

[4] To be precise, [RuTu93] deals with the category CMS of complete metric spaces and non-expansive functions instead of the subcategory CUM. However, it is easy to see that the fixed point theorem of [RuTu93] carries over to the category CUM.

It is easy to see that the functors Pow_* and $\mathcal{F}_A^*$ are well-defined and that Pow_{Hoare} and $\mathcal{F}_A^{cont}$ are locally d-continuous, Pow_{comp} is locally non-expansive while $\mathcal{F}_A^{cum}$ is locally contracting.

12.1.4 Evaluations

We recall the definition of evaluations on topological spaces as introduced by Jones & Plotkin [JoPl89, Jone90].

Evaluations (cf. [JoPl89, Jone90]): For X to be a topological space, $Opens(X)$ denotes the set of open sets in X. A function $E : Opens(X) \to [0,1]$ is called an *evaluation*[5] iff the following three conditions are satisfied:

1. If $(U_i)_{i\in I}$ is a directed family of open sets U_i in X (i.e. $(U_i)_{i\in I}$ is a family in $Opens(X)$ such that for all i, $j \in I$ there exists $k \in I$ with $U_i \subseteq U_k$ and $U_j \subseteq U_k$) then
$$E\left(\bigcup_{i\in I} U_i\right) = \sup_{i\in I} E(U_i).$$
2. $E(U \cap U') + E(U \cup U') = E(U) + E(U')$
3. $E(X) = 1$

The *probabilistic powerdomain of evaluations* $Eval(X)$ of a topological space X is the set of evaluations on X. Clearly, for each evaluation $E \in Eval(X)$, $E(\emptyset) = 0$, and, whenever U, $U' \in Opens(X)$ with $U \subseteq U'$ then $E(U) \leq E(U')$. We extend evaluations to closed subsets of X where we put $E(A) = 1 - E(X \setminus A)$ for each closed subset A of X.

The function $Eval(f)$: If X, X' are topological spaces and $f : X \to X'$ is a continuous function then $Eval(f) : Eval(X) \to Eval(X')$ is defined by $Eval(f)(E)(U) = E(f^{-1}(U))$. Thus, *Eval* can be considered as a functor $TOP \to SET$ where TOP denotes the category of topological spaces and continuous functions.

The evaluation E_μ for a distribution μ: If $\mu \in Distr(X)$ then
$$E_\mu : Opens(X) \to [0,1],\ E_\mu(U) = \mu[U].$$
is an evaluation on X. Whether the function $Distr(X) \to Eval(X)$, $\mu \mapsto E_\mu$, is injective (and hence can be considered as an embedding) depends on the underlying topology on X. Consider the topology $\{\emptyset, X\}$ on a set X which contains at least two points; it is easy to see that this function is not injective. In our applications – where X is equipped with an ultrametric or a directed-complete partial order – $Distr(X)$ can be considered as a subspace of $Eval(X)$ (cf. Theorem 5.1.12, page 95, and Theorem 5.1.16, page 97).

Remark 12.1.3 Let $eval_X : Distr(X) \to Eval(X)$ be the function $eval_X(\mu) = E_\mu$. It is easy to see that $eval_Y \circ Distr(f) = Eval(f) \circ eval_X$ for every function $f : X \to Y$. I.e. for each distribution $\mu \in Distr(X)$,
$$E_{Distr(f)(\mu)} = Eval(f)(E_\mu).$$

[5] An evaluation in our sense is a probabilistic continuous evaluation in the terminology of Jones & Plotkin [JoPl89].

Hence, *eval* is a natural transformation $Distr \to Eval$ where $Distr$ is considered as a functor $SET \to TOP$ (where $Distr(X)$ is supposed to be equipped with the discrete topology). ■

Composition of evaluations (cf. [Heck95]): If X and Y are topological spaces and $E_X \in Eval(X)$, $E_Y \in Eval(Y)$ then $E_X * E_Y$ denotes the unique evaluation on the product space $X \times Y$ such that $(E_X * E_Y)(U \times V) = E_X(U) \cdot E_Y(V)$ for all $U \in Opens(X)$ and $V \in Opens(Y)$. Note that, if $\mu \in Distr(X)$, $\nu \in Distr(Y)$ then $E_\mu * E_\nu = E_{\mu * \nu}$ (where the definition of $\mu * \nu$ was given on page 30).

The probabilistic powerdomain of evaluations on dcpo's: Recall that we suppose a dcpo D to be equipped with the Scott-topology, i.e. $Opens(D)$ consists of all subsets U of D where U is upward-closed and $D \setminus U$ is lub-closed. If D is a dcpo then $Eval(D)$ is a dcpo where the partial order $\sqsubseteq$ on $Eval(D)$ is given by

$$E_1 \sqsubseteq E_2 \text{ iff } E_1(U) \leq E_2(U) \text{ for all } U \in Opens(D)$$

(cf. [JoPl89]). The bottom element $\bot_{Eval(D)}$ of $Eval(D)$ is $E_{\mu^1_{\bot_D}}$ (where $\bot_D$ is the bottom element of D), i.e. it is given by $\bot_{Eval(D)}(U) = 0$ if $U \neq D$ and $\bot_{Eval(D)}(D) = 1$. If $(E_i)_{i \in I}$ is a directed family of evaluations then the least upper bound $E = \bigsqcup E_i$ in $Eval(D)$ is given by $E(U) = \sup_{i \in I} E_i(U)$. It is shown by Heckmann [Heck95] that, for every dcpo D, the composition operator $* : Eval(D) \times Eval(D) \to Eval(D \times D)$, $(E_1, E_2) \mapsto E_1 * E_2$, is d-continuous.

The evaluation functor $Eval : CONT_\bot \to CONT_\bot$**:** If D, D' are dcpo's and $f : D \to D'$ is a strict, d-continuous function then $Eval(f)$ is strict and d-continuous. From the results of Jones [Jone90], it can be derived that $Eval(D)$ is continuous if D is continuous. Hence, *Eval* can be considered as a functor $CONT_\bot \to CONT_\bot$.

Lemma 12.1.4 *The functor* $Eval : CONT_\bot \to CONT_\bot$ *is locally d-continuous.*

Proof: easy verification. ■

12.2 Ordered balanced trees

For the implementation of the algorithms for deciding bisimulation and simulation equivalence (Chapters 6 and 7), we propose the ordered balanced trees (binary search trees with a certain balance criteria, such as AVL, BB[α] or Red-Black trees) for the computation of certain equivalence classes. The definition of (the several types of) ordered balanced trees can be found in any standard book about data structures; see e.g. [Knut73, CLR96]. We just explain our notations.

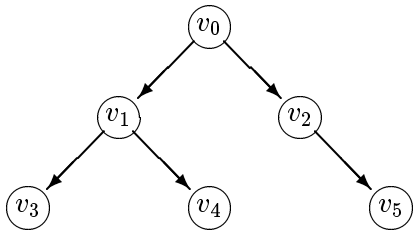
Let I be a nonempty and finite set and p_i, $i \in I$, real numbers. By an *ordered balanced tree* for p_i, $i \in I$, we mean a binary balanced tree (e.g. an AVL-tree [AVL62] or BB[α]-tree [NiRe73]) which arises by successively inserting the elements p_i, $i \in I$, (in any order) and performing the necessary rebalance steps. Each node v is labelled by a key-value $v.key \in \{p_i : i \in I\}$ such that $v_l.key < v.key < v_r.key$ for all nodes v_l (v_r) in the left (right) subtree of v.[6] The construction of an ordered balanced tree for p_i, $i \in I$, takes

[6] Note that we do not allow different nodes to be labelled by the same key-value.

$\mathcal{O}(|I|\log(r+1))$ time and $\mathcal{O}(|I|)$ space where r is the cardinality of $\{p_i : i \in I\}$. We often use additional labels for the nodes, e.g. $v.indices = \{i \in I : p_i = v.key\}$. We describe the additional labels by their final value (i.e. the value in the final tree). For example, let $I = \{1, \ldots, 10\}$ and

$$p_1 = p_4 = 5,\ \ p_2 = p_7 = p_8 = 7,\ \ p_3 = 4,\ \ p_5 = 3,\ \ p_6 = 2,\ p_9 = p_{10} = 0.$$

The final tree depends on the type of ordered trees (AVL-, BB[α] or whatever) and on the order in which the elements p_i are inserted. For instance, it is possible to obtain the following final tree.



$$\begin{array}{llllll} v_0.key & = & 4 & v_0.indices & = & \{3\} \\ v_1.key & = & 2 & v_1.indices & = & \{6\} \\ v_2.key & = & 5 & v_2.indices & = & \{1,4\} \\ v_3.key & = & 0 & v_3.indices & = & \{9,10\} \\ v_4.key & = & 3 & v_4.indices & = & \{5\} \\ v_5.key & = & 7 & v_5.indices & = & \{2,7,8\} \end{array}$$

If we deal with a function $f : X \to \{1, \ldots, 10\}$ where $X = \{x_1, x_2, x_3, x_4\}$ and $f(x_1) = f(x_2) = 5$, $f(x_3) = 6$, $f(x_4) = 7$ and the additional labels

$$v.elements \ = \ \{x \in X : f(x) \in v.indices\}$$

then $v_i.elements = \emptyset$, $i = 0, 1, 3, 4$, $v_4.elements = \{x_1, x_2\}$ and $v_5.elements = \{x_4\}$.

12.3 Multiterminal binary decision diagrams

Chapter 10 deals with MTBDD-based verification methods. In this section, we briefly recall the definition of *multi-terminal binary decision diagrams* (MTBDDs), also called *algebraic decision diagrams* (ADDs). For further details and possible applications see e.g. [CFM$^+$93, BFG$^+$93, HMP$^+$94, CFZ96, SaFu96, FMY97].

MTBDDs were introduced by Clarke et al [CFM$^+$93] as an efficient data structure for matrices. MTBDDs are an extension of Bryant's ordered binary decison diagrams (OBDDs or BDDs for short) [Brya86]. While BDDs are a data structure for boolean functions $f : \{0,1\}^n \to \{0,1\}$, MTBDDs represent functions from bit vectors into a certain domain $\Re$ (i.e. functions of the type $f : \{0,1\}^n \to \Re$). In most applications, the underlying domain $\Re$ is the set $\mathbb{R}$ of real numbers.

MTBDDs: Let *Var* be a finite set of variables, $<$ a total order on *Var* and $\Re$ a nonempty set. A $\Re$-valued MTBDD over $\langle Var, < \rangle$ is a rooted acyclic directed graph with vertex set V containing two types of vertices, *nonterminal* and *terminal*. Each nonterminal vertex v is labelled by a variable $var(v) \in Var$ and has two sons $left(v)$, $right(v) \in V$. Each terminal vertex v is labelled by an element $value(v) \in \Re$. For the labelling of the nonterminal vertices by variables we require that, on any path from the root to a terminal vertex, the variables respect the given ordering $<$, i.e., for all nonterminal vertices v,

- $var(v) < var(left(v))$ if $left(v)$ is nonterminal,
- $var(v) < var(right(v))$ if $right(v)$ is nonterminal.

A BDD is a $\{0,1\}$-valued MTBDD, i.e. a MTBDD where all terminal vertices are labelled by 0 or 1.[7] If $Var = \{x_1, \ldots, x_n\}$ and $x_1 < x_2 < \ldots < x_n$ then we also speak about MTBDDs over $(x_1, \ldots, x_n)$ rather than MTBDDs over $\langle Var, < \rangle$.

Representing real-valued functions by MTBDDs: Each $\Re$-valued MTBDD Q over $\langle Var, < \rangle$ represents a function $f_{\mathrm{Q}}^{Var} : (Var \to \{0,1\}) \to \Re$. Given an interpretation $\iota : Var \to \{0,1\}$ for the variables by the boolen values 0 and 1, the function value $f_{\mathrm{Q}}^{Var}(\iota)$ is the label of the terminal vertex that we obtain by traversing the MTBDD starting in the root and, whenever we reach a nonterminal vertex v, then we go to $left(v)$ if $\iota(var(v)) = 0$, otherwise we go to $right(v)$. When we abstract from the names of the variables then any MTBDD with n (or less) variables represents a function from bit vectors of length n into the underlying domain $\Re$. Formally, for each $\Re$-valued MTBDD Q over $(x_1, \ldots, x_n)$, we define the function

$$f_{\mathrm{Q}}^{(x_1,\ldots,x_n)} : \{0,1\}^n \to \Re$$

by

$$f_{\mathrm{Q}}^{(x_1,\ldots,x_n)}(b_1, \ldots, b_n) \;=\; f_{\mathrm{Q}}^{\{x_1,\ldots,x_n\}}([x_1 := b_1, \ldots, x_n := b_n])$$

where $[x_1 := b_1, \ldots, x_n := b_n]$ denotes those interpretation $\iota : \{x_1, \ldots, x_n\} \to \{0,1\}$ where $\iota(x_i) = b_i$, $i = 1, \ldots, n$.[8] Note that the function $f_{\mathrm{Q}}^{(x_1,\ldots,x_n)}$ depends on the variables $(x_1, \ldots, x_n)$ over which Q is considered. For instance, the MTBDD Q shown in Figure 12.1 (page 316) can be viewed as a MTBDD over (x, y, z) and as a MTBDD over (w, x, y). Q viewed as a MTBDD over (x, y, z) induces the function

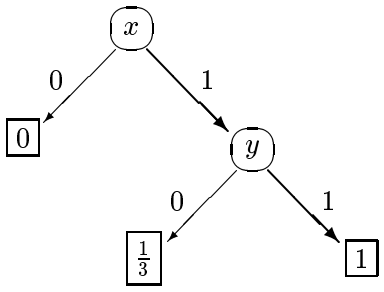


Figure 12.1: The MTBDD Q

$$f_{\mathrm{Q}}^{(x,y,z)}(b_1, b_2, b_3) \;=\; b_1 \cdot (1/3 \;+\; 2/3 \cdot b_2) \;=\; \begin{cases} 0 & : \text{ if } b_1 = 0 \\ \frac{1}{3} & : \text{ if } (b_1, b_2) = (1,0) \\ 1 & : \text{ if } (b_1, b_2) = (1,1) \end{cases}$$

while Q viewed as a MTBDD over (w, x, y) yields the function

$$f_{\mathrm{Q}}^{(w,x,y)}(b_1, b_2, b_3) \;=\; b_2 \cdot (1/3 \;+\; 2/3 \cdot b_3) \;=\; \begin{cases} 0 & : \text{ if } b_2 = 0 \\ \frac{1}{3} & : \text{ if } (b_2, b_3) = (1,0) \\ 1 & : \text{ if } (b_2, b_3) = (1,1). \end{cases}$$

[7] Note that a MTBDD Q over $\langle Var, < \rangle$ is also a MTBDD over $\langle Var', <' \rangle$ for any superset Var' of Var and total order $<'$ on Var' such that $x_1 < x_2$ iff $x_1 <' x_2$ for all $x_1, x_2 \in Var$.

[8] I.e. to obtain the value $f_{\mathrm{Q}}^{(x_1,\ldots,x_n)}(b_1, \ldots, b_n)$, we traverse Q starting in the root. If we reach a nonterminal vertex v labelled by x_i then we go to $left(v)$ (resp. $right(v)$) if $b_i = 0$ (resp. $b_i = 1$). If we reach a terminal vertex v then we put $f_{\mathrm{Q}}(\overline{b}) = value(v)$.

If the variables $(x_1, \ldots, x_n)$ over which a MTBDD Q is considered are clear from the context then we shortly write f_Q rather than $f_Q^{(x_1,\ldots,x_n)}$. Vice versa, each function f from bit vectors of length n into a domain $\Re$ can be represented by a MTBDD. Given a function $f : \{0,1\}^n \to \Re$, the *decison tree* can be viewed as a MTBDD that represents f. For real-valued functions (i.e. $\Re \subseteq I\!R$), the decision tree is obtained from the *Shannon expansion* $f(b_1, \ldots, b_n) = (1 - b_1) \cdot f(0, b_2, \ldots, b_n) + b_1 \cdot f(1, b_2, \ldots, b_n)$. Canonical (in some sense "minimized") MTBDD-representations can be obtained using the REDUCE operator by Bryant [Brya86].

In Section 10.3 we need the following operators on MTBDDs which are taken from [Brya86, CFM+93]. Let Q, Q_1, Q_2 be MTBDDs over $(x_1, \ldots, x_n)$.

Combining two MTBDDs via binary operators: If *op* is a binary operator on $\Re$ (e.g. summation + or multiplication $*$ for $\Re = I\!R$, disjunction $\vee$ or conjunction $\wedge$ for $\Re = \{0,1\}$) then APPLY(Q_1, Q_2, op) returns the unique reduced MTBDD Q over $(x_1, \ldots, x_n)$ where $f_Q = f_{Q_1}$ *op* f_{Q_2}.

Variable renaming: Let $y_1, \ldots, y_k$ be pairwise distinct variables and $1 \leq i_1 < \ldots < i_k \leq n$ such that $y_h \notin \{x_1, \ldots, x_n\} \setminus \{x_{i_1}, \ldots, x_{i_k}\}$, $h = 1, \ldots, k$. Let $x'_i = x_i$ if $i \in \{1, \ldots, n\} \setminus \{i_1, \ldots, i_k\}$ and $x'_{i_h} = y_h$, $h = 1, \ldots, k$. Then,

$$Q\{x_{i_1} \leftarrow y_1, \ldots, x_{i_k} \leftarrow y_k\}$$

denotes those MTBDD over $(x'_1, \ldots, x'_n)$ that arises from Q by renaming simultaneously the variables x_{i_h} by y_h, $h = 1, \ldots, k$. (I.e. for each nonterminal vertex v in Q with $var(v) = x_{i_h}$ we set $var(v)$ to y_h.)[9]

Restriction: If $i \in \{1, \ldots, n\}$ and $b \in \{0,1\}$ then $Q|_{x_i=b}$ denotes those MTBDD over the variables $(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$ that represents the function $\{0,1\}^{n-1} \to \{0,1\}$,

$$(b_1, \ldots, b_i, b_{i+1}, \ldots, b_n) \mapsto f_Q(b_1, \ldots, b_{i-1}, b, b_{i+1}, \ldots, b_n).$$

$Q|_{x_i=b}$ is obtained from Q by removing all x_i-labelled vertices and "replacing" them by their left or right son depending on whether $b = 0$ or $b = 1$. For instance, if $b = 0$ and $var(v) = x_i$ then any edge $w \to v$ in Q is replaced by the edge $w \to left(v)$.

[9] Note that Q and $Q\{x_{i_1} \leftarrow y_1, \ldots, x_{i_k} \leftarrow y_k\}$ represent the same function (when viewed as MTBDDs over $(x_1, \ldots, x_n)$ and $(x'_1, \ldots, x'_n)$ respectively).

Bibliography

[AbLa88] M. Abadi, L. Lamport: The Existence of Refinement Mappings, Proc. LICS'88, pp 165-175, 1988.

[AbJo93] P. Abdulla, B. Jonsson: Verifying Programs with Unreliable Channels, Proc. LICS'93. The full version with the same title has appeared in *Information and Computation*, Vol. 127, No. 2, pp 91-101, 1996.

[Abra91] S. Abramsky: A Domain Equation for Bisimulation, *Information and Computation*, Vol. 92, pp 161-218, 1991.

[AbJu94] S. Abramsky, A. Jung: Domain Theory, In S. Abramsky, D.M. Gabbay and T.S.E. Maibaum (ed.), *Handbook of Logic in Computer Science*, Vol. 3, Clarendon Press, pp 1-168, 1994.

[AHS90] J. Adámek, H. Herrlich, G. Strecker: Abstract and Concrete Categories: The Joy of Cats, John Wiley & Sons, 1990.

[AVL62] G. Adel'son-Velshii, Y. Landis: An Algorithm for the Organization of Information, *Soviet. Math. Dokl.*, Vol. 3, pp 1259-1262, 1962.

[AHU74] A. Aho, J. Hopcroft, J. Ullman: The Design and Analysis of of Computer Algorithms, Addison-Wesley Publishing Company, 1974.

[dAlf97a] L. de Alfaro: Formal Verification of Probabilistic Systems, Ph.D.Thesis, Stanford University, 1997.

[dAlf97b] L. de Alfaro: Temporal Logics for the Specification of Performance and Reliability, Proc. STACS'97, *Lecture Notes in Computer Science*, Vol. 1200, pp 165-176, 1997.

[AlSch84] B. Alpern, F. Schneider: Defining Liveness, *Information Processing Letters*, Vol. 21, 1985.

[ACD90] R. Alur, C. Courcoubetis, D. Dill: Model Checking for Real-Time Systems, Proc. LICS'90, pp 414-425, 1990.

[ACD91a] R. Alur, C. Courcoubetis, D. Dill: Verifying Automata Specifications of Probabilistic Real-Time Systems, Proc. REX Workshop'91, *Lecture Notes in Computer Science*, Vol. 600, pp 27-44, 1991.

[ACD91b] R. Alur, C. Courcoubetis, D. Dill: Model-Checking for Probabilistic Real-Time Systems, Proc. ICALP'91, *Lecture Notes in Computer Science*, Vol. 510, pp 115-127, 1991.

[dAHK98] P. d'Argenio, H. Hermanns, J. Katoen: On Generative Parallel Composition, Proc. PROBMIV'98, Techn. Report CSR-98-4, University Birmingham, pp 105-122, 1998.

[dAKB98] P. d'Argenio, J. Katoen, E. Brinksma: An Algebraic approach to the Specification of Stochastic Systems (extended abstract), Proc. PROCOMET'98, Chapman & Hall, 1998.

[AmRu89] P. America, J. Rutten: Solving Recursive Domain Equations in a Category of Complete Metric Spaces, *Journal of Computer and System Sciences*, Vol. 39, No. 3, pp 343-375, 1989.

[AtEm89] P. Attie, E.A. Emerson: Synthesis of Concurrent Systems with Many Similar Sequential Processes, Proc. POPL'89, pp 191-201, 1989.

[ASB+95] A. Aziz, V. Singhal, F. Balarin, R. Brayton, A. Sangiovanni-Vincentelli: It usually works: The Temporal Logic of Stochastic Systems, Proc. CAV'95, *Lecture Notes in Computer Science*, Vol. 939, pp 155-165, 1995.

[BBS92] J. Baeten, J. Bergstra, S. Smolka: Axiomatizing Probabilistic Processes: ACP with Generative Probabilities, Proc. CONCUR'92, *Lecture Notes in Computer Science*, Vol. 630, pp 472-485, 1992. The full version with the same title has appeared in *Information and Computation*, Vol. 122, pp 234-255, 1995.

[BFG+93] I. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Padro, F. Somenzi: Algebraic Decision Diagrams and their Applications, Proc. ICCAD'93, pp 188-191, 1993. The full version with the same title has appeared in *Formal Methods in Systems Design*, Vol. 10, No. 2/3, pp 171-206, 1997.

[Bai96] C. Baier: Polynomial Time Algorithms for Testing Probabilistic Bisimulation and Simulation, Proc. CAV'96, *Lecture Notes in Computer Science*, Vol. 1102, pp 38-49, 1996. A revised version with the title "Deciding Bisimilarity and Similarity" is submitted for publication.

[Bai97] C. Baier: Trees and Semantics, *Theoretical Computer Science*, Vol. 179, pp 217-250, 1997.

[BaCl98] C. Baier, E. Clarke: The Algebraic Mu-Calculus and MTBDDs, Proc. WoLLIC'98, pp 27-38, 1998.

[BCH+97] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, M. Ryan: Symbolic Model Checking for Probabilistic Processes, Proc. ICALP'97, *Lecture Notes in Computer Science*, Vol. 1256, pp 430-440, 1997.

[BCH98] C. Baier, E. Clarke, V. Hartonas-Garmhausen: On the Semantic Foundations of Probabilistic *VERUS*, Proc. PROBMIV'98, Techn. Report CSR-98-4, University Birmingham, pp 7-32, 1998.

[BaEn98] C. Baier, B. Engelen: Establishing Qualitative Properties for Probabilistic Lossy Channel Systems: an Algorithmic Approach, submitted for publication.

[BaHe97] C. Baier, H. Hermanns: Weak Bisimulation for Fully Probabilistic Processes, Proc. CAV'97, *Lecture Notes in Computer Science*, Vol. 1254, pp 119-130, 1997.

[BaKw97] C. Baier, M. Kwiatkowska: Domain Equations for Probabilistic Processes, Proc. EXPRESS'97, *Electronic Notes in Theoretical Computer Science*, Vol. 7 1997. The full version is available as Techn. Report, CSR-97-7, University Birmingham.

[BaKw98] C. Baier, M. Kwiatkowska: Model Checking for a Probabilistic Branching Time Logic with Fairness, *Distributed Computing*, Vol. 11, No. 3, 1998. A preliminary version with the title "Automatic Verification of Liveness Properties of Randomized Systems" has appeared in Proc. PODC'97, ACM Press, 1997.

[BaKw98a] C. Baier, M. Kwiatkowska: On the Verification of Qualitative Properties of Probabilistic Processes under Fairness Constraints, *Information Processing Letters*, Vol. 66, No. 2, pp 71-79, 1998.

[BKN98] C. Baier, M. Kwiatkowska, G. Norman: Computing Lower and Upper Bounds for *LTL* Formulae over Sequential and Concurrent Markov Chains, Proc. PROBMIV'98, Techn. Report CSR-98-4, University Birmingham, pp 91-104, 1998.

[BMC94] C. Baier, M. Majster-Cederbaum: Denotational Semantics in the Cpo and Metric Approach, *Theoretical Computer Science*, Vol. 135, pp 171-220, 1994.

[BMC97] C. Baier, M. Majster-Cederbaum: How to Interpret and Establish Consistency Results for Semantics of Concurrent Programming Languages, *Fundamenta Informaticae*, Vol. 29, No. 3, pp 225-256, 1997.

[BSV98] C. Baier, M. Stoelinga, F. Vaandrager: private discussion on the decidability of probabilistic branching bisimulation and simulation, October 1998. Draft in preparation.

[dBaMe88] J. de Bakker, J. Meyer: Metric semantics for concurrency, Report CS-R8803, Centre for Mathematics and Computer Science, Amsterdam, 1988.

[dBdRR88] J. de Bakker, W. de Roever, G. Rozenberg (eds.): Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, Proc. REX Workshop'88, *Lecture Notes in Computer Science*, Vol. 354, 1988.

[dBdV96] J. de Bakker, E. de Vink: Control Flow Semantics, MIT Press, 1996.

[dBaZu82] J. de Bakker, J. Zucker: Processes and the Denotational Semantics of Concurrency, *Information and Control*, Vol. 54, No. 1/2, pp 70-120, 1982.

[Barr93] M. Barr: Terminal coalgebras in well-founded set theory, *Theoretical Computer Science*, Vol. 114, pp 299-315, 1993.

[BaWe90] M. Barr, C. Wells: Category Theory for Computing Science, Prentice-Hall International Series in Computer Science, Prentice Hall, 1990.

[BeKl84] J. Bergstra, J. Klop: Process Algebra for Synchronous Communication, *Information and Computation*, Vol. 60, pp 109-137, 1984.

[BMS95] J. Bern, C. Meinel, A. Slobodova: Global Rebuilding of OBDDs Avoiding Memory Requirement Maxima, Proc. CAV'95, *Lecture Notes in Computer Science*, Vol. 939, pp 4-15, 1995.

[BeGor98] M. Bernardo, R. Gorrieri: A Tutorial on EMPA: a Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, Vol. 202, pp 1-54, 1998.

[BeGon92] G. Berry, G. Gonthier: The *ESTEREL* Synchronous Programming Language: Design, Semantics, Implementation, *Science of Computer Programming*, Vol. 19, 1992.

[Bert87] D. Bertsekas: Dynamic Programming, Prentice Hall, 1987.

[BCC+98] A. Biere, A. Cimatti, E. Clarke, Y. Zhu: Symbolic Model Checking without BDDs, submitted for publication.

[BidAl95] A. Bianco, L. de Alfaro: Model Checking of Probabilistic and Nondeterministic Systems, Proc. Foundations of Software Technology and Theoretical Computer Science, *Lecture Notes in Computer Science*, Vol. 1026, pp 499-513, 1995.

[BlMe89] B. Bloom, A. Meyer: A Remark on Bisimulation between Probabilistic Processes, Proc. Symposium on Logical Foundations of Computer Science, *Lecture Notes in Computer Science*, Vol. 363, pp 26-40, 1989.

[BDE+97] R. Blute, J. Desharnais, A. Edalat, P. Panangaden: Bisimulation for Labelled Markov Processes, Proc. LICS'97, pp 149-159, 1997.

[BoSm87] T. Bolognesi, S. Smolka: Fundamental Results for the Verification of Observational Equivalence: a Survey, Proc. Protocol Specification, Testing and Verification, Elsevier Science Publishers, IFIP, pp 165-179, 1987.

[BHR84] S. Brookes, C.A.R. Hoare, A. Roscoe: A Theory of Communicating Sequential Processes, *Journal of the ACM*, Vol. 31 (3), pp 560-599, 1984.

[BCG88] M. Brown, E. Clarke, O. Grumberg: Characterizing Finite Kripke Structures in Propositional Temporal Logic, *Theoretical Computer Science*, Vol. 59, pp 115-131, 1988.

[Brya86] R. Bryant: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Transactions on Computers*, Vol. C-35, No. 8, pp 677-691, 1986.

[BrCh95] R. Bryant, Y. Chen: Verification of Arithmetic Functions with Binary Moment Diagrams, Proc. 32nd ACM/IEEE Design Automation Conference, pp 535-541, 1995.

[BCM+90] J. Burch, E. Clarke, K. McMillan, D. Dill, L. Hwang: Symbolic Model Checking: 10^{20} States and Beyond, Proc. LICS'90, pp 428-439, 1990. The full version with the same title has appeared in *Information and Computation*, Vol. 98 (2), pp 142-170, 1992.

[Camp96] S. Campos: A Quantitative Approach to the Formal Verification of Real-Time Systems, Ph.D.Thesis, Carnegie Mellon University, 1996.

[CCM+95] S. Campos, E.M. Clarke, W. Marrero, M. Minea: Verus: a Tool for Quantitative Analysis of Finite-State Real-Time Systems, Proc. Workshop on Languages, Compilers and Tools for Real-Time Systems, 1995.

[CGT+96] G. Chehaivbar, H. Garavel, N. Tawbi, F. Zulian: Specification and Verification of the Powerscale Bus Arbitration Protocol: An Industrial Experiment with LOTOS, *Formal Description Techniques* IX, pp 435-450, Chapmann & Hall, 1996.

[CCH+96] Y. Chen, E. Clarke, P. Ho, Y. Hoskote, T. Kam, M. Khaira, J.O'Leary, X. Zhao: Verification of all Circuits in a Floating-Point Unit using Word Level Model Checking, Proc. Formal Methods in Computer Aided Designs, 1996.

[CHM90] J. Cheriyan, T. Hagerup, K. Mehlhorn: Can a Maximum Flow be Computed in $\mathcal{O}(nm)$ Time?, Proc. ICALP'90, *Lecture Notes in Computer Science*, Vol. 443, pp 235-248, 1990.

[Chri90a] I. Christoff: Testing Equivalences for Probabilistic Processes, Ph. D. Thesis, Department of Computer Science, Uppsala University, 1990.

[Chri90b] I. Christoff: Testing Equivalences and Fully Abstract Models for Probabilistic Processes, Proc. CONCUR'90, *Lecture Notes in Computer Science*, Vol. 458, pp 126-140, 1990.

[Chri93] L. Christoff: Specification and Verification Methods for Probabilistic Processes, Ph. D. Thesis, Department of Computer Science, Uppsala University, 1993.

[ChCh91] L. Christoff, I. Christoff: Efficient Algorithms for Verification of Equivalences for Probabilistic Processes, Proc. CAV'91, *Lecture Notes in Computer Science*, Vol. 575, pp 310-321, 1991.

[ChCh92] L. Christoff, I. Christoff: Reasoning about Safety and Liveness Properties for Probabilistic Processes, Proc. 12th Conference on Foundations of Software Technology and Theoretical Computer Science, *Lecture Notes in Computer Science*, Vol. 652, pp 342-355, 1992.

[ClEm81] E. Clarke, E.A. Emerson: Design and Synthesis of Synchronization Skeletons from Branching Time Temporal Logic, Proc. Workshop on Logics of Programs, *Lecture Notes in Computer Science*, Vol. 131, pp 52-71, 1981.

[CES83] E.M. Clarke, E.A. Emerson, A.P. Sistla: Automatic Verification of Finite State Concurrent Systems Using Temporal Logic Specifications: A Pratical Approach, Proc. POPL'83, 1983. The full version with the same title has appeared in *ACM Trans. Programming Languages and Systems*, Vol. 1 (2), 1986.

[CFM+93] E. Clarke, M. Fujita, P. McGeer, J. Yang, X. Zhao: Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation, In Proc. IWLS'93: International Workshop on Logic Synthesis, Tahoe City, 1993.

[CFZ96] E. Clarke, M. Fujita, X. Zhao: Multi-Terminal Binary Decision Diagrams and Hybrid Decision Diagrams, In *Representations of Discrete Functions*, T. Sasao and M. Fujita (eds.), Kluwer Academic Publishers, pp 93-108, 1996.

[CGL93] E. Clarke, O. Grumberg, D. Long: Verfication Tools for Finite-State Concurrent Programs, Proc. REX Workshop'93, *Lecture Notes in Computer Science*, Vol. 803, pp 124-175, 1993.

[CGH94] E. Clarke, O. Grumberg, K. Hamaguchi: Another Look at LTL Model Checking, Proc. CAV'94, *Lecture Notes in Computer Science*, Vol. 818, pp 415-427, 1994.

[CKZ96] E. Clarke, M. Khaira, X. Zhao: Word Level Symbolic Model Checking – a New Approach for Verifying Arithmetic Circuits, Proc. 33rd ACM/IEEE Design Automation Conference, IEEE Computer Society Press, 1996.

[CPS90] R. Cleaveland, J. Parrow, B. Steffen: A Semantic Based Verification Tool for Finite State Systems, Proc. Protocol Specification, Testing and Verification IX, Elsevier Science Publishers, IFIP, pp 287-302, 1990.

[CSZ92] R. Cleaveland, S. Smolka, A. Zwarico: Testing Preorders for Probabilistic Processes, Proc. ICALP 1992, *Lecture Notes in Computer Science*, Vol. 623, pp 708-719, 1992.

[CoWi87] D. Coppersmith, S. Winograd: Matrix Multiplication via Arithmetic Progressions, Proc. 19th ACM Symposium on Theory of Computing, pp 1-6, 1987.

[CLR96] T. Cormen, C. Leiserson, R. Rivest: Introduction to Algorithms, McGraw Hill, 1996.

[CoYa88] C. Courcoubetis, M. Yannakakis: Verifying Temporal Properties of Finite-State Probabilistic Programs, Proc. 29th Annual Symp. on Foundations of Computer Science, pp 338-345, 1988.

[CoYa90] C. Courcoubetis, M. Yannakakis: Markov Decision Processes and Regular Events, Proc. ICALP'90, *Lecture Notes in Computer Science*, Vol. 443, pp 336-349, 1990.

[CoYa95] C. Courcoubetis, M. Yannakakis: The Complexity of Probabilistic Verification, *Journal of the ACM*, Vol. 42, No. 4, pp 857-907, 1995.

[Dam94] M. Dam: CTL^* and $ECTL^*$ as Fragments of the Modal Mu-Calculus, *Theoretical Computer Science*, Vol. 126, pp 77-96, 1994.

[Derm70] C. Derman: Finite-State Markovian Decision Processes, Academic Press, New York, 1970.

[DEP98] J. Desharnais, A. Edalat, P. Panangaden: A Logical Characterization of Bisimulation for Labeled Markov Processes, Proc. LICS'98, 1998.

[Dini70] E. Dinic: Algorithm for Solution of a Problem of Maximal Flow in a Network with Power Estimation, Soviet. Math. Dokl., Vol. 11, pp 1277-1280, 1970.

[Dugu66] J. Dugundji: Topology, Allyn and Bacon, inc., 1966.

[EdSm92] A. Edalat, M. Smyth: Compact Metric Information Systems, Proc. REX Workshop'92, *Lecture Notes in Computer Science*, Vol. 666, pp 154-173, 1992.

[Emer85] E.A. Emerson: Automata, Tableaux and Temporal Logics, in "Logic of Programs", *Lecture Notes in Computer Science*, Vol. 193, pp 79-87, 1985.

[Emer90] E.A. Emerson: Temporal and Modal Logic, Volume B of *Handbook of Theoretical Computer Science*, Elsevier Science Publishers (North-Holland), pp 995-1072, 1990.

[Emer92] E.A. Emerson: Real-Time and the Mu-Calculus, Proc. REX Workshop'92, *Lecture Notes in Computer Science*, Vol. 666, pp 176-194, 1992.

[EmCl82] E.A. Emerson, E.M. Clarke: Using Branching Time Logic to Synthesize Synchronization Skeletons, *Sci. Comput. Programming*, Vol. 2, pp 241-266, 1982.

[EmHa85] E.A. Emerson, J. Halpern: Decision Procedures and Expressiveness in the Temporal Logic of Branching Time, *Journal of Computer and System Science*, Vol. 30, pp 1-24, 1985.

[EmHa86] E.A. Emerson, J. Halpern: "Sometimes" and "Not Never" Revisited: on Branching versus Linear Time Temporal Logic, *Journal of the ACM*, Vol. 33, No. 1, pp 151-178, 1986.

[EmJu88] E.A. Emerson, C. Jutla: The Complexity of Tree Automata and Logics of Programs, Proc. FOCS'88, pp 328-337, 1988.

[EmJu91] E.A. Emerson, C. Jutla: Tree Automata, Mu-Calculus and Determinacy, Proc. FOCS'91, pp 368-377, 1991.

[EJS93] E.A. Emerson, C. Jutla, A. Sistla: On Model-Checking for Fragments of the Mu-Calculus, Proc. CAV'93, *Lecture Notes in Computer Science*, Vol. 697, pp 385-396, 1993.

[EmLei85] E.A. Emerson, C. Lei: Modalities for Model Checking: Branching Time Strikes Back, Proc. POPL'85, pp 84-96, 1985.

[EmLei86] E.A. Emerson, C. Lei: Efficient Model Checking for Fragments of the Propositional Mu-Calculus, Proc. LICS'86, pp 267-278, 1986.

[EFT93] R. Enders, T. Filkorn, D. Taubner: Generating BDDs for Symbolic Model checking in CCS, *Distributed Computing*, Vol. 6, pp 155-164, 1993.

[Enge89] R. Engelking: General Topology, Sigma Series in Rure Mathematics, Vol. 6, Heldermann Verlag Berlin, 1989.

[Espa94] J. Esparza: Model Checking Using Net Unfoldings, *Science of Computer Programming*, Vol. 23, pp 151-195, 1994.

[Even79] S. Even: Graph Algorithms, Computer Science Press, 1979.

[FHZ93] M. Fang, C. Ho-Stuart, H. Zedan: Specification of Real-Time Probabilistic Behaviour, Proc. Protocol, Specification, Testing and Verification, IFIP, Elsevier Science Publishers, pp 143-157, 1993.

[Feld83] Y. Feldmann: A Decidable Propositional Dynamic Logic, Proc. 15th ACM Symp. on Theory of Computing, pp 298-309, 1983.

[FeHa84] Y. Feldmann, D. Harel: A Propositional Dynamic Logic, *Journal of Computer and System Science*, Vol. 28, pp 193-215, 1984.

[Fell68] W. Feller: An Introduction to Probability Theory and its Applications, Wiley, Ney York, 1968.

[Fern89] J.C. Fernandez: An Implementation of an Efficient Algorithm for Bisimulation Equivalence, *Science of Computer Programming*, Vol. 13, pp 219-236, 1989.

[FoFu62] L. Ford, D. Fulkerson: Flows in Networks, Princeton University Press, 1962.

[Fran88] N. Francez: Fairness, Springer-Verlag, New York, 1988.

[FMK91] M. Fujita, Y. Matsunaga, T. Kakadu: On Variable Ordering of Binary Decision Diagrams for the Application of Multi-Valued Logic Synthesis, Proc. EDAC'91, pp 50-53, 1991.

[FMY97] M. Fujita, P. McGeer, J. Yang: Multi-Terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation, *Formal Methods in System Design*, Vol. 10, No. 2/3, pp 149-170, 1997.

[GPV+95] R. Gerth, D. Peled, M. Vardi, P. Wolper: Simple On-The-Fly Verification of Linear Time Logic, Proc. Symposium on Protocol Specification, Testing and Verification, pp 3-18, 1995.

[GJS90] A. Giacalone, C. Jou, S. Smolka: Algebraic Reasoning for Probabilistic Concurrent Systems, Proc. IFIP TC2 Working Conference on Programming Concepts and Methods, 1990.

[GHK+80] G. Gierz, H. Hofmann, K. Keimel, J. Lawson, M. Mislove, D. Scott: A Compendium of Continuous Lattices, Springer-Verlag, 1980.

[GiHi94] S. Gilmore, J. Hillston: The PEPA Workbench: A Tool to Support a Process Algebra-Based Approach to Performance Modelling, Proc. 7th Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Wien, 1994.

[vGSST90] R. van Glabbeek, S. Smolka, B. Steffen, C. Tofts: Reactive, Generative, and Stratified Models for Probabilistic Processes, Proc. LICS'90, pp 130-141, 1990. A revised version with the same title by the authors R. van Glabbeek, S. Smolka, B. Steffen has appeared in *Information and Computation*, Vol. 121, pp 59-80, 1995.

[vGlWe89] R. van Glabbeek, W. Weijland: Branching Time and Abstraction in Bisimulation Semantics, *Information Processing*, Vol. 89, pp 613-618, 1989. The full version with the same title has appeared in *Journal of the ACM*, Vol. 43(3), pp 555-600,1996.

[Gode94] P. Godefroid: Partial-Order Methods for the Verification of Concurrent Systems: An Approach to the State Explosion Problem, Ph. D. Thesis, University of Liège, 1994; published in *Lecture Notes in Computer Science*, Vol. 1032 (1996).

[GPS96] P. Godefroid, D. Peled, M. Staskauskas: Using Partial Order Methods in the Formal Validation of Industrial Concurrent Programs, Proc. ISSTA'96, pp 261-269, 1996.

[GHR93] N. Götz, U. Herzog, M. Rettelbach: Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis using Stochastic Process Algebras, Proc. Performance Evaluation of Computer and Communication Systems, *Lecture Notes in Computer Science*, Vol. 729, pages 121-146, 1993.

[Goul88] R. Gould: Graph Theory, The Benjamin/Cummings Publishing Company, 1988.

[GLN+97] C. Gregorio-Rodriguez, L. Llana-Diaz, M. Núñez, P. Paloa-Gostanza: Testing Semantics for a Probabilistic-Timed Process Algebra, Proc. ARTS'97, *Lecture Notes in Computer Science*, Vol. 1231, pp 353-349, 1997.

[GrWe86] G. Grimmett, D. Welsh: Probability: an Introduction, Oxford University Press, 1986.

[GoRo83] W. Golson, W.C. Rounds: Connection between two Theories of Concurrency: Metric Spaces and Synchronisation Trees, *Information and Control*, Vol. 57, pp 102-124, 1983.

[GriVa98] D. Griffioen, F. Vaandrager: Normed Simulations, Proc. CAV'98, *Lecture Notes in Computer Science*, Vol. 1427, pp 332-344, 1998.

[GroVa90] J. Groote, F. Vaandrager: An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence, Proc. ICALP'90, *Lecture Notes in Computer Science*, Vol. 443, pp 626-638, 1990.

[GSB94] R. Gupta, S. Smolka, S. Bhaskar: On Randomization in Sequential and Distributed Algorithms, *ACM Computing Surveys*, Vol. 26, pp 1-86, 1994.

[HMP+94] G. Hachtel, E. Macii, A. Padro, F. Somenzi: Probabilistic Analysis of Large Finite State Machines, Proc. ACM/IEEE DAC'94, pp 270-275, 1994. The full version with the title "Markovian Analysis of Large Finite State Machines" has appeared in *IEEE Transactions on CAD/ICAS*, Vol. 15, No. 12, pp 1479-1493, 1996.

[Halm50] P. Halmos: Measure Theory, Springer-Verlag, 1950.

[Hans91] H. Hansson: Time and Probability in Formal Design of Distributed Systems, Ph.D.Thesis, Uppsala University, 1991; published in *Real-Time Safety Ciritical Systems*, Vol. 1, Elsevier, 1994.

[HaJo89] H. Hansson, B. Jonsson: A Framework for Reasoning about Time and Reliability, Proc. IEEE Real-Time Systems Symposium, IEEE Computer Society Press, pp 102-111, 1989.

[HaJo90] H. Hansson, B. Jonsson: A Calculus for Communicating Systems with Time and Probabilities, Proc. IEEE Real-Time Systems Symposium, IEEE Computer Society Press, pp 278-287, 1990.

[HaJo94] H. Hansson, B. Jonsson: A Logic for Reasoning about Time and Probability, *Formal Aspects of Computing*, Vol. 6, pp 512-535, 1994.

[HaSh84] S. Hart, M. Sharir: Probabilistic Temporal Logic for Finite and Bounded Models, Proc. 16th ACM Symposium on Theory of Computing, pp 1-13, 1984. The full version with the title "Probabilistic Propositional Temporal Logic" has appeared in *Information and Control*, Vol. 70, pp 97-155, 1986.

[HSP83] S. Hart, M. Sharir, A. Pnueli: Termination of Probabilistic Concurrent Programs, *ACM Transactions on Programming Languages*, Vol. 5, pp 356-380, 1983.

[Hart98] J. Hartog: Comparative Semantics for a Process Language with Probabilistic Choice and Non-Determinism, Techn. Report IR-445, Vrije Universiteit Amsterdam, 1998.

[HadVi98] J. Hartog, E. de Vink: Mixing up Nondeterminism and Probability: a Preliminary Report, submitted for publication.

[HarG98] V. Hartonas-Garmhausen: Probabilistic Symbolic Model Checking with Engineering Models and Applications, Ph.D.Thesis, Carnegie Mellon University, 1998.

[HMS97] J. He, A. McIver, K. Seidel: Probabilistic Models for the Guarded Command Language, *Science of Computer Programming*, Vol. 28, No. 2/3, pp 171-192, 1997.

[Henn88] M. Hennessy: Algebraic Theory of Processes, MIT Press, Boston, Mass., 1988.

[HeMi85] M. Hennessy, R. Milner: Algebraic Laws for Nondeterminism and Concurrency, *Journal of the ACM*, Vol. 32, pp 137-161, 1985.

[HHK95] M. Henzinger, T. Henzinger, P. Kopke: Computing Simulations on Finite and Infinite Graphs, Proc. FOCS'95, pp 453-462, 1995.

[HHW97] T. Henzinger, P. Ho, H. Wang-Toi: HYTECH: a Model Checker for Hybrid Systems, Proc. CAV'97, *Lecture Notes in Computer Science*, Vol. 1254, pp 460-463, 1997.

[Herm98] H. Hermanns: Interactive Markov Chains, Ph.D.Thesis, Universität Erlangen-Nürnberg, 1998.

[HHM98] H. Hermanns, U. Herzog, V. Mertsiotakis: Stochastic Process Algebras: between LOTOS and Markov Chains, *Comp. Netw. and ISDN Syst.*, Vol. 9/10, pp 901-924, 1998.

[Herz90] U. Herzog: Formal Description, Time and Performance Analysis – a Framework, in "Entwurf und Betrieb verteilter Systeme", Informatik Fachberichte 264, Springer, 1990.

[Hill94] J. Hillston: A Compositional Approach to Performanec Modelling, Ph.D.Thesis, University Edinburgh, 1994; published in Cambridge University Press (1996).

[Hoar85] C.A.R. Hoare: Communicating Sequential Processes, Prentice Hall, 1985.

[HuKw97] M. Huth, M. Kwiatkowska: Quantitative Analysis and Model Checking, Proc. LICS'97, IEEE Computer Society Press, 1997.

[HuKw98] M. Huth, M. Kwiatkowska: Comparing *CTL* and *PCTL* on Labelled Markov Chains, Proc. PROCOMET'98, Chapman & Hall, 1998.

[HuTi92] T. Huynh, L. Tian: On some Equivalence Relations for Probabilistic Processes, Fundamenta Informaticae, Vol. 17, pp 211-234, 1992.

[Heck95] R. Heckmann: Spaces of Valuations, Technical Report A 09/95, FB 14 Informatik, Universität des Saarlandes.

[HoPe94] G. Holzmann, D. Peled: An Improvement in Formal Verification, Proc. 7th International Conference on Formal Description Techniques, pp 177-194, 1994.

[IyNa96] P. Iyer, M. Narasimha: "Almost Always" and "Definitely Sometime" are not enough: Probabilistic Quantifiers and Probabilistic Model-Checking, Techn. Report, TR-96-16, North Carolina State University, 1996.

[IyNa97] P. Iyer, M. Narasimha: Probabilistic Lossy Channel Systems, Proc. TAPSOFT'97, *Lecture Notes in Computer Science*, Vol. 1214, pp 667-681, 1997.

[Jone90] C. Jones: Probabilistic Non-Determinism, Ph.D.Thesis, University of Edinburgh, 1990.

[JoPl89] C. Jones, G.D. Plotkin: A Probabilistic Powerdomain on Evaluations, Proc. LICS'89, pp 186-195, 1989.

[Jons91] B. Jonsson: Simulations between Specifications of Distributed Systems, Proc. CONCUR'91, *Lecture Notes in Computer Science*, Vol. 527, pp 346-360, 1991.

[JHP89] B. Jonsson, C. Hussain Khan, J. Parrow: Implementing a Model Checking Algorithm by Adapting Existing Automated Tools, Proc. Automatic Verification Methods for Finite State Systems, *Lecture Notes in Computer Science*, Vol. 407, pp 179-188, 1989.

[JHY94] B. Jonsson, C. Ho-Stuart, W. Yi: Testing and Refinement for Nondeterministic and Probabilistic Processes, Proc. FTRTFT'94, *Lecture Notes in Computer Science*, Vol. 863, pp 418-430, 1994.

[JoLa91] B. Jonsson, K.G. Larsen: Specification and Refinement of Probabilistic Processes, Proc. LICS'91, pp 266-277, 1991.

[JoYi95] B. Jonsson, W. Yi: Compositional Testing Preorders for Probabilistic Processes, Proc. LICS'95, pp 431-443, 1995.

[JoSm90] C. Jou, S. Smolka: Equivalences, Congruences and Complete Axiomatizations for Probabilistic Processes, Proc. CONCUR'90, *Lecture Notes in Computer Science*, Vol. 458, pp 367-383, 1990.

[KaSm83] P. Kannelakis, S. Smolka: CCS Expressions, Finite State Processes and Three Problems of Equivalence, Proc. 2nd ACM Symposium on the Principles of Distributed Computing, pp 228-240, 1983. The full version with the same title has appeared in *Information and Computation*, Vol. 86, pp 43-68, 1990.

[Karp91] R. Karp: An Introduction to Randomized Algorithms, *Discrete Applied Mathematics*, Vol. 34, pp 191-201, 1991.

[Kato96] J. Katoen: Quantitative and Qualitative Extensions of Event Structures, Ph.D.Thesis, Universiteit Twente, 1996.

[KLL94] J. Katoen, R. Langerak, D. Latella: Modelling Systems by Probabilistic Process Algebras: an Event Structure Approach, in *Formal Description Techniques* VI, Vol. C 22 of IFIP Transactions, North-Holland, pp 253-268, 1994.

[Kell76] R. Keller: Formal Verification of Parallel Programs, *Communications of the ACM*, Vol. 7(19), pp 561-572, 1976.

[Koze79] D. Kozen: Semantics for Probabilistic Programs, Proc. 20th IEEE Symposium on Foundations of Computer Science, 1979. The full version with the same title has appeared in *Journal of Computer and System Science*, Vol. 22, pp 328-350, 1981.

[Koze83] D. Kozen: Results on the Propositional Mu-Calculus, *Theoretical Computer Science*, Vol. 27, No. 3, pp 333-354, 1983.

[Koze85] D. Kozen: A Probabilistic PDL, *Journal of Computer and System Sciences*, Vol. 30, 1985.

[Knut73] D. Knuth: Sorting and Searching, Vol. 3 of "The Art of Computer Programming", Addison-Wesley, 1973.

[Kura56] K. Kuratowski: Sur une méthode de métrisation complète des certains espaces d'ensembles compacts, *Fundamentae Mathematicae* 43, pp 114-138, 1956.

[KwNo96] M. Kwiatkowska, G. Norman: Probabilistic Metric Semantics for a simple Language with Recursion, Proc. MFCS'96, *Lecture Notes in Computer Science*, Vol. 1113, pp 419-430, 1996.

[KwNo98a] M. Kwiatkowska, G. Norman: A Testing Equivalence for Reactive Probabilistic Processes, Proc. EXPRESS'98, *Electronic Notes in Theoretical Computer Science*, Vol. 16, 1998.

[KwNo98b] M. Kwiatkowska, G. Norman: A Fully Abstract Metric-Space Denotational Semantics for Reactive Probabilistic Processes, Proc. COMPROX'98, *Electronic Notes in Theoretical Computer Science*, Vol. 13, 1998.

[Kwia89] M. Kwiatkowska: Survey of Fairness Notions, *Information and Software Technology*, Vol. 31, No. 7, pp 371-386, 1989.

[LPV94] Y. Lai, M. Pedram, B. Vrudhula: Edge-Valued Binary Decision Diagrams for Integer Linear Programming, Spectral Transformation, and Function Decomposition, *IEEE Transactions on CAD*, Vol. 13, No. 8, pp 959-975, 1994.

[Lamp77] L. Lamport: Proving the Correctness of Multiprocess Programs, *IEEE Transactions on Software Engineering*, Vol. 3, pp 125-143, 1977.

[Lamp80] L. Lamport: Sometimes is Sometimes "Not Never" – on the Temporal Logic of Programs, Proc. POPL'80, pp 174-185, 1980.

[Lamp94] L. Lamport: The Temporal Logic of Actions, *ACM Transactions on Programming Languages and Systems*, Vol. 16, No. 3, pp 872-923, 1994.

[LPY97] K. Larsen, P. Pettersson, W. Yi: UPPAAL: Status & Developments, Proc. CAV'97, *Lecture Notes in Computer Science*, Vol. 1254, pp 456-459, 1997.

[LaSk89] K. Larsen, A. Skou: Bisimulation through Probabilistic Testing, Proc. POPL'89, 1989. The full version with the same title has appeared in *Information and Computation*, Vol. 94, pp 1-28, 1991.

[LaSk92] K. Larsen, A. Skou: Compositional Verification of Probabilistic Processes, Proc. CONCUR'92, *Lecture Notes in Computer Science*, Vol. 630, pp 456-471, 1992.

[LNS82] J. Lassez, V. Nguyen, E. Sonenberg: Fixed Point Theorems and Semantics: a Folf Tale, *Information Processing Letters*, Vol. 14, No. 3, pp 112-116, 1982.

[LSP81] D. Lehmann, A. Pnueli, J. Stavi: Impartiality, Justice and Fairness: The Ethics of Concurrent Termination, Proc. ICALP'81, *Lecture Notes in Computer Science*, Vol. 115, Springer, 1981.

[LeRa81] D. Lehmann, M. Rabin: On the Advantage of Free Choice: a Symmetric and Fully Distributed Solution to the Dining Philosophers Problem, Proc. POPL'81, pp 133-138, 1981.

[LeSh82] D. Lehmann, S. Shelah: Reasoning with Time and Chance, *Information and Control*, Vol. 53, pp 165-198, 1982.

[LiPn85] O. Lichtenstein, A. Pnueli: Checking that Finite State Concurrent Programs Satisfy Their Linear Specification, Proc. POPL'85, pp 97-107, 1985.

[LPZ85] O. Lichtenstein, A. Pnueli, L. Zuck: The Glory of the Past, in "Logics of Programs", *Lecture Notes in Computer Science*, Vol. 193, pp 196-218, 1985.

[Lowe93a] G. Lowe: A Probabilistic Model of Timed CSP, Ph.D.Thesis, Oxford University, 1991.

[Lowe93b] G. Lowe: Representing Nondeterminism and Probabilistic Behaviour in Reactive Processes, Techn. Report PRG-TR-11-93, Oxford University, 1993.

[Lowe95] G. Lowe: Probabilistic and Prioritized Models of Timed CSP, *Theoretical Computer Science*, Vol. 138, pp 315-352, 1995.

[Lync95] N. Lynch: Distributed Algorithms, Morgan Kaufmann Publishers, inc., 1995.

[LSS94] N. Lynch, I. Saias, R. Segala: Proving Time Bounds for Randomized Distributed Algorithms, Proc. PODC'94, pp 314-323, 1994.

[LyVa91] N. Lynch, F. Vaandrager: Forward and Backward Simulations for Timing-Based Systems, Proc. REX Workshop'91, *Lecture Notes in Computer Science*, Vol. 600, pp 397-446, 1991.

[MajC88] M. Majster-Cederbaum: On the Uniqueness of Fixed Points of Endofunctors in a Category of Complete Metric Spaces, *Information Processing Letters*, Vol. 29, pp 277-281, 1988.

[MajC89] M. Majster-Cederbaum: The Contraction Property is Sufficient to Guarantee the Uniqueness of Fixed Points of Endofunctors in a Category of Complete Metric Spaces, *Information Processing Letters*, Vol. 33, pp 15-19, 1988.

[MaZe91] M. Majster-Cederbaum, F. Zetzsche: Towards a Foundation for Semantics in Complete Metric Spaces, *Information and Computation*, Vol. 90, No. 2, pp 217-243, 1991.

[MPM78] V. Malhotra, M. Pramodh Kumar, S. Maheshwari: An $\mathcal{O}(|V^3|)$ Algorithm for Finding Maximum Flows in Networks, Computer Science Program, Indian Institute of Technology, Kanpur 208016, 1978.

[MaPn90] Z. Manna, A. Pnueli: A Hierarchy of Temporal Properties, Proc. PODC'90, pp 377-408, 1990.

[MaPn92] Z. Manna, A. Pnueli: The Temporal Logic of Reactive and Concurrent Systems: Specification, Springer-Verlag, 1992.

[MaPn95] Z. Manna, A. Pnueli: Temporal Verification of Reactive Systems: Safety, Springer-Verlag, 1995.

[MaWo84] Z. Manna, P. Wolper: Synthesis of Communication Processes from Temporal Logic Specifications, *ACM Trans. Programming Languages and Systems*, Vol. 6 (1), pp 68-93, 1984.

[McLan71] S. MacLane: Categories for the Working Mathematician, Graduate Texts in Mathematics, Springer, 1971.

[MBC84] M. Ajmone Marsan, G. Balbo, G. Conte: A Class of Generalized Stocahstic Petric Nets for the Performance Evaluation of Multiprocessor Systems, *ACM Transactions on Computer Systems*, Vol. 2, No. 2, pp 93-122, 1984.

[McMil92] K. McMillan: Symbolic Model Checking, Ph.D.Thesis, Carnegie Mellon University, Pittsburgh, 1992; published in Kluwer Academic Publishers (1993).

[McMil92a] K. McMillan: Using Unfoldings to Avoid the State Explosion Problem in the Verification of Asynchronous Circuits, Proc. CAV'92, *Lecture Notes in Computer Science*, Vol. 663, pp 164-177, 1992.

[McIv98] A. McIver: Reasoning about Efficiency within a Probabilistic Mu-Calculus, Proc. PROBMIV'98, Techn. Report CSR-98-4, University Birmingham, pp 45-58, 1998.

[MKR92] M. Mercer, R. Kapur, D. Ross: Functional Approaches to Generating Orderings for Efficient Symbolic Representation, Proc. ACM/IEEE DAC'92, pp 614-619, 1992.

[Moll82] K. Molloy: Performance Analysis Using Stochastic Petri Nets, *IEEE Trans. on Computers*, Vol. C-31 (9), pp 913-917, 1982.

[MoMcI97] C. Morgan, A. McIver: A Probabilistic Temporal Calculus based on Expectations, Proc. Formal Methods Pacific'97; also available as Technical Report TR-13-97, University of Oxford, 1997.

[MMS+94] C. Morgan, A. McIver, K. Seidel, J. Sanders: Refinement-Oriented Probability for *CSP*, Techn. Report TR-12-94, Oxford University, to appear in *Formal Aspects of Computing*.

[MMS96] C. Morgan, A. McIver, K. Seidel: Probabilistic Predicate Transformers, *ACM Transactions on Programming Languages and Systems*, Vol. 8, No. 3, pp 325-353, 1996.

[MoRa95] R. Motwani, P. Raghavan: Randomized Algorithms, Cambridge University Press, 1995.

[Miln80] R. Milner: A Calculus of Communicating Systems, *Lecture Notes in Computer Science*, Vol. 92, 1980.

[Miln83] R. Milner: Calculi for Synchrony and Asynchrony, *Theoretical Computer Science*, Vol. 25, pp 269-310, 1983.

[Miln89] R. Milner: Communication and Concurrency, Prentice Hall, 1989.

[NiRe73] I. Nievergelt, E. Reinhold: Binary Search Trees of Bounded Balance, SICOMP 2, pp 33-43, 1973.

[dNHe83] R. de Nicola, M. Hennessy: Testing Equivalences for Processes, *Theoretical Computer Science*, Vol. 34, pp 83-133, 1983.

[dNVa90] R. de Nicola, F. Vaandrager: Three Logics for Branching Bisimulation, Proc. LICS'90, pp 118-129, 1990.

[NRS+90] X. Nicollin, J. Richier, J. Sifakis, J. Voiron: *ATP*: an Algebra of Timed Processes, Proc. IFIP TC2 Working Conference on Programming Concepts and Methods, Sea of Gallilea, 1990.

[Niva79] M. Nivat: Infinite Words, Infinite Trees, Infinite Computations, Foundations of Computer Science III, Mathematical Centre Tracts 109, pages 3-52, 1979.

[Niwi88] D. Niwinski: Fixed Points Vs. Infinite Generation, Proc. LICS'88, 1988.

[Norm97] G. Norman: Metric Semantics for Reactive Probabilistic Processes, Ph.D. Thesis, University Birmingham, 1997.

[NúdF95] M. Núñez, D. de Frutos: Testing Semantics for Probabilistic *LOTOS*, in *Formal Description Techniques* VIII, pp 365-380, Chapman & Hall, 1995.

[NdFL95] M. Núñez, D. de Frutos, L. Llana: Acceptance Trees for Probabilistic Processes, Proc. CONCUR'95, *Lecture Notes in Computer Science*, Vol. 962, pp 249-263, 1995.

[OwGr76] S. Owicki, D. Gries: An Axiomatic Proof Technique for Parallel Programs I, *Acta Informatica*, Vol. 6, pp 319-340, 1976.

[OwLa82] S. Owicki, L. Lamport: Proving Liveness Properties of Concurrent Programs, *ACM Transactions on Programming Languages and Systems*, Vol. 4(3), pp 455-495, 1982.

[PaTa87] R. Paige, R. Tarjan: Three Partition Refinement Algorithms, *SIAM Journal of Computing*, Vol. 16, No. 6, pp 973-989, 1987.

[Park74] D. Park: Finiteness is Mu-ineffable, Theory of Computation Report No. 3, The University of Warwick, 1974. Published in *Theoretical Computer Science*, Vol. 3(2), pp 173-181, 1976.

[Park81] D. Park: Concurrency and Automata on Infinite Sequences, Proc. 5th GI Conference, *Lecture Notes in Computer Science*, Vol. 104, pp 167-183, 1981.

[Pele93] D. Peled: All from One, One from All: On Model Checking Using Representatives, Proc. CAV'93, *Lecture Notes in Computer Science*, Vol. 697, pp 409-423, 1993.

[PPH96] D. Peled, V. Pratt, G. Holzmann (eds): Proc. *DIMACS* Workshop on Partial Order Methods in Verification (POMIV'96), Americal Mathematical Society, Series in Discrete Mathematics and Theoretical Computer Science, Bd. 29, 1996.

[PSS98] A. Philippou, O. Sokolsky, I. Lee: Weak Bisimulation for Probabilistic Systems, submitted for publication.

[Plot81] G. Plotkin: A Structural Approach to Operational Semantics, Techn. Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Pnue77] A. Pnueli: The Temporal Logic of Programs, Proc. FOCS'77, pp 46-57, 1977.

[Pnue83] A. Pnueli: On the Extremely Fair Treatment of Probabilistic Algorithms, Proc. 15th ACM Symposium on Theory of Computing, 1983.

[PnRo89] A. Pnueli, R. Rosner: On the Synthesis of a Reactive Module, Proc. POPL'89, pp 191-201, 1989.

[PnZu86a] A. Pnueli, L. Zuck: Verification of Multiprocess Probabilistic Protocols, *Distributed Computing*, Vol. 1, No. 1, pp 53-72, 1986.

[PnZu86b] A. Pnueli, L. Zuck: Probabilistic Verification by Tableaux, Proc. LICS'86, pp 322-331, 1986.

[PnZu93] A. Pnueli, L. Zuck: Probabilistic Verification, *Information and Computation*, Vol. 103, pp 1-29, 1993.

[PoSe95] A. Pogosyants, R. Segala: Formal Verification of Timed Properties of Randomized Distributed Algorithms, Proc. PODC'95, 1995.

[Prat76] V. Pratt: Semantical Considerations of Floyd-Hoare Logic, Proc. FOCS'76, pp 109-121, 1976.

[Prat81] V. Pratt: A Decidable Mu-Calculus, Proc. FOCS'81, pp 421-427, 1981.

[Pria96] C. Priami: Stochastic π-calculus with general distributions, Proc. 4th Int. Workshop on Process Algebra and Performance Modelling, pp 41-57. C.L.U.T. Press, 1996.

[PuSu89] S. Purushothaman, P. Subrahmanyam: Reasoning about Probabilistic Behaviour in Concurrent Systems, *IEEE Transaction on Software Engineering*, SE-13 (6), 1989.

[Pute94] M. Puterman: Markov Decision Processes, John Wiley and Sons, 1994.

[QuSi82] J. Queille, J. Sifakis: Specification and Verification of Concurrent Systems in CESAR, Proc. 5th International Symposium on Programming, *Lecture Notes in Computer Science*, Vol. 137, pp 337-351, 1982.

[QuSi83] J. Queille, J. Sifakis: Fairness and Related Properties in Transition Systems – a Temporal Logic to deal with Fairness, *Acta Informatica*, Vol. 19, pp 195-220, 1983.

[Rabi63] M. Rabin: Probabilistic Automata, *Information and Control*, Vol. 6, 1963.

[Rabi76a] M. Rabin: N-Process Mutual Exclusion with Bounded Waiting by $4 \log N$ Shared Variables, *Journal of Computer and System Science*, Vol. 25, pp 66-75, 1976.

[Rabi76b] M. Rabin: Probabilistic Algorithms, in "Algorithms and Complexity: Recent Results and New Directions" (J. Traub, ed.), Academic Press, New York, pp 21-40, 1976.

[Rabi80] M. Rabin: Probabilistic Algorithms for Testing Primality, *J. Number Theory*, Vol. 12, pp 128-138,1980.

[Ross83] S. Ross: Introduction to Stochastic Dynamic Programming, Academic Press, New York, 1983.

[Rude93] R. Rudell: Dynamic Variable Ordering for Ordered Binary Decision Diagrams, Proc. IEEE ICCAD'93, pp 42-47, 1993.

[Rudi66] W. Rudin: Real Complex Analysis, McGraw-Hill, 1966.

[RuTu93] J.J.M.M. Rutten, D. Turi: On the Foundations of Final Semantics: Non-Standard Sets, Metric Spaces and Partial Orders, Proc. REX Workshop'92, *Lecture Notes in Computer Science*, Vol. 666, pp 477-530, 1993.

[Safr88] S. Safra: On the Complexity of ω-Automata, Proc. FOCS'88, pp 319-327, 1988.

[SaFu96] T. Sasao, M. Fujita: Representations of Discrete Functions, Kluwer Academic Publishers, 1996.

[Schr87] A. Schrijver: Theory of Linear and Integer Programming, J. Wiley & Sons, 1987.

[Seid92] K. Seidel: Probabilistic Communicating Processes, Ph.D.Thesis, Oxford University, 1992.

[Seid95] K. Seidel: Probabilistic Communicating Processes, *Theoretical Computer Science*, Vol. 152, pp 219-249, 1995.

[Seidl96] H. Seidl: A Modal Mu-Calculus for Durational Transition Systems, Proc. LICS'96, pp 128-137, 1996.

[Sega95a] R. Segala: Modeling and Verification of Randomized Distributed Real-Time Systems, Ph.D.Thesis, Massachusetts Institute of Technology, 1995.

[Sega95b] R. Segala: A Compositional Trace-Based Semantics for Probabilistic Automata, Proc. CONCUR'95, *Lecture Notes in Computer Science*, Vol. 962, pp 234-248, 1995.

[Sega96] R. Segala: Testing Probabilistic Automata, Proc. CONCUR'96, *Lecture Notes in Computer Science*, Vol. 1119, pp 299-314, 1996.

[SeLy94] R. Segala, N. Lynch: Probabilistic Simulations for Probabilistic Processes, Proc. CONCUR'94, *Lecture Notes in Computer Science*, Vol. 836, pp 481-496, 1994. The full version with the same title has appeared in *Nordic Journal of Computing*, Vol. 2 (2), pp 250-273, 1995.

[SiCl86] A. Sistla, E. Clarke: Complexity of Propositional Temporal Logics, *Journal of the ACM*, Vol. 32(3), pp 733-749, 1986.

[SVW85] A. Sistla, M. Vardi, P. Wolper: The Completion Problem for Büchi Automata with Applications to Temporal Logic, Proc. ICALP'85, *Lecture Notes in Computer Science*, Vol. 194, pp 465-474, 1985.

[SCV92] R. Sisto, L. Ciminiera, A. Valenzo: Probabilistic Characterization of Algebraic Protocol Specifications, Proc. 12th International Conference on Distributed Computing Systems, pp 260-268, IEEE Comp. Soc. Press, 1992.

[SmSt90] S. Smolka, B. Steffen: Priority as Extremal Probability, Proc. CONCUR'90, *Lecture Notes in Computer Science*, Vol. 458, pp 456-466, 1990. The full version with the same title has appeared in *Formal Aspects of Computing*, Vol. 8, pp 585-606, 1996.

[SmPl82] M. Smyth, G. Plotkin: The Category-Theoretic Solution of Recursive Equations, *SIAM J. Comput.*, Vol. 11, pp 761-783, 1982.

[StWa91] C. Stirling, D. Walker: Local Model Checking in the Modal Mu-Calculus, *Theoretical Computer Science*, Vol. 89, pp 161-177, 1991.

[StVa98] M. Stoelinga, F. Vaandrager: Root Contention in IEEE 1394, submitted for publication.

[StEm84] R. Street, E.A. Emerson: An Automata Theoretic Decision Procedure for Propositional Mu-Calculus, Proc. ICALP'84, *Lecture Notes in Computer Science*, Vol. 172, pp 465-472, 1984.

[SLG94] V. Stoltenberg-Hansen, I. Lindström, E. Griffor: Mathematical Theory of Domains, Cambridge University Press, 1994.

[Stoy77] J. Stoy: Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory. MIT Press, Cambridge, 1977.

[Suth77] W. Sutherland: Introduction to Metric and Topological Spaces, Oxford University Press, 1977.

[Thom90] W. Thomas: Automata on Infinite Objects, *Handbook of Theoretical Computer Science*, Vol. B, Elsevier Science Publishers, Amsterdam, pp 135-191, 1990.

[Thom96] W. Thomas: Languages, Automata, and Logic, Techn. Report 9607, Christian Albrechts Universität Kiel, 1996.

[Toft90] C. Tofts: A Synchronous Calculus of Relative Frequency, Proc. CONCUR'90, *Lecture Notes in Computer Science*, Vol. 458, pp 467-480, 1990.

[Toft94] C. Tofts: Processes with Probabilities, Priority and Time, *Formal Aspects of Computing*, Vol. 6, No. 5, 1994.

[Valm94] A. Valmari: State of the Art Report: Stubborn Sets, *Petri Net Newsletters*, Vol. 46, pp 6-14, 1994.

[Vard85] M. Vardi: Automatic Verification of Probabilistic Concurrent Finite-State Programs, Proc. FOCS'85, pp 327-338, 1985.

[Vard96] M. Vardi: An Automata-Theoretic Approach to Linear Temporal Logic, in "Logics for Concurrency – Structure versus Automata" (F. Moller, G. Birtwistle, eds.), *Lecture Notes in Computer Science*, Vol. 1043, pp 238-265, 1996.

[VaWo86] M. Vardi, P. Wolper: An Automata-Theoretic Approach to Automatic Program Verification, Proc. LICS'86, pp 332-344, 1986.

[VaWo94] M. Vardi, P. Wolper: Reasoning about Infinite Computations, *Information and Computation*, Vol. 115 (1), pp 1-37, 1994.

[Varg62] R. Varga: Matrix Iterative Analysis, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1962.

[dVin98] E. de Vink: On a Functor for Probabilistic Bisimulation and Preservation of Weak Pullbacks, Techn. Report, Vrije Universiteit Amsterdam, 1998.

[dViRu97] E. de Vink, J. Rutten: Bisimulation for Probabilistic Transition Systems: A Coalgebraic Approach, Proc. ICALP'97, *Lecture Notes in Computer Science*, Vol. 1256, pp 460-470, 1997.

[Wins84] G. Winskel: Synchronisation Trees, *Theoretical Computer Science*, Vol. 34, pp 33-82, 1984.

[WVS83] P. Wolper, M. Vardi, A. Sistla: Reasoning about Infinite Computation Paths, Proc. FOCS'83, pp 185-194, 1983.

[WSS94] S. Wu, S. Smolka, E. Stark: Composition and Behaviours of Probabilistic I/O-Automata, Proc. CONCUR'94, *Lecture Notes in Computer Science*, Vol. 836, 1994. The full version with the same title has appeared in *Theoretical Computer Science*, Vol. 176, pp 1-38, 1997.

[YCDS94] S. Yuen, R. Cleaveland, Z. Dayar, S. Smolka: Fully Abstract Characterizations of Testing Preorders for probabilistic Processes, Proc. CONCUR'94, *Lecture Notes in Computer Science*, Vol. 836, pp 497-512, 1994.

[Yi91] W. Yi: A Calculus of Real Time Systems, Ph. D. Thesis, Chalmers University, 1991.

[Yi94] W. Yi: Algebraic Reasoning for Real-Time Probabilistic Processes with Uncertain Information, Proc. FTRTFT'94, *Lecture Notes in Computer Science*, Vol. 863, pp 680-693, 1994.

[YiLa92] W. Yi, K. Larsen: Testing Probabilistic and Nondeterministic Processes, Proc. Protocol, Specification, Testing, Verification XII, pp 47-61, 1992.

[YoGr73] D. Young, R. Gregory: A Survey of Numerical Mathematics, Vol. II, Addison-Wesley Publishing Company Reading, Massachussetts, 1973.

[Zhao96] X. Zhao: Verification of Arithmethic Circuits, Ph.D.Thesis, Carnegie Mellon University, 1996.