# Probabilistic Extensions of Semantical Models

Jerry den Hartog

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

VRIJE UNIVERSITEIT

# Probabilistic Extensions of Semantical Models

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. T. Sminia,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de faculteit der Exacte Wetenschappen
op donderdag 17 oktober 2002 om 13.45 uur
in het auditorium van de universiteit,
De Boelelaan 1105

door

Jeremy Ian den Hartog

geboren te Amsterdam

| | |
|---|---|
| promotor: | prof.dr. J.W. de Bakker |
| copromotoren: | dr. E.P. de Vink |
| | dr. J.J.M.M. Rutten |

# Acknowledgments

The path towards the completion of a thesis is a long one, and many have helped me along this path.

I would like to take this opportunity to mention some of them. I would particularly like to thank Jaco de Bakker, Jan Rutten and Erik de Vink for the extensive attention they paid to this thesis, including much-needed proof reading. I am also grateful to the reading committee members for evaluating this thesis and for their valuable comments, and Mirna Bognar and Michel Oey for acting as Paranyphms.

I took my first steps into the world of scientific research at the Vrije Universiteit under the guidance of Erik de Vink and the watchful eye of Jaco de Bakker. With catching enthusiasm Erik introduced me to the world of $\Phi$'s and $\Psi$'s: The world of writing papers on comparative metric semantics. Erik is one of those rare people who are able to combine a relaxed atmosphere with a great dedication to doing high quality research. Erik, it always was, and still is a pleasure to work with you.

My roommate Mirna Bognar provided many hours of fruitful, and fruitless but still enjoyable discussion. It is very nice to have someone on the same path that you can talk to and share your triumphs and complaints with. Regularly Michel Oey and Gerard Kok dropped in to add to the fun, not to mention some applied probability theory and psychology during our breaks. At lunch time we often joined up with Stefan Blom, Frank Dehne and Frank Niessink. Stefan and I regularly cut the lunch short to join up with Evert Wattel, Perry Groot, Martijn Bot and anyone else we could find for a quick hand of bridge. Thank you all for making my time at the VU so much fun.

I found in-depth knowledge of the research field as well as a source of feedback on my own ideas in the members of the Amsterdam Concurrency Group (ACG) during our regular meetings. The schools and conferences of the research school IPA also helped increase my knowledge of the area of formal methods. A more specific forum for discussing research on probabilistic systems was provided by the PROMISE (PRobabilistic Methods In Software Engineering) colloquium, a cooperation set up with researchers working on probability at several Dutch research institutes. The members of the PROMACS-project (Probabilistic Methods for the Analysis of Continuous Systems) kept me up to date on all of their work at the project meetings.

Over time, I migrated from the VU to the Center for Mathematics and Computer Science (CWI). Here, Jan Rutten took over part of the supervising duties. During this period, I had the pleasure of sharing rooms with Marcello Bonsangue, Alexandru Baltag, Kees Everaars and Peter Zoeteweij. Many a tea break was again spent with Stefan Blom, talking mainly about computer hardware. Finally, I moved to the Formal Methods group at the Technische Universiteit Eindhoven (TU/e). I look forward to spending more time there. I would like to thank my colleges at VU, CWI and TU/e for creating a pleasant working environment over the years.

Luckily I also had family and friends to remind me that there is more to life than work alone. My mother Miny Hulst is an never ending source of energy, support, energy, fun and energy. Her energy is not limited to work. Many a Sunday she had me trying to keep up biking, ice-skating or rollerblading or playing tennis or squash. My sister Deanne den Hartog provided inspirational leadership by example, by preceding me first as a student and then as PhD-candidate at the VU. She also provided refreshing different interpretations of mathematical proofs. Together with Robert Verburg she occasionally dragged me from behind my terminal to visit the theme parks of Europe and the white slopes of "Washington, Tirol" or other famous skiing destinations. With my father Niek den Hartog I had many very philosophical deliberations, while he helped me on very practical jobs and problems. My bi-continental family was there in thought and always made me feel at home whenever I did manage to visit.

Fortunately, my opinion that chasing after a little, hardly bouncing ball is fun, was luckily shared by Ronald Starink and we got together for many intensive squash work-outs. During our time spent on our shared hobby of stripping and rebuilding computers, Stefan Blom proved to be a source for everything you want to know about computers but did not even know how to ask...

# Contents

# Chapter 1

# Introduction

The rapidly increasing complexity of computer systems and computer programs, makes the design of correct and secure programs a challenge. This does not only apply to general purpose computers and their programs. Hardware for specific purposes and embedded systems range in complexity from simple collections of switches to complex dynamically changing networks such as a mobile phone network or an air-traffic control system. The complexity of the latter systems has reached a point where no one can comprehend the system in its entirety and traditional programming approaches fail [87] while correctness of the system is critical. Criteria for the correctness of the behavior of a system are the absence of certain unwanted events (safety properties) and a guaranteed performance (performance and liveness properties). An increasingly important correctness criterion is the protection of the information within a system (security properties). Several design methods have emerged to aid in meeting the challenge of creating and maintaining systems. At first, guidelines to structured programming were developed. Then complete programming paradigms such as object-oriented, functional and logical programming were employed. Another step in the battle of designing correct software is the use of formal methods. Formal methods combine a precise mathematical modeling of the semantics of systems with formal reasoning. The use of formal methods aids in making precise and understanding the specification of a system, and formal methods can be used to verify properties of the implementation of a system. Some examples of methods used for specification and verification are process algebras, model checking and theorem proving and logics developed to reason about program properties such as Hoare logic and dynamic logic. The use of formal methods does not necessarily guarantee correctness but it does help in finding errors early on and it can nearly eliminate some classes of errors [101].

This thesis deals with probabilistic extensions of semantical models. The work reported concentrates mainly on the semantical modeling of probabilistic choice. Furthermore, specification of probabilistic properties and logical reasoning is studied as well but to a lesser extent. A question that can be asked is: "Why is probability included in semantical modeling, computers are not random, are they?". The easiest answer to this question is that many programming languages contain an operation or system call 'random', that can introduce random behavior into the computation. (Even though the numbers generated by 'random' are usually pseudo-random numbers, it is conceptually a probabilistic choice.) There are, however, more causes for probabilistic behavior of computer systems. For example, for a faulty communication channel the error-rate provides stochastic information on the likelihood to have packages delivered without corruption along the channel.

Probability plays an important role in many algorithms. Randomized algorithms can be used to solve programs more efficiently. For example, the sorting algorithm quicksort is probabilistic and also one of the best of the known primality testing algorithms is probabilistic [156]. In some settings randomization can even be used to solve problems for which no deterministic solution exists. Self-stabilizing token rings are early examples of the use of probabilistic protocols (see e.g. [116]). Other examples of issues that can be addressed using probabilistic choice are fairness concerns, breaking of symmetry between identical parallel components and finding safe or optimal strategies in a known strategy game. Probability is also used to design algorithms to solve optimization problems, e.g. for efficient searching. Evolutionary computing (see e.g. [17, 16]) is also inherently probabilistic. (See e.g. [164, 5, 156] for more on randomized algorithms.) Thus probability in the semantical model of a system can be used to express behavior of systems which is inherently probabilistic or only approximately known.

## 1.1 Semantic modeling

As mentioned above, this thesis concentrates mainly on the semantical modeling of probabilistic choice. The main goal is to add probability to existing semantical models and to study the interplay of probability with other concepts. To this end several languages that are used to specify programs, or more generally systems, are introduced: First a basic language with sequential composition, recursion and probabilistic choice is studied. Then several extensions of this basic setting with probability are treated. The main concepts that are combined with probabilistic choice are nondeterminism, concurrency and action refinement. Another extension that is considered allows a probabilistic choice to be made between infinitely many options in a single step.

The study of the control flow aspects of the programs is the main goal of this work. As such the actual data used and the basic computational steps are of less concern and therefore in most cases abstracted away from in the languages at hand. Schematic or process description languages are used in which the basic steps of the computation are described by atomic actions which are left without further interpretation in the modeling.

The semantic modeling is mostly done in a metric setting, i.e. the mathematical domains used to express the meaning of programs have a metric structure. The presence of recursion in the languages introduces the possibility of infinite behavior. One of the major advantages of using the metric setting is that it helps in dealing with the issues caused by this possibility.

Two main approaches are distinguished in giving the semantics of a program resulting in two, possibly different semantics. The first approach captures the computational intuition of a program. The behavior that can be observed for a machine running the program is central in this approach. A virtual machine that runs the program is described by a transition system which gives the possible transitions for each possible state. Each program has a corresponding state in the transition system, and the possible behavior when running the program can be found by looking at all transitions possible for this state and again for the resulting states. In this way a tree of possible steps is obtained referred to as the transition tree for the program. From the transition tree the observable behavior for a program is obtained by removing information that cannot be seen by someone observing the machine. The model that specifies the meaning of a program in this way is called the operational semantics.

The semantical model of a program is used to verify properties of a program. Model checking, for example, is a method that is often used to check properties of a program by looking at all transitions for the program as given by the transition system. The largest problem in model checking is that of the state space explosion. The number of states involved can grow very vast if the complexity of a program increases. To address this issue one can try split the verification of a large program into verification of properties of parts of the program. To aid in doing this, the denotational semantics gives the meaning of programs in a compositional way. In the denotational approach the structure of the program is central in giving the semantics instead of the computational intuition. If, for example, a program consists of a probabilistic choice between two subprograms, the meaning of the program should be a combination of the meanings of these two subprograms. For this to be made precise one needs to specify how to combine the meanings of two subprograms to obtain the meaning of the program consisting of the probabilistic choice between these subprograms. To this end a domain of all possible meanings is formed. The elements of this domain are called denotations (hence the name denotational semantics) or processes. An operation on this domain which composes two processes into another process then specifies how meanings of subprograms are combined. In this way, finding the meaning of a program is reduced to finding the denotations for all basic components of the program and giving the operations which compose these denotations.

For this approach of composing meanings to work, clearly it must be possible to compose processes: If two programs that have the same meaning are both combined with a third program then for both cases the meaning of the resulting program will be the same. As the operational semantics is not always compositional this shows that the denotational semantics cannot always be the same as the operational semantics. In general, the denotational semantics will have to maintain extra information about a program in its meaning to be able to compose programs. In case extra information does need to be added, an abstraction function is given to be able to recover the operational meaning of a program from the process yielded by the denotational semantics. (In chapter 5, the issue of full abstractness is also addressed. Full abstractness of a denotational model basically states that this model is the best one possible; only information that is really needed to make the model compositional is added.)

The motivation for introducing the denotational approach is to deal with the increasing complexity of larger programs. Using the denotational meaning of a program, verification of a program can be split into verification for subprograms of the program (for suitable properties).

In the metric approach, the domain of processes has a metric structure. This structure helps in reasoning about infinite processes, in the definition of the operations on the domain as well as in the comparison of the operational semantics and the denotational semantics.

As mentioned above, specification of probabilistic properties and logical reasoning is also treated in this thesis. In this setting properties of a program that one wants to check are expressed using formulae in a suitable logical specification language. To check that these properties are satisfied one can use methods based on the semantics of the programs, such as model checking. One can also reason about programs directly without calculating the semantics. The use of Hoare logic falls in the latter category. Hoare triples consist of a precondition and a postcondition both expressed in some logical specification language together with a program. Hoare logic provides a deduction system to obtain correctness of such triples. In this thesis a Hoare-style logic to reason about probabilistic programs is given.

## 1.2 Probabilistic extensions of semantical models: Overview, main contributions and chapter dependencies.

In this section an overview of the results presented in this thesis is given. The mathematical preliminaries in chapter 2 below recall some known definitions and results used in later chapters. No new work is presented there.

In chapter 3 the modeling of probabilistic choice itself is the central theme. This chapter provides both a technical and a conceptual basis for all further chapters. Probabilistic choice is introduced into a schematic language $\mathcal{L}_P$ by adding a binary operator for probabilistic choice, denoted $\oplus_\rho$. The program '*heads* $\oplus_{\frac{1}{2}}$ *tails*', for example, can be used to describe the tossing of a fair coin. A probabilistic choice with more than two but still finitely many options can be expressed by repeated uses of the operator $\oplus_\rho$ (possibly for different values of $\rho$).

To find the operational meaning of programs in the language $\mathcal{L}_P$ the notion of transition system is extended to a notion of probabilistic transition system. A minor technical complication with probabilistic transitions is that multiple occurrences of the same transition should not be identified. The extension of transition systems introduced in chapter 3 deals with this complication in a relatively transparent manner.

The transition system specifies the possible transitions in each step. A program is executed by choosing, in each step, one of the possible transitions at that stage. Each transition can have some observable effect, together forming the observable behavior for this single execution of the program. A property of the behavior of a program that one can observe is called an observable event. For a single execution of a program one can, thus, check whether or not an observable event is produced. For a probabilistic program, however, an event may occur in some executions of the program but not in others. The operational meaning of a probabilistic program is described by giving, for each observable event, the probability that an execution of the program will produce this event. Measures are the mathematical structures used in probability theory to assign a probability to all events. In chapter 3 the notion of a compact support measure is introduced. The compact support measures are a subclass of the measures, suitable for modeling the observable behavior of programs.

A denotational semantics for the language $\mathcal{L}_P$ with probabilistic choice is also given in chapter 3. In the denotations or processes describing the meaning of programs, the probabilities for choices made in each step are given instead of the probabilities of all observable events. Instead of using measures, the collection of options together with their probabilities are used to model a step. As multiple occurrences cannot be identified, a multiset is used to describe this collection. To be able to use the metric techniques in this setting, a distance between finite multisets is defined.

The main technical contributions of chapter 3 are the introduction of a method of specifying probabilistic transition systems, the definition of the functor to construct the metric space of all finite multisets over a given metric space and a new characterization of the distance on the space of compact support measures.

In chapter 4 the modeling techniques for probabilistic choice developed in chapter 3 are used to study the interplay of nondeterministic choice and concurrency with probabilistic choice. In general a program is called nondeterministic if there is more than one possible flow of control in the execution of the program. A nondeterministic choice introduces nondeterminism by allowing a choice between execution of different subprograms. Here nondeterministic choice is seen as some choice made in an unspecified manner. In chapter 4 an operator, denoted $\Box$, for nondeterministic choice is introduced into the language.

Concurrency is one cause of nondeterministic behavior. An interleaving interpretation of concurrency is used in this thesis: Each step produced by a program consisting of several components running in parallel is produced by one of the component programs, or possibly by communication between parallel components. Any of the components running in parallel may produce the next step. Many factors that one does not want to model or may not even be able to model, influence which component produces the next step. The selection of the process which produces the next step is therefore modeled as a choice made in some unspecified manner, i.e. a form of nondeterministic choice.

Nondeterministic choice can also be used to express many other different aspects of a system besides concurrency. For example, user interaction and underspecification can also be expressed through nondeterministic choice. The different uses of nondeterministic choice lead to different interpretations of the operator $\Box$. Already without the presence of probability, the different interpretations of the operator $\Box$ lead to different models. With the addition of probabilistic choice the influence of the interpretation of nondeterministic choice becomes even larger. Interpretations that could be modeled in the same manner in a setting without probabilistic choice now need to be distinguished.

In chapter 4 three main classes of interpretations of nondeterministic choice will be considered, with priority for nondeterminism, with priority for probabilistic choice or without priorities. In section 4.2 the nondeterministic choices are always made before probabilistic choices in finding the next step of a system. This approach, referred to as giving priority to the nondeterminism, is well suited for an interpretation of nondeterministic choice as a choice made by a user. In this setting, one can look at program properties that may hold if the right choices are made or properties that must hold, no matter how the choices are made.

In section 4.3 the order of the choices is reversed compared to section 4.2 by giving priority to probability: In section 4.3 probabilistic choices are always made before nondeterministic choices in finding the next step of a system. This approach fits with a resource oriented view of nondeterministic choice: One is interested in the probability that a given set of nondeterministic options is available, i.e. a given set of resources is offered.

Finally in section 4.4 an approach without 'priority' for either choice is treated. This section addresses some of the disadvantages of giving priority to either choice. Instead of giving a denotational semantics, a notion of bisimulation called first step bisimulation is treated in this section. The notion of first step bisimulation extends the commonly used notions of Larsen-Skou bisimulation [149] for probabilistic systems and the standard notion of strong bisimulation [171, 159] for nondeterministic systems. The first step bisimulation is shown to be correct with respect to the operational semantics; two programs that are bisimilar have the same traces. In the metric setting equality in a denotational domain is usually used instead of defining a bisimulation relation as these relations usually coincide in the metric approach. (A more precise formulation of this property can be given in a coalgebraic setting [179, 191].) The first step bisimulation is shown to be a congruence for all operators except for postfixing where restrictions on the program being appended are needed. As first step bisimulation is not a full congruence relation, no compositional model can be given that yields the same equivalence as first step bisimulation.

Chapter 4 provides an overview of several different interpretations of nondeterminism and corresponding models in a setting which also contains probabilistic choice. The main technical contributions of this chapter are the introduction, for the first two interpretations of nondeterministic choice, of an operational and a denotational model and the comparison of these models. Also an operational model is presented for the third interpretation and a notion of bisimulation is introduced that extends both Larsen-Skou bisimulation and strong bisimulation.

The work in chapter 4 shows that nondeterminism and probability are not easy to combine. Many choices have to be made about the exact properties of the nondeterminism that is considered. One may wonder if this is an intrinsic property of the combination of nondeterminism and probability or whether it is just a weakness in our modeling of probabilistic choice. The work in chapter 5 and chapter 6 argues in favor of the former, the combination of nondeterminism and probability is intrinsically difficult. In chapter 5 a model with action refinement is extended with probabilistic choice using the same techniques to model probabilistic choice as in chapters 3 and 4 without creating major issues. A full abstractness result obtained in this chapter also indicates that the compact support measures are indeed an appropriate way of modeling probabilistic choices. In chapter 6 the technical modeling of the probabilistic choice is different. Nondeterminism returns in this chapter as underspecification of properties as well as in the form of a specific operator for nondeterministic choice. In both cases several choices need to be made about the interpretation of the nondeterminism. This again indicates that different ways of modeling nondeterminism are needed for the different possible interpretations of nondeterminism.

Section 5.2 treats the concept of action refinement in an interleaving framework. Usually when treating action refinement a true concurrency model is used because no compositional model is thought to be possible in the interleaving setting. An operational semantics for a language with action refinement, however, can easily be given in an interleaving framework. For the denotational model in section 5.2 two programs are considered semantically equivalent whenever their meanings coincide for all interpretations of the actions describing the basic elements of the computation, i.e. if their meaning coincides *under all refinements*. The view is used that in the framework of schematic languages employed here, the fact that the elementary actions are uninterpreted induces as a natural counterpart that the semantic equivalence of two programs requires equality of their associated meanings under all interpretations of the elementary actions, or equivalently, under all their possible refinements. Using this view of program equivalence, a compositional denotational model can also be given in the interleaving framework as is shown in subsection 5.2.2.

The strength of the probabilistic modeling techniques is shown in section 5.3 where probabilistic choice and action refinement are combined. The work in this section is a relatively straightforward combination of the techniques introduced in chapter 3 with those discussed in section 5.2.

In both section 5.2 and section 5.3 an operational semantics and a denotational semantics are given. The processes used to express the denotational meaning of programs in section 5.3 are based on compact support measures. The definition of the denotational semantics requires that an operation for sequential composition is defined on these processes. To this end a notion of composition of measures is treated that can be applied in this specific setting. (General composition of compact support measures is treated in chapter 7.) In both sections 5.2 and 5.3 a full abstractness result is obtained for the denotational model. This shows that the denotational models are optimal in a particular way: The denotational model contains more information about a program than the operational model; it provides the meaning of the program for all possible interpretations of the atomic actions, in accordance with the view mentioned above. The full abstractness result expresses that the denotational model does not contain too much information; all information that is added is needed to obtain a compositional model. As an auxiliary result we obtain in section 5.3 that for any compact support measure, a program can be found the meaning of which is arbitrarily close to the given measure. In other words, a syntactic representation of a measure can be given. This shows that the space of compact support measures is also 'optimal' in some sense: The space of all measures only contains those objects that are needed either to model actual program behavior, or to guarantee completeness of the space.

Chapter 5 treats a language with action refinement in an interleaving setting and also deals with a language which combines action refinement and probabilistic choice. The main technical contributions of this chapter are operational and denotational models with full abstractness results for both languages, an operator for the sequential composition of compact support measures in a specific setting and the syntactical representation of compact support measures.

Chapter 6 deviates from the approach in the other chapters in that the main concern of this chapter is not the semantic modeling of a language but rather the reasoning about properties of the programs in the languages. A probabilistic Hoare-style logic is introduced in this chapter. Unlike the semantical models for schematic languages discussed above, the languages and Hoare logic in chapter 6 explicitly deal with data, and also deal with infinite behavior in a different way.

Three languages are considered. The first language $\mathcal{L}_{\text{pif}}$ is a basic language with assignment, sequential composition, conditional choice and probabilistic choice. Two extensions of this language, either with iteration $\mathcal{L}_{\text{pw}}$ or with nondeterministic choice $\mathcal{L}_{\text{pnif}}$

are also treated. As the programs in these languages can be probabilistic, the properties that one wants to check for these programs may also state claims about probabilities of given events. To be able to express such probabilistic properties, a notion of probabilistic predicate is introduced. The probabilistic predicates allow the specification of logical properties while at the same time enabling the use of arithmetical properties of probabilities.

To reason about programs and probabilistic predicates, deduction systems are introduced for each of the languages considered. These deduction systems are used to deduce so-called Hoare triples. A Hoare triple consists of a precondition, a program and a postcondition and expresses that if the precondition holds then the postcondition will be met after execution of the program. To show that the reasoning supported by these deduction systems is sound, (denotational) semantics are also given for the languages in chapter 6. A complete partial order approach is used as, in this input-output setting, it is simpler than giving a metric model and the main benefits of the metric setting are not needed here.

Each of the deduction systems introduced is shown to be sound. Each Hoare triple deduced using this system expresses a valid claim about the precondition, program and postcondition that form the triple. For the languages $\mathcal{L}_{\text{pif}}$ and $\mathcal{L}_{\text{pnif}}$ a completeness result is also available: Any Hoare triple that expresses a valid claim can be deduced using the deduction system as long as its postcondition satisfies some minor restrictions. To obtain the completeness results for the languages $\mathcal{L}_{\text{pif}}$ and $\mathcal{L}_{\text{pnif}}$ the weakest preconditions are found for these languages. For a given program and postcondition, the weakest precondition is the weakest property which guarantees that the postcondition will hold after execution of the program.

The main technical contributions of chapter 6 are the introduction of the notion of probabilistic predicate, the specification of sound and complete proof systems for the languages $\mathcal{L}_{\text{pif}}$ and $\mathcal{L}_{\text{pnif}}$ and a sound proof system for the language $\mathcal{L}_{\text{pw}}$ as well as the definition of weakest preconditions for $\mathcal{L}_{\text{pif}}$ and $\mathcal{L}_{\text{pnif}}$.

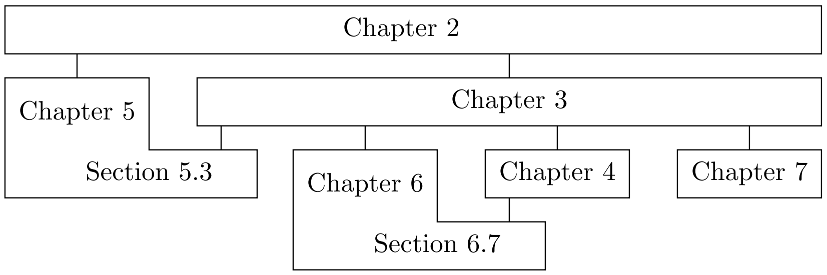
All probabilistic choices discussed so far have had one important aspect in common: Each probabilistic choice has finitely many alternatives. Each of these choices can be seen as a choice which can be decided by a finite series of coin flips. Some probabilistic processes have infinitely many options in a single step and can therefore not be expressed in this way. Consider, for example, the selection of a random real number from an interval $[a, b]$ using a uniform distribution. Each number greater than or equal to $a$ but less than or equal to $b$ can be selected. This amounts to infinitely many options. The uniform distribution on $[a, b]$ is an example of a continuous distribution. Examples of processes that may exhibit continuous probabilistic behavior are processes dealing with time, such as the life time of a light bulb, as well as processes describing hybrid automata such as a process monitoring the amount of fluid in a leaking tank. Continuous probability also plays an important role in much work in the area of performance modeling and real time systems. There are also other types of infinite probabilistic choices which are not based on a continuous choice. An example of a process with such infinite probabilistic choices is a process that counts the number of arrivals in a given time interval. The outcome of this counting process may be any number in $\mathbb{N}$ giving infinitely many options.

Chapter 7 provides an overview of the possibilities of methods developed in the previous chapters in dealing with processes with infinite probabilistic choices. The recursion present in the probabilistic language treated in chapter 3 already introduced the need to express probabilities for a process with infinitely many options. A compact support measure was used for this. In chapter 7 measures are also used to describe the infinite probabilistic choice. An important difference with chapter 3 is that, while in chapter 3 the measures are only needed for processes describing the meaning of complete programs, the measures in chapter 7 are already needed to describe a single step. In chapter 7, the steps produced by a program still need to be combined into a single process describing the meaning of the program. Finite probabilistic choices can be combined by using multiplication and addition. The combination of infinite probabilistic choices uses integration over a measure. Integration, however, is only possible for measurable functions. To guarantee that the integration is possible, extra structure is required. This extra structure is provided by using stochastic kernels [169] in the modeling instead of just looking at measures. In this way the measures describing the steps are linked in such a way that measurability is guaranteed for the functions that need to be integrated.

In chapter 7 a metric version of stochastic kernels is introduced based on compact support measures. Using these kernels the notion of probabilistic transition system is extended to be able to deal with steps described by measures. A transition system and operational semantics are given for a language with the construct of random assignment. A random assignment statement is a probabilistic assignment in which the value assigned to a variable is chosen according to some measure over possible values.

The main technical contributions of chapter 7 are the definition of a metric variant of the notion of a kernel as well as the introduction of a general method of composing compact support measures through the use of this variant of kernels.

The following graph illustrates the dependencies between the chapters of this thesis.



The preliminaries in chapter 2 contain definitions and results used in all chapters. Chapter 3 introduces probabilistic choice and other concepts used in all parts dealing with probabilistic choice. Section 5.2 does not depend on chapter 3 as no probability is present in this section. Section 5.3 which adds probability, however, does depend on chapter 3. The other three chapters 4, 6 and 7 also depend on chapter 3. Finally section 6.7, which adds nondeterminism to the probabilistic Hoare-style logic developed in earlier sections of chapter 6, depends on chapter 4 for a description of the interpretation of nondeterministic choice in a probabilistic setting.

Parts of this thesis are based on the following work: The report [107] treats the combination of nondeterminism and probability. Chapter 3 and the first sections of chapter 4 build on this report. The report [110] and the conference paper [109] form the basis for the remainder of chapter 4. Section 5.2 has been previously published in the journal article [112]. The results of section 5.3 have appeared in the conference paper [113]. The conference paper [108] introduces the probabilistic Hoare-style logic discussed in chapter 6. The journal article [111] presents the more precise syntax used in chapter 6 and adds the completeness result presented in section 6.5. Section 6.7 and chapter 7 have not been previously published.

## 1.3 Related approaches

In this section several approaches to formal methods and to semantical modeling in particular are briefly discussed. Only some basic references are given here. An overview of related work can be found within each chapter.

This thesis uses the metric approach to giving semantic models. The use of metric spaces in semantical modeling was initiated by Arnold and Nivat [15, 166]. A lot of work in this area has been done by the Amsterdam Concurrency group (ACG). See e.g. [36] and more recently also [38, 39, 55]. A comprehensive overview of the use of metric techniques and results is given in the monograph [38]. Further references can be found there.

Domain equations are regularly employed to specify the metric domains in this thesis. Domain equations over metric spaces were pioneered by De Bakker and Zucker [41] and a general, categorical approach was developed by America and Rutten [6]. The work presented here is based on the thesis of Van Breugel [54] where sufficient conditions for the existence of unique solutions for domain equations are given. The domain equations used in this thesis will be shown to satisfy these conditions. References to further work in the field of domain equations can also be found in [54].

The notion of a probabilistic transition system used in this thesis extends Plotkin's structured operational semantics [173] to be able to deal with probabilistic choice. The use of some form of probabilistic transition system is a standard way of describing the operational behavior of a probabilistic system. Probabilistic automata (see e.g. [181]) are also essentially probabilistic transition systems.

In the metric setting a trace model is constructed from the probabilistic transition system and a denotational semantics is used to give a compositional notion of equality of processes. An alternative approach is to define a notion of bisimulation on the transition systems. Two programs are identified by a bisimulation relation if either program can mimic the steps of the other program: If one program produces a step to some next state then the other program can produce a similar step resulting in a related next state. For nondeterministic systems strong or Park-Milner bisimulation [171, 159] is a standard notion. The most standard definition for bisimulation on probabilistic systems is due to Larsen and Skou [149]. Probabilistic bisimulation relates states which, not only produce the same steps, but also yield the same probabilities for the resulting classes of states.

Weak bisimulation [159, 160] is a variation where 'internal' and 'external' computation are distinguished and only the external behavior has to be the same. Several notions of weak bisimulation have been introduced for probabilistic systems. (See e.g. [26, 10, 172].)

A whole spectrum of equivalences of processes, which include trace equivalence, as well as several bisimulation equivalences is treated in [88, 89].

Closely related to the metric approach is the use of coalgebras [179, 180]. Many of the metric techniques can be seen as instances of more general coalgebraic methods. A coalgebra defines a structure on some object by the use of some endofunctor $\mathcal{F}$. A coalgebra can thus be used to describe the structure of a system. A general notion of similarity for all $\mathcal{F}$-coalgebras exists. Several notions of bisimulation have been shown to fit in this general framework including the probabilistic Larsen-Skou bisimulation [190]. A 'final' coalgebra is a canonical representation of a system. Bisimilarity coincides with equality in this canonical representation. Usually the denotational domains used in a metric setting can be described as the final coalgebra for some functor $\mathcal{F}$. A notion of bisimulation equivalence coinciding with equality in the denotational domain is thus automatically obtained.

Although tools have been developed to calculate bisimulation equivalence classes (see e.g. [24, 122, 185, 26] for probabilistic bisimulation) it is useful to be able to check bisimilarity, or more generally equivalence of processes without having to actually find all transitions and calculating the bisimulation classes. An important strand of research in this area uses process algebras to accomplish this. In a process algebra an equational theory is used to express the equivalence of processes. The equational theory contains open equations expressing properties like e.g. the associativity of sequential composition. Equational and logical reasoning are thus used to obtain equivalence from the syntax of a system rather than having to find the semantics. Transition systems with a notion of bisimulation equivalence are given, but are only needed to justify the equational theory. The process algebra can also be used to reduce a given program to some equivalent standard representation. A vast amount of research exists in the area of process algebras. Early work on process algebras can be found e.g. in [43, 42]. See e.g. the text books [82, 22, 44] for a description of process algebras and some of the results obtained using process algebras.

Extensions of process algebra with probabilistic choice have been considered e.g. in [8, 21]. Transition systems for the programs together with a notion of bisimulation are used to justify the equations in the process algebra. These transition systems differ only on some technical points from the transitions systems used to give the operational semantics in this thesis. Between the bisimilarity in the process algebra setting and the denotational equivalence used e.g. in chapter 3 similar minor differences exist.

For process algebras which also deal with time (see e.g. [18, 83, 20]) extensions with probability have also been considered [9, 125]. In stochastic process algebras, such as TIPP [124, 117] and SPADES [64, 68] the probability considered is continuous. Chapter 7 of this thesis aims to extend the modeling techniques introduced in earlier chapters to be able to express continuous probability.

Another way of reasoning about a bisimulation relations is by giving a logical charterisation: A logical language, usually some form of model logic, is given and processes are shown to be bisimilar exactly when they satify the same formulas. Showing that two processes are not bisimilar can then be done by giving a destinguishing formula in the logic. Desharnais et al give a logical charaterization of bisimulation for both discreet and continuous probabilistic processes in [75]. A surpricingly weak logical language needed for this charaterization. Logical charictarizations of bisimulation are not considered any further in this thesis. See e.g. [149, 75, 148] for more on this topic.

Above we have mentioned semantical domains based on metric spaces, and coalgebras. Partial orders are also often used to define structures. In the complete partial order (cpo) approach (see e.g. [187, 34, 197]) a partial order structure is defined on a domain such that every ascending sequence has a least upper bound. The cpo structure is used to solve reflexive definitions and to be able to deal with infinite behavior. The use of least fixed points replaces the use of unique fixed points in metric spaces.

The thesis work of Jones [133] uses evaluations, which are a slight simplification of measures, to introduce probabilistic semantical domains. (In the setting used in this thesis each evaluation has a unique extension to a measure, though not necessarily a compact support measure.) Both a metric domain of evaluations and a complete partial order domain of evaluations can be defined. A comparison of these two domains and of denotational models for a probabilistic process language is given in [23]. The metric model captures bisimulation and the cpo model describes simulation.

Event structures (e.g. [198, 137]) and partially ordered multisets (e.g. [85, 98]) also use partial orders. Here, however, the order is not defined on the processes but on the elements of the computation. The order expresses the structure of the computation. If one element is smaller than another, it must appear earlier in the computation. Two incomparable elements are independent and can be computed concurrently. In this way 'true concurrency' can be expressed within a process. This thesis will only consider so-called interleaving models of concurrency.

The papers [30, 32, 31] provide comparisons of partial order semantics with metric semantics.

Instead of specifying the complete behavior of a program, one may want to require only some specific properties of this behavior. Several logical approaches for the specification of and reasoning about program properties have been developed. As mentioned above, chapter 6 treats a probabilistic version of Hoare logic. Hoare logic [126] allows specification of properties of the state which then specify preconditions and postconditions for programs. A program together with a precondition and a postcondition is referred to as a Hoare triple. A proof system is introduced to deduce valid Hoare triples. See [34, 12, 13] for an overview of Hoare-style logics.

Other logical approaches do not specify properties of the state but rather properties of the computation, for example using dynamic, temporal and modal logics. Verification methodes such as model checking are then used to check if a system satisfies the properties. Model checking starts with a finite state machine representation of a system and uses exhaustive search of the state space to check a given logical formula. An advantage of this approach is that if the formula fails to hold, a counter example showing why this is the case is found. Since the state space can easily become vary large, a symbolic representation of the state space e.g. using binary decision diagrams (BDDs) is often used. Also reduction techniques are used to find minimal representations of the system. For example process algebras can be used to find a smaller bisimilar system.

Model checking for probabilistic systems has also been studied extensively, see e.g. [45, 3, 145, 4]. Multi-terminal binary decision diagrams (MTBDDs) [60, 4] are used in symbolic model checking of probabilstic systems. Chapter 7 contains a discussion of some of the work done in the area of model checking continuous time Markov chains.

# Chapter 2

# Mathematical preliminaries

In this chapter several definitions and results from topology and category theory are recalled. This chapter is not intended to provide a comprehensive introduction to these subjects. Rather, it is aimed to list the main results and notation used in later chapters. Results are presented in a way suitable for this instead of in their most general form.

## 2.1 Metric spaces

The presentation in this thesis assumes knowledge of basic metric and topological notions. In this section several definitions and results are briefly recalled. A more extensive introduction to metric spaces and their application to semantical modeling can be found in [38]. The monographs [81, 79] can be consulted for general topological definitions and results.

**Definition 2.1.1**

*(a) A metric space $(M, d)$ is a set $M$ with a function $d : M \times M \to [0, \infty)$ such that for all $x, y, z \in M$*

1. $d(x, y) = 0 \iff x = y$
2. $d(x, y) = d(y, x)$
3. $d(x, y) \leq d(x, z) + d(z, y)$

*The function $d$ is called a metric or a distance on $M$. Often the metric $d$ is clear from the context and $M$ is written for $(M, d)$.*

*(b) An ultrametric space $(M, d)$ is a metric space in which the third requirement is strengthened to*

3. $d(x, y) \leq \max\{\, d(x, z), d(z, y) \,\}$

*for all $x, y, z \in M$.*

(c) *A function $d : M \times M \to [0, \infty)$ is called* 1*-bounded if $d(x, y) \leq 1$ for all $x, y \in M$. The metric space $(M, d)$ is called* 1*-bounded if the metric $d$ is* 1*-bounded.*

In this thesis we will mainly deal with 1-bounded ultrametric spaces.

**Example 2.1.2** *The real numbers $\mathbb{R}$ with the usual metric $d(x, y) = |x - y|$ is a metric space but not an ultrametric space. This space is also not* 1*-bounded.*

*For any, not necessarily* 1*-bounded, metric space $(M, d)$ the space $(M, d_1)$ with $d_1(x, y) = \min\{\, d(x, y), 1 \,\}$ is a* 1*-bounded metric space.*

*For any set $A$ we have that $(A, d)$ with $d(x, y) = \begin{cases} 0 & x{=}y \\ 1 & otherwise \end{cases}$ is a 1-bounded ultrametric space. This metric $d$ is called the discrete metric on $A$.*

An important metric space is that of sequences over a given set $A$ called the *alphabet*. These sequences are also referred to as *words*. The set of all finite words is denoted by $A^*$. The set of all infinite words is denoted by $A^\omega$. Combining these two sets gives the set of all words over $A$, denoted by $A^\infty$. The concatenation of the words $w$ and $w'$ is denoted by $w \cdot w'$ or simply by $ww'$. For an infinite word $w$ we have that $w \cdot w' = w$. For $S$ a set of words we additionally put $wS = \{\, ww' \mid w' \in S \,\}$. The *Baire metric* defines the distance between words. Two words are closer the longer the prefix they have in common.

**Example 2.1.3** *For a word $w$, $w[n]$ denotes the sequence $w$ truncated after $n$ elements, i.e. the prefix of length $n$. Given any alphabet $A$ the spaces $(A^*, d_B)$, $(A^\omega, d_B)$ and $(A^\infty, d_B)$, with $d_B(w, w') = \inf\{\, 2^{-n} \mid w[n] = w'[n] \,\}$, are 1-bounded ultrametric spaces. The metric $d_B$ is called the Baire metric.*

Recursive definitions play an important role in semantical modeling using metric spaces. To be able to guarantee the existence of a solution (see theorem 2.1.9), the underlying metric space should be *complete*. In the next definition a few auxiliary topological notions are given along with the notion of a complete metric space.

**Definition 2.1.4** *Let $(M, d)$ be a metric space.*

(a) *For any element $x$ in $M$ and ratio $\epsilon$ greater than* 0 *, the open ball $B_\epsilon(x)$ is the set of all points in $M$ with distance less than $\epsilon$ from $x$, thus*

$$B_\epsilon(x) \;=\; \{\, y \in M \mid d(x, y) < \epsilon \,\}$$

(b) *A subset $O$ of $M$ is called open if an open ball which remains completely within $O$ can be placed around all $x$ in $M$, i.e. $M$ is open when $\forall x \in O : \exists \epsilon > 0 : B_\epsilon(x) \subseteq 0$.*

(c) *A subset $C$ of $M$ is called closed if it is the complement of an open subset of $M$.*

(d) *A Cauchy sequence in $M$ is a sequence $(x_i)_{i \in \mathbb{N}}$ for which the elements of the tail are close together, that is the sequence satisfies $\forall \epsilon > 0 : \exists N_\epsilon : \forall i, j > N_\epsilon : d(x_i, x_j) < \epsilon$.*

(e) *The limit of a sequence $(x_i)_{i \in \mathbb{N}}$ is a point $x$ such that the distance between $x$ and $x_i$ goes to zero when $i$ goes to infinity, that is $x$ satisfies $\forall \epsilon{>}0 : \exists N_\epsilon : \forall i{>}N_\epsilon : d(x, x_i){<}\epsilon$.*

(f) *A metric space $(M, d)$ is called complete if every Cauchy sequence in $M$ has a limit in $M$.*

Note that any open set is the union of a collection of open balls. A subset $C$ of $M$ is closed exactly when every point of $M$ that is a limit of a sequence in $C$ is also in $C$.

**Example 2.1.5** *Consider the metric space $(A^\infty, d_B)$ with the alphabet $A = \{ a, b \}$. An open ball with radius ${\frac{1}{2}}^2$ around the word abab consists of all words starting with aba, $B_{{\frac{1}{2}}^2}(abab) = \{ abaw \mid w \in A^\infty \}$ The open ball with radius ${\frac{1}{2}}^3$ around the word abab consists of all words starting with abab. The open ball with radius ${\frac{1}{2}}^4$ around the word abab consists of only the word abab itself. To get a distance between a word w and abab of less than ${\frac{1}{2}}^4$, $w[5]$ has to be equal to $abab[5] = abab$. Only abab itself has this property.*

*The set $\{ a, aa, aaa, \dots \}$ is open but not closed. The set $\{ a^\omega \}$ is closed but not open. The singleton set $\{ abab \}$ is both open and closed.*

*The sequence $a, aa, aaa, \dots$ is a Cauchy sequence in $A^*$ and also in $A^\infty$. The word $a^\omega$ is the limit of this sequence in $A^\infty$. In $A^*$ the sequence does not have a limit.*

*For any set $A$ the spaces $(A^\omega, d_B)$ and $(A^\infty, d_B)$ are complete. The space $(A^*, d_B)$ is not complete (unless $A$ is empty).*

*If $C$ is a closed subset of $M$ and $(M, d)$ is a complete metric space then $(C, d)$ is also a complete metric space.*

As $x$ is always in $B_\epsilon(x)$, an open ball is always a nonempty set. In an ultrametric space two open balls are disjoint or one is contained in the other. For two balls of size $\epsilon$ this means that the balls are disjoint or the same.

**Lemma 2.1.6** *Let $(M, d)$ be an ultrametric space and $\epsilon$ and $\delta$ two ratios with $0 < \epsilon \leq \delta$. Then for all $x, x' \in M$ we have $B_\epsilon(x) \subseteq B_\delta(x')$ or $B_\epsilon(x) \cap B_\delta(x') = \emptyset$.*

**Proof** If $x \in B_\delta(x')$ then $B_\epsilon(x) \subseteq B_\delta(x')$ because for all $y \in B_\epsilon(x)$ we have $d(y, x') \leq \max d(y, x), d(x, x') \leq \delta$ and thus $y \in B_\delta(x')$.

If $x \notin B_\delta(x')$ then $B_\epsilon(x) \cap B_\delta(x') = \emptyset$ because for all $y \in B_\delta(x')$ we have $d(x, x') \leq \max d(x, y), d(y, x')$. Since $d(y, x') < \delta$ and $d(x, x') > \delta$ this gives $d(x, y) \geq d(x, x') \geq \delta \geq \epsilon$ and thus $y \notin B_\epsilon(x)$. □

Several operations on sets can be extended to metric spaces. By introducing the right metric on the resulting set, these operations not only yield metric spaces but also preserve completeness and ultrametricity of metric spaces.

**Lemma 2.1.7**

(a) *For any two metric spaces $(M, d_M)$ and $(N, d_N)$ the Cartesian product, $(M \times N, d_{M\times N})$ the disjoint union, $(M + N, d_{M+N})$ and the function space, $(M \to N, d_{M\to N})$ with*

$$\begin{aligned}
d_{M\times N}((m, n), (m', n')) &= \max\{ d_M(m, m'), d_N(n, n') \} \\
d_{M+N}(x, y) &= \begin{cases} d_M(x, y) & \textit{if } x, y \in M \\ d_N(x, y) & \textit{if } x, y \in N \\ 1 & \textit{otherwise} \end{cases} \\
d_{M\to N}(f, g) &= \sup\{ d_N(f(m), g(m)) \mid m \in M \}
\end{aligned}$$

*are also metric spaces. If both* $(M, d_M)$ *and* $(N, d_N)$ *are ultrametric spaces then* $(M \times N, d_{M\times N})$, $(M + N, d_{M+N})$ *and* $(M \to N, d_{M\to N})$ *are also ultrametric. If both* $(M, d_M)$ *and* $(N, d_N)$ *are complete then so are* $(M \times N, d_{M\times N})$, $(M + N, d_{M+N})$ *and* $(M \to N, d_{M\to N})$.

*(b) For any metric space* $(M, d)$ *the space of closed subsets of* $M$ *with the Hausdorff distance,* $(\mathcal{P}_{cl}(M), d_H)$, *where* $\mathcal{P}_{cl}(M) = \{\, S \subseteq M \mid S \text{ closed} \,\}$ *and*

$$\begin{aligned} d_H(S, S') \quad = \quad \inf\{\, \epsilon > 0 \mid & \forall x \in S : \exists y \in S' : d(x, y) \leq \epsilon \text{ and} \\ & \forall y \in S' : \exists x \in S : d(x, y) \leq \epsilon \,\} \end{aligned}$$

*is also a metric space. If* $(M, d)$ *is an ultrametric space then* $(\mathcal{P}_{cl}(M), d_H)$ *is also ultrametric. If* $(M, d)$ *is complete then so is* $(\mathcal{P}_{cl}(M), d_H)$.

This lemma provides the means to build a metric space with the given structure for certain simple structures. A more powerful way of building metric spaces is treated in section 2.2. First one of the reasons why metric spaces are useful in metric modeling is derived: A recursive equation of the right type has a unique solution on a complete metric space. An equation of the right type is an equation of the form $x = f(x)$ where $f$ is a *contractive function.*

**Definition 2.1.8** *Let* $(M, d_M)$ *and* $(N, d_N)$ *be metric spaces. A function* $f : M \to N$ *is called continuous the reverse image of an open set is again open and it is called* $\alpha$*-Lipschitz if for all* $m, m' \in M$:

$$d_N(f(m), f(m')) \quad \leq \quad \alpha d_M(m, m')$$

*The function* $f$ *is called contractive if it is* $\alpha$*-Lipschitz for some* $\alpha < 1$ *and nonexpansive if it is* $1$*-Lipschitz. The space of all nonexpansive functions from* $M$ *to* $N$ *is denoted by* $M \xrightarrow{1} N$.

The set $M \xrightarrow{1} N$ is a closed subset of $M \to N$. If $M$ and $N$ are compete, this means that $M \xrightarrow{1} N$ is also complete. For a function $f : M \to M$, a solution of the equation $x = f(x)$ is called a *fixed point* of the function $f$. Contractive functions have at most one fixed point.

**Theorem 2.1.9** *(Banach's Fixed Point Theorem) For a contractive function* $f : M \to M$ *on a complete metric space* $M$ *there exists a unique point* $x \in M$, *called the fixed point of* $f$, *that satisfies* $f(x) = x$.

The fixed point of a function $f$ is denoted by *fix*$(f)$.

**Example 2.1.10** *The function* $f$ *on the space* $(\{\, a, b \,\}^{\infty}, d_B)$ *given by* $f(w) = aw$ *is a contractive function. The unique fixed point fix*$(f)$ *of* $f$ *is* $a^{\omega}$.

An important property of subsets of metric spaces that has not yet been mentioned is that of *compactness.* The notion of compact sets extends the notion of finite sets. Finite collections, modeled by finite sets, are import structures within semantical modeling. The collection of finite subsets of a space, however, does not always form a complete metric space. The limit of a sequence of finite sets does not need to be finite. The compact sets are exactly those sets which can be obtained as the limit of a sequence of finite sets.

**Definition 2.1.11** *Let $(M, d)$ be a metric space. An open cover of a subset $C$ of $M$ is a collection of open sets $(O_i)_{i \in I}$ such that each element of $C$ is in at least one of these open sets, $C \subseteq \cup_{i \in I} O_i$. A subset $C$ of a metric space is called compact if every open cover $(O_i)_{i \in I}$ of $C$ has a finite subcover, i.e. there exists a number of indices $i_1, \ldots, i_n$ in $I$ such that $C \subseteq O_{i_1} \cup O_{i_2} \cup \ldots \cup O_{i_n}$.*

*The space of all compact subsets of $M$ is denoted by $\mathcal{P}_{co}(M)$.*

Each finite set is clearly compact. Each compact set is also a closed set, thus the space $\mathcal{P}_{co}(M)$ is a subset of $\mathcal{P}_{cl}(M)$. The same metric $d_H$ (see lemma 2.1.7(b)) is used to turn $\mathcal{P}_{co}(M)$ into a metric space. The space of compact sets $\mathcal{P}_{co}(M)$ is a closed subset of the space of all closed sets $\mathcal{P}_{cl}(M)$. As such $\mathcal{P}_{co}(M)$ is a complete metric space for each complete space $M$.

**Lemma 2.1.12** *A set $C$ is compact exactly when it is the limit of finite sets.*

**Proof** That the limit of finite sets is compact is clear from the completeness of $\mathcal{P}_{co}(M)$ (see [38]). The reverse implication, that each compact set is the limit of finite sets can be seen as follows: For a compact set $C$ and a positive number $n$ take the cover $(B_{1/n}(x))_{x \in C}$. This cover must have a finite subcover $B_{1/n}(x_{(1,n)}), \ldots, B_{1/n}(x_{(m_n,n)})$. When $n$ goes to infinity, the finite sets $\{ x_{(1,n)}, \ldots, x_{(m_n,n)} \}$ go to $C$ as each point in this set is in $C$ and each point in $C$ has at most distance $1/n$ to a point in this set. $\square$

Thus we see that the space of compact sets $\mathcal{P}_{co}(M)$ is the smallest complete subspace of $\mathcal{P}_{cl}(M)$ that contains all finite sets.

**Lemma 2.1.13** *If $f : X \to Y$ is a continuous function and $C$ is a compact subset of $X$ then $f[C] = \{ f(x) \mid x \in C \}$ is a compact subset of $Y$.*

The proof of this lemma can be found e.g. in [38, lemma 2.13].

## 2.2 Category theory and domain equations

Throughout this thesis different metric spaces are employed to model the meaning of programs. Thus the semantical domain which contains all possible meanings of programs is given as a metric space. To describe the meaning of a program, the structure of the (elements of) the semantical domain is essential. Several examples of metric spaces have already been given, as well as ways of combining metric spaces. For example, a sequence can be used to describe events that occur one after another. Joining sets with disjoint union and creating pairs using Cartesian product are two important ways of building structured domains. A pair in the Cartesian product can be used to combine two events that belong together and disjoint union can be used to describe that one out of two different types of events may occur. From these examples one can see that relatively simple structure in the domain can be constructed directly. If, however, the required structure of the domain is more complicated, it is convenient to be able to specify the structure using *domain equations*. A domain equation gives a property that the domain must satisfy instead of specifying the structure of the domain directly. The equations may be recursive, allowing easy specification of possibly infinite structures.

**Example 2.2.1** *The domain equation $\mathbb{P} \simeq \{1\} + \mathbb{P}$ specifies a domain $\mathbb{P}$ with infinitely many distinct copies of the element* 1.

Specifying a domain using domain equations is of course only useful if the equations have a solution. For the domain equations to completely describe the structure of the domain the solution should also be 'unique'. What exactly uniqueness of the solution means in this setting will be explained below. Several types of domain equations can be shown to have a unique solution using metric means (cf. [38]) but for other types a categorical approach is required. This section introduces the categorical notions required to show that the domain equations used in this thesis have a unique solution. The main result of this section is that a domain equation built with a so called *locally contractive functor* (definition 2.2.8) has a unique solution. The only explicit use of *categories* and *functors* (definition 2.2.6) in this thesis is in the building and solving of domain equations.

**Definition 2.2.2** *A category $\mathcal{C}$ is a collection of objects with for each pair of objects $X$ and $Y$ a collection of arrows from $X$ to $Y$ and a notion of composition $\circ$ of arrows satisfying*

- *For every arrow $f$ from $X$ to $Y$ and arrow $g$ from $Y$ to $Z$, $g \circ f$ is an arrow from $X$ to $Z$.*
- *For every object $X$ there is an identity arrow $\mathrm{id}_X$ satisfying $\mathrm{id}_X \circ f = f$ for every arrow $f$ to $X$ and $f \circ \mathrm{id}_X = f$ for every arrow $f$ from $X$.*
- *The composition of arrows is associative. That is, for all objects $Q, X, Y, Z$ and arrows $f$ from $Q$ to $X$, $g$ from $X$ to $Y$ and $h$ from $Y$ to $Z$ we have that $h \circ (g \circ f) = (h \circ g) \circ f$.*

Looking at the properties of objects and arrows in a category one immediately sees the connection with sets and functions. Indeed, the collection of all sets with functions as arrows is a standard example of a category, denoted by *SET*.

**Example 2.2.3** *Taking sets as objects and the functions from $X$ to $Y$ as the arrows from $X$ to $Y$ for any sets $X$ and $Y$ gives the category SET. It is obvious that the requirements of a category are satisfied.*

*The category CMS has complete metric spaces as its objects and continuous functions as arrows.*

*Note that taking metric spaces as objects and any functions as arrows also gives a category. When working with metric spaces, however, one often restricts arrows to continuous functions as these functions preserve important structures within the metric spaces.*

*The category CUMS has complete ultrametric spaces as its objects and nonexpansive functions as arrows. The category CUMS is a* subcategory *of CMS: All objects and arrows in CUMS are also in CMS and the composition of arrows is the same in CUMS and CMS (for arrows from CUMS).*

*The product of categories is also a category. The category CUMS $\times$ CUMS with pairs of complete ultrametric spaces as objects and pairs of nonexpansive function as arrows is a category. The composition of the arrows is element wise.*

The sets $\{ n \in \mathbb{N} \mid n \text{ is even} \}$ and $\{ n \in \mathbb{N} \mid n \text{ is uneven} \}$ are clearly not the same as they have different elements. Both, however, have the same structure (infinitely many independent elements). As such one does not want to distinguish these two sets as being two different domains, just two ways of describing the same domain. Different objects in a category which are structurally the same are called *isomorphic*.

**Definition 2.2.4** *Let $\mathcal{C}$ be a category and $X$, $Y$ objects in $\mathcal{C}$. The objects $X$ and $Y$ are called isomorphic if there exist arrows $f$ from $X$ to $Y$ and $g$ from $Y$ to $X$ such that $g \circ f$ is the identity $\mathrm{id}_X$ on $X$ and $f \circ g$ is the identity $\mathrm{id}_Y$ on $Y$.*

Two isomorphic objects in the category have the same structure. One can go back and forth between the two objects using the arrows $f$ and $g$. Sets are just collections without any other structure. In the category *SET* two objects are isomorphic if a bijection exists between the sets, i.e. if they have the same cardinality.

**Example 2.2.5** *In the category SET the objects $S_e = \{ n \in \mathbb{N} \mid n \text{ is even} \}$ and $S_u = \{ n \in \mathbb{N} \mid n \text{ is uneven} \}$ are isomorphic because one can take the functions $f : S_e \to S_u$ given by $f(x) = x + 1$ and $g : S_u \to S_e$ given by $g(x) = x - 1$. Clearly both $f \circ g$ and $g \circ f$ are the identity function on $S_u$ and $S_e$ respectively.*

*In the category CUMS $\times$ CUMS the objects $M \times N$ and $N \times M$ are isomorphic for any spaces $M$ and $N$ in CUMS.*

*In the category CUMS the objects $(\{ a \}^\infty, d_B)$ and $(\{ a \}^\infty, d_B) \times (\{ b \}^\infty, d_B)$ are not isomorphic. Intuitively this is clear as the structure of these spaces is different. Formally one checks that no bijection $f$ exists between $(\{ a \}^\infty, d_B)$ and $(\{ a \}^\infty, d_B) \times (\{ b \}^\infty, d_B)$ such that both $f$ and $f^{-1}$ are nonexpansive.*

On a metric space, equations are defined using nonexpansive functions. For a contractive function $f$, the equation $x = f(x)$ has a unique solution. In domain equations, a *functor* is used instead of a function.

**Definition 2.2.6** *For two categories $\mathcal{C}$ and $\mathcal{D}$, a functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ is a function assigning to every object $X$ in $\mathcal{C}$ an object $\mathcal{F}(X)$ in $\mathcal{D}$ and to every arrow $f$ from $X$ to $Y$ in $\mathcal{C}$ an arrow $\mathcal{F}(f)$ from $\mathcal{F}(X)$ to $\mathcal{F}(Y)$ in $\mathcal{D}$ such that identities and compositions are preserved, i.e. $\mathcal{F}(\mathrm{id}_X) = \mathrm{id}_{\mathcal{F}(X)}$ and $\mathcal{F}(g \circ f) = \mathcal{F}(g) \circ \mathcal{F}(f)$.*

A functor is a function on the objects of the category, but also on the arrows of the category. To create a functor for operations on sets and metric spaces one thus needs to specify the effect of the operations on functions.

**Example 2.2.7** *The function $\mathcal{F}$ which adds the discrete metric to any given set and does not change the functions is a functor from SET to CUMS. Any function between discrete metric spaces is nonexpansive thus each arrow in SET is indeed assigned an arrow in CUMS. Also each set is assigned an ultrametric space and identities and composition are preserved.*

*The function $\mathcal{P}$ which gives the powerset of a set as the object and which lifts functions by applying them element wise is a functor on SET, i.e. a functor from SET to SET.*

*The function $\mathcal{P}_{cl}$ which assigns the space $(\mathcal{P}_{cl}(M), d_H)$ of closed subsets of $M$ to the space $(M, d)$ is a functor on CUMS. Similar functors are $\mathcal{P}_{co}$ and $\mathcal{P}_{nco}$ which yield the compact subsets and the nonempty compact subsets of $M$ respectively.*

*Another functor on CUMS is the scaling $id_{\frac{1}{2}}$. The scaling functor assigns the space $(M, \frac{1}{2}d)$ to a metric space $(M, d)$, i.e. the elements are the same as in $M$ but their distance is reduced by $\frac{1}{2}$. The nonexpansive functions are unaffected: On the arrows of CUMS, $id_{\frac{1}{2}}$ is the identity.*

*Cartesian product $\times$ and disjoint union $+$ are functors from CUMS$\times$CUMS to CUMS. The Cartesian product functor assigns the space $(M \times N, d_N \times d_M)$ (see lemma 2.1.7) to the pair of spaces $(M, d_M)$ and $(N, d_N)$. An arrow in CUMS $\times$ CUMS is a pair of functions. This pair of functions is exactly a function on the Cartesian product. Disjoint union is similar.*

Using a functor $\mathcal{F}$ on a category $\mathcal{C}$ one can form the domain equation $\mathbb{P} \simeq \mathcal{F}(\mathbb{P})$. From this point we restrict ourselves to complete ultrametric spaces, in particular to the categories *CUMS* and *CUMS* $\times$ *CUMS*. Although much more general results can be obtained (see e.g. [54]) all domain equations used in this thesis remain within this setting. The recursive equation $x = f(x)$ has a unique solution if the function $f$ is contractive. A similar property can be derived for domain equations: The domain equation $\mathbb{P} \simeq \mathcal{F}(\mathbb{P})$ has a unique solution if $\mathcal{F}$ is a *locally contractive* functor.

**Definition 2.2.8** *Let $\mathcal{C}$ be the category CUMS or the category CUMS$\times$CUMS. A functor $\mathcal{F} : \mathcal{C} \to$ CUMS is called locally nonexpansive if for all objects $X, Y$ in $\mathcal{C}$ the function $\mathcal{F} \downarrow (X \xrightarrow{1} Y)$ that maps $f : X \xrightarrow{1} Y$ to $\mathcal{F}(f) : \mathcal{F}(X) \xrightarrow{1} \mathcal{F}(Y)$ is nonexpansive. The functor is called locally contractive if this function is contractive.*

*In other words, if for all objects $X, Y$ in $\mathcal{C}$ and for all nonexpansive mappings $f, g : X \xrightarrow{1} Y$*

$$d_{\mathcal{F}(X) \to \mathcal{F}(Y)}(\mathcal{F}(f), \mathcal{F}(g)) \leq d_{X \to Y}(f, g)$$

*then $\mathcal{F}$ is locally nonexpansive and if*

$$d_{\mathcal{F}(X) \to \mathcal{F}(Y)}(\mathcal{F}(f), \mathcal{F}(g)) \leq \alpha d_{X \to Y}(f, g)$$

*for $\alpha < 1$ then $\mathcal{F}$ is locally contractive.*

A locally contractive functor has a unique fixed point enabling its use in domain equations. Most of the functors treated so far are locally nonexpansive.

**Example 2.2.9** *The constant functor $Z$ : CUMS $\to$ $CUMS$ is locally contractive: For $X, Y \in$ CUMS, $f, g : X \xrightarrow{1} Y$ we have*

$$d_{Z \to Z}(Z(f), Z(g)) \;=\; d_{Z \to Z}(\mathrm{id}_Z, \mathrm{id}_Z) \;=\; 0 \;\leq\; \tfrac{1}{2} d_{X \to Y}(f, g)$$

*The functor $id_{\frac{1}{2}}$ : CUMS $\to$ CUMS is locally contractive: For $X, Y \in$ CUMS, $f, g : X \xrightarrow{1} Y$ and $x \in id_{\frac{1}{2}}(X)$ we have*

$$\begin{array}{rcll}
d_{id_{\frac{1}{2}}(Y)}(id_{\frac{1}{2}}(f)(x), id_{\frac{1}{2}}(g)(x)) & = & [\text{def. } id_{\frac{1}{2}}(f)] & d_{id_{\frac{1}{2}}(Y)}(f(x), g(x)) \\
& = & [\text{def. } d_{id_{\frac{1}{2}}(Y)}] & \frac{1}{2} d_Y(f(x), g(x)) \\
& \leq & [\text{def. } d_{X \to Y}] & \frac{1}{2} d_{X \to Y}(f, g)
\end{array}$$

*That the functors $\times : (CUMS \times CUMS) \to CUMS$ and $+ : (CUMS \times CUMS) \to CUMS$ are locally nonexpansive is directly clear from the definitions.*

A locally nonexpansive functor can be used in composition with a contractive functor: The composition of a locally nonexpansive functor and a locally contractive functor is again a locally contractive functor.

**Theorem 2.2.10** *Let $\mathcal{F}$ be a locally contractive functor $\mathcal{F} : CUMS \to CUMS$. Then the domain equation $\mathbb{P} \simeq \mathcal{F}(\mathbb{P})$ has a unique solution up to isomorphism.*

A proof of this theorem can be found in e.g. [177]. A domain equation based on a locally contractive functor has a unique solution up to isomorphism. This means that if both $\mathbb{P}$ and $\mathbb{Q}$ are solutions of the equation then they have the same structure, i.e. they are isomorphic. Domain equations thus specify the structure of a domain independent of the names of the actual elements of the domain. This is exactly what one wants in a semantical domain that is used to model program behavior where it is the structure of the objects that describes the behavior rather than the precise identity of the objects.

**Example 2.2.11** *The domain equation $\mathbb{P} \simeq A \times id_{\frac{1}{2}}(\mathbb{P})$ has a unique solution, namely infinite words with the Baire distance $(A^\omega, d_B)$.*

*The domain equation $\mathbb{P} \simeq A + A \times id_{\frac{1}{2}}(\mathbb{P})$ also has a unique solution, namely finite or infinite words with the Baire distance $(A^\infty, d_B)$.*

*The solution for the domain equation $\mathbb{P} \simeq \mathcal{P}_{nco}(A \times \mathbb{P})$ describes infinite trees (with identification of bisimilar trees) with arcs labeled by elements from $A$.*

# Chapter 3

# Modeling probabilistic choice

## 3.1 Introduction

In this chapter the control flow aspects of discrete probabilistic choice are studied. To this end a schematic language $\mathcal{L}_p$ containing the operator $\oplus_\rho$ is introduced. The operator $\oplus_\rho$ denotes a probabilistic choice between two alternatives. Each probabilistic choice can be thought of as flipping a coin or throwing a die: All probabilistic choices are assumed to be made independently of any choices that have been made before, and of any actions that may have occurred. The outcome of a single probabilistic choice cannot be predicted. The probability gives information about how often each alternative is selected in the long run, i.e. the relative frequencies of the possible outcomes. When modeling a fair coin flip, for example, this will be done with a probabilistic choice with probability $\frac{1}{2}$ for heads and $\frac{1}{2}$ for tails. Before flipping the coin the outcome is not known. It is known, however, that when the coin is flipped often, approximately half of all the outcomes is heads.

As the control flow is the main concern, the language does not deal with data or the details of the basic steps of computation. The language is based on a set of atomic actions. The atomic actions represent the basic elements of the computation and as such are left without any further interpretation.

The behavior that can be observed when a program in $\mathcal{L}_p$ is executed is seen as the operational meaning of the program. As the atomic actions are left without further interpretation, the behavior that can be observed when executing an atomic action is the action itself. In general, more than one atomic action is performed when a program is executed; a sequence of actions can be observed. A single execution of the program is referred to as a run of the system. Thus the observable behavior for a run of the system is a sequence of actions.

The probability associated with a probabilistic choice predicts the relative frequencies of the outcomes. A sequence of probabilistic choices is subject to statistical laws like the law of large numbers. The probability associated with a choice can be estimated by repeating the choice and taking the relative frequency of an outcome as its probability. This means that the probability of an alternative of a probabilistic choice can actually be found by observing the system: For a single run of the system a sequence of actions can be observed. Using multiple runs a probability can be assigned to the sequence of actions.

The sequences of actions that a program can execute together with the probability of executing that sequence, is part of the observable behavior of a program. The probabilities that one can see are, however, not only the probabilities of the complete sequences that the execution of a program produces, but also the probabilities for other events like e.g. "the sequence of actions produced starts with action $a$ followed by action $b$". This event can be described by the set of sequences starting with $ab$. A set of sequences of a special form, like e.g. the set of all sequences starting with $ab$, is called an "observable event". Every sequence of actions, or more precisely every singleton set consisting of a sequence of actions, is also an observable event. By looking at the results of multiple runs of the system, a probability can be assigned to every observable event. Thus the observable behavior of a program consists of the probabilities of all observable events.

If an observable event consists of a finite or countable set of sequences, the probability of the observable event can be found from the probability of each single sequence. The sequences of actions produced when executing a program may, however, be infinite in length. The number of infinitely long sequences is too large to find the probability of an observable event from the probabilities of the individual sequences: A collection of sequences may have positive probability while all individual sequences have probability 0. To give the observable behavior of a program it is not sufficient to list the probability of each sequence of actions that the program may produce. Instead a measure over sequences is used to describe the meaning of a program. Besides the probabilities of the individual sequences, a measure also gives the probability for other observable events. Example 3.3.12 shows a program in $\mathcal{L}_P$ for which the extra information given by the measure is really needed.

In order to find the sequences of actions of a program, a transition system is given. The transition system is an abstract representation of a machine on which the program runs. In the transition system, a step which takes the abstract machine from one configuration to another and produces some observable behavior, is called a transition. If, for example, the observable behavior produced by the transition is an atomic action, the transition represents the execution of that atomic action by the machine.

As the probability of a certain alternative is seen as something which is observable, it must be incorporated in the observable behavior that the transition system produces. In the transition system for $\mathcal{L}_P$ each transition will produce a pair of a real number in $[0, 1]$ and an atomic action. The number gives the probability that in the current configuration this atomic action is executed. If several probabilistic choices have to be made before it is clear which action is executed, the making of these choices is not visible in the transition system, only the resulting action with its probability. A transition system which deals with probability in this way is called a generative transition system [92]. It is also possible to model the actual making of probabilistic choices in the transition system. A transition system which explicitly models the making of each probabilistic choice is called a stratified transition system. In a stratified transition system the observable behavior produced by a transition is either a probability or an atomic action. A stratified transition system for a language with probabilistic choice is given in the next chapter, section 4.3. The setting in that section does not allow for a generative transition system.

The transition system still contains a lot of information which is not actually observable:

e.g. the state of the abstract machine, given by a configuration in the transition system, cannot be observed. Only the actions the abstract machine produces can be seen. The (operational) meaning of a program is only the observable behavior, i.e. the probability of observable events. The operational semantics of a program gives exactly the observable behavior of the program.

The operational semantics is given as a function from programs to the operational domain. The definition of the operational semantics is based on the transition system. The operational domain, denoted $\mathbb{P}_o$, contains all possible observable behaviors of programs in $\mathcal{L}_P$. Recall that the observable behavior of a program consists of the probabilities of all observable events, i.e. of a measure over sequences of actions. The elements of a semantical domain such as the operational domain are called processes.

The programs in $\mathcal{L}_P$ may exhibit infinite behavior. To have a convenient way to deal with infinite behavior, a metric structure is given to the operational domain. Instead of seeing $\mathbb{P}_o$ as a set, $\mathbb{P}_o$ is defined to be a complete metric space. On a metric space, uniqueness of the fixed point of a contractive operator (using the well-known Banach fixed point theorem 2.1.9) is an important tool in definitions and equality proofs. Showing that two possibly infinite processes coincide is reduced to finding an operator which has both processes as a fixed point and showing that this operator is contractive.

The operational semantics is based on the observational behavior of an abstract machine. Although this describes the computational intuition of a program, it does not support checking properties the program must satisfy. To be able to check a property it would be useful to be able to split a program into parts, which are presumably easier to analyze. To this end a denotational semantics is given.

In the denotational view, every program denotes some value and the combination of program components using the syntactical operators should be the same as combining the values they denote. This means that the denotational semantics uses the compositionality principle: the meaning of a program can be found by composing the meanings of the parts of the program. Using compositionality, the analysis of a program can be split into the analysis of the parts of the program.

The denotational meaning of a program is an element of the denotational domain $\mathbb{P}_d$. Like the operational domain the denotational domain is given a metric structure. To compose meanings, a semantical operation is defined on $\mathbb{P}_d$ for every syntactical operator. The denotational semantics is based on these operations using the compositionality principle.

The denotational meaning of a program is often not the same as the operational meaning of that program. In particular, to allow composing meanings, it may be required to remember more about a program than only its observable behavior. (Two programs with the same observable behavior may act differently when placed in a context.) It should be possible to remove the extra information contained in the denotational meaning to obtain the operational meaning. An abstraction function is used to relate denotational meanings and operational meanings. As the operational semantics of a program describes the computational intuition of a program, the denotational semantics must be correct with respect to the operational semantics, i.e. when applying the abstraction function to the meaning given by the denotational semantics, one should obtain the meaning given by the operational semantics. If this is the case, the operational semantics is called an abstraction of the denotational semantics.

The remainder of this chapter is organized as follows: In section 3.2 several notions required in the chapter are introduced. In section 3.3 the language $\mathcal{L}_p$ is defined. The notion of a transition system is made formal and a transition system for the language $\mathcal{L}_p$ is provided. To construct the metric space of measures of sequences, the functor *Meas* which constructs the metric space of measures over a given space is defined and using *Meas* the operational domain $\mathbb{P}_o$ is constructed. Based on the transition system the operational semantics is given as a function from programs to the operational domain $\mathbb{P}_o$.

In section 3.4 the functor $\mathcal{MP}_f$, which constructs multisets over a given domain is introduced. Using $\mathcal{MP}_f$ the denotational domain $\mathbb{P}_d$ is constructed. On the domain $\mathbb{P}_d$, semantical versions of the syntactical operators of $\mathcal{L}_p$ are defined. Using the semantical operators, the denotational semantics of $\mathcal{L}_p$ is given.

In section 3.5 an abstraction function is introduced which relates denotational meanings and operational meanings. The operational semantics is shown to be an abstraction of the denotational semantics. Finally section 3.6 contains some concluding remarks and references to related work.

## 3.2 Mathematical preliminaries

### 3.2.1 Finite multisets

In this subsection a formal model for *finite multisets* is given and several important operations on multisets are defined. Informally multisets are sets in which elements may occur more than once. How often an element occurs is called the multiplicity of the element. A multiset is finite if it contains finitely many elements, each with a finite multiplicity.

The formal model of a multiset over a given set $S$ is a (partial) *labeling* of $\mathbb{N}$ with elements of $S$. The labeling is a function from $\mathbb{N}$ to $S + \{ * \}$, where $*$ is used to denote undefinedness of a label. The support of the labeling is the set of numbers not mapped to $*$. For example, the multiset containing $a$ twice and $b$ one can be modelled by assigning label $a$ to 1 and 2, $b$ to 3 and $*$ to all other numbers. (See also example 3.2.2 below.)

This way of modeling multisets is derived from event structures, see e.g. [198, 137], and pomsets, see e.g. [85, 98]. A more usual way of representing multisets over a set $S$ is as functions from $S$ to the natural numbers; assigning a multiplicity to each element. The representation as labelings is used here because it fits nicely with the introduction of a metric on multisets over metric spaces as is done in section 3.4.

**Definition 3.2.1** *Let $S$ be a set. The support of a function $f : \mathbb{N} \to S + \{ * \}$ is the subset of $\mathbb{N}$ mapped to $S$, i.e. $f^{-1}[S]$. A partial labeling of $\mathbb{N}$ with elements of $S$ is a function $\mathbb{N} \to S + \{ * \}$ with finite support. The set of all partial labelings of $\mathbb{N}$ with elements of $S$ is denoted by $\mathbb{L}(S)$ and ranged over by $L$.*

$$\mathbb{L}(S) \;=\; \mathbb{N} \to (S + \{*\}) \textit{ with finite support}$$

From here on, the elements of $\mathbb{L}(S)$ are simply referred to as labelings instead of as partial labelings of $\mathbb{N}$ with elements of $S$.

**Example 3.2.2** *The multiset over $\{a, b\}$ containing the element $a$ twice and the element $b$ once can be represented by the following labelings. The support of labeling $L_1$ is $\{1, 2, 3\}$ and the support of $L_2$ is $\{2, 4, 5\}$.*

$$\begin{array}{lcl} L_1(1) & = & a \\ L_1(2) & = & b \\ L_1(3) & = & a \end{array} \qquad \begin{array}{lcl} L_2(2) & = & b \\ L_2(4) & = & a \\ L_2(5) & = & a \end{array} \qquad \begin{array}{rcccccccc} L_1(x): & a & b & a & * & * & * \\ x: & 1 & 2 & 3 & 4 & 5 & \ldots \\ L_2(x): & * & b & * & a & a & * \end{array}$$

As example 3.2.2 already shows, several labelings can describe the same multiset. Two different labelings that describe the same multiset list the same elements, but for different numbers in $\mathbb{N}$. The relation $\sim$, given below, relates labelings describing the same multisets. A multiset is defined as a $\sim$-equivalence class of labelings.

In a multiset the order of the elements is not relevant, so the difference between e.g. the labelings $L_1$ and $L_2$ above should be abstracted away from. Formally two labelings $L_1$ and $L_2$ are related if there exists a bijection $\Phi$ on $\mathbb{N}$ for which $L_1 \circ \Phi = L_2$.

**Definition 3.2.3** *Two labelings $L_1, L_2 : \mathbb{N} \to (S + \{*\})$ are related, $L_1 \sim L_2$, if there exists a bijection $\Phi : \mathbb{N} \to \mathbb{N}$ such that $L_1 \circ \Phi = L_2$. A class of labelings with representative $L$ is denoted by $\overline{L}$.*

A labeling $L_1$ that describes the same multiset as the labeling $L_2$ yields the same elements of $S$ with the same multiplicity. The only possible difference is for which numbers in $\mathbb{N}$ the elements are obtained. The bijection $\Phi$ rearranges the numbers in $\mathbb{N}$ to remove this difference.

It is clear that $\sim$ is an equivalence relation on $\mathbb{L}(S)$. The formal definition of a multiset over a set $S$ can now be given.

**Definition 3.2.4** *For a set $S$, the set of (finite) multisets over $S$, denoted by $\mathcal{MP}_f(S)$, is given by:*

$$\mathcal{MP}_f(S) \quad = \quad \mathbb{L}(S)/\sim$$

The multiplicity of an element of a multiset is the number of times that the element occurs in the multiset. That an element $x$ occurs exactly $n$ times, for $n > 0$, in a multiset $M$ is denoted by $x \in_n M$. If an element $x$ occurs at least once in a multiset $x \in M$ is written.

When using multisets, one does not want to think in terms of the formal definition as equivalence classes of labelings. To allow a more intuitive way of thinking about multisets, the following notation is used to describe a multiset. The notation $\{\!|\, x_1, x_2, \ldots, x_n \,|\!\}$ is used for the multiset containing $x_1$ through $x_n$. Here $x_1$, $x_2$, …, $x_n$ are, not necessarily distinct, elements of $S$. If $M = \{\!|\, x_1, x_2, \ldots, x_n \,|\!\}$ then $\{\!|\, f(x) \mid x \in M \,|\!\}$ denotes the multiset $\{\!|\, f(x_1), f(x_2), \ldots, f(x_n) \,|\!\}$ and $\sum M = x_1 + x_2 + \ldots + x_n$ (when summation is defined on $S$).

**Definition 3.2.5**

*(a) Let $S$ be a set and let $M = \overline{L}$ a multiset over $S$. For $x \in S$, the multiplicity of $x$ in $M$ is $\#\{i \mid L(i) = x\}$. The following notation is used*

$$\begin{array}{rl} x \in_n M & \textit{if the multiplicity of } x \textit{ in } M \textit{ is } n > 0 \\ x \in M & \textit{if } x \in_n M \textit{ for some } n > 0 \end{array}$$

(b) *Let $S$ be a set and let $x_1, x_2, \ldots, x_n$ be, not necessarily distinct, elements of $S$. Define $L(i) = x_i$ for $i = 1, 2, \ldots, n$ and $L(i) = *$ for $i > n$, then*

$$\{\!|\, x_1, x_2, \ldots, x_n \,|\!\} \quad = \quad \overline{L}$$

(c) *Let $S_1, S_2$ be sets, $f : S_1 \to S_2$ and let $M = \overline{L}$ be a multiset over $S_1$. Then $\{\!|\, f(x) \mid x \in M \,|\!\}$ denotes a multiset over $S_2$,*

$$\{\!|\, f(x) \mid x \in M \,|\!\} \quad = \quad \overline{f \circ L}$$

(d) *Let $M = \overline{L}$ be a multiset over $\mathbb{R}$, the real numbers, and let $X \subseteq \mathbb{N}$ be the support of $L$. Then*

$$\sum M \quad = \quad \sum_{n \in X} L(n)$$

As multisets are equivalence classes of labelings, the definitions may not depend on the choice of representative of a class. It is, however, easy to check that the definitions give the same result for different labelings from the same class.

**Example 3.2.6**

(a) *The multiset described by $L_1$ and $L_2$ in example 3.2.2 above can be written as $\{\!|\, a, b, a \,|\!\}$.*

(b) *The multiplicity of $a$ in $\{\!|\, a, b, a \,|\!\}$ is 2, $a \in \{\!|\, a, b, a \,|\!\}$, $a \in_2 \{\!|\, a, b, a \,|\!\}$, $a \not\in_1 \{\!|\, a, b, a \,|\!\}$.*

(c) *Let $M = \{\!|\, \frac{1}{2}, \frac{1}{4}, \frac{1}{4} \,|\!\}$. Then $\{\!|\, x/2 \mid x \in M \,|\!\} = \{\!|\, \frac{1}{4}, \frac{1}{8}, \frac{1}{8} \,|\!\}$ and $\sum M = \frac{1}{2} + \frac{1}{4} + \frac{1}{4} = 1$.*

A basic operation on multisets is union, denoted by $\sqcup$. The union of the two multisets $\{\!|\, x_1, x_2, \ldots, x_n \,|\!\}$ and $\{\!|\, y_1, y_2, \ldots, y_m \,|\!\}$ is $\{\!|\, x_1, x_2, \ldots, x_n, y_1, y_2, \ldots, y_m \,|\!\}$.

**Definition 3.2.7** *Let $\overline{L_1}$ and $\overline{L_2}$ be two multisets over a set $S$. The union of the multisets is given by*

$$\overline{L_1} \sqcup \overline{L_2} = \overline{L_3}$$

*where $L_3(2n) = L_1(n)$ and $L_3(2n-1) = L_2(n)$ for all $n \in \mathbb{N}$.*

It is again easy to check that the definition does not depend on the choice of the representatives $L_1$ and $L_2$.

**Example 3.2.8** *The union of the multisets $\{\!|\, a, b, a \,|\!\}$ and $\{\!|\, c, a \,|\!\}$ is given by $\{\!|\, a, b, a \,|\!\} \sqcup \{\!|\, c, a \,|\!\} = \{\!|\, a, a, a, b, c \,|\!\}$.*

## 3.3 The syntax and operational semantics of $\mathcal{L}_p$

In this section the syntax of the language $\mathcal{L}_p$ is given. The elements of the language $\mathcal{L}_p$ are called *programs*. A transition system $\mathcal{T}_p$ which abstractly describes the execution of the programs in $\mathcal{L}_p$ is defined. A functor *Meas* is introduced to build the space of all measures over a given space. The operational domain $\mathbb{P}_o$ which contains all possible meanings of programs is defined using the functor *Meas*. Finally the operational semantics $\mathcal{O}$ is defined as a function from programs to the domain $\mathbb{P}_o$ based on the transition system $\mathcal{T}_p$.

### 3.3.1 The syntax of the language $\mathcal{L}_p$

A program in $\mathcal{L}_p$ consists of a *declaration* in *Decl* and a *statement* in *Stat*. A typical declaration is denoted by $D$ and a typical statement by $s$.

The statements in *Stat* are built from a set of atomic actions *Act* which is ranged over by $a$ and a set of procedure variables *PVar* ranged over by $x$. The atomic actions represent the primitive elements of the computation, and are left without further interpretation. The meaning of an action $a$ is simply $a$ itself. A procedure variable represents a procedure to be called. The body of a procedure is the statement that should be executed as a result of a call of the procedure. A declaration $D$ gives the body $D(x)$ for each procedure variable $x$.

Statements are built from the atomic actions and procedure variables by constructing combinations using the syntactical operators $\oplus_\rho$ and $;$.

**Definition 3.3.1**

*(a) The set of statements Stat, ranged over by s, is given by*

$$s \quad ::= \quad a \mid x \mid s\,;s \mid s \oplus_\rho s$$

*where $\rho \in (0,1)$.*

*(b) The set of guarded statements GStat, ranged over by g, is given by*

$$g \quad ::= \quad a \mid g\,;s \mid g \oplus_\rho g$$

*where $\rho \in (0,1)$.*

*(c) The set of declarations Decl, ranged over by D, is given by*

$$\mathit{Decl} \quad = \quad \mathit{PVar} \to \mathit{GStat}$$

*(d) The language $\mathcal{L}_p$ is given by*

$$\mathcal{L}_p \quad = \quad \mathit{Decl} \times \mathit{Stat}$$

The statements of the language are combinations of atomic actions and procedure variables using the syntactical operators $;$ and $\oplus_\rho$. The statements which are built using the operators $;$ and $\oplus_\rho$ are assumed to be enclosed in brackets. One should, therefore, actually write $(s\,;s)$ instead of $s\,;s$ and $(s \oplus_\rho s)$ instead of $s \oplus_\rho s$. However, the brackets are omitted when no confusion is possible.

The operator $;$ denotes sequential composition. The statement $s_1\,;s_2$ behaves like $s_1$ until $s_1$ terminates after which $s_1\,;s_2$ continues by behaving like $s_2$. The operator $\oplus_\rho$ denotes probabilistic choice. The argument $\rho$ denotes the probability that the first alternative is chosen and is assumed to be strictly between 0 and 1. The execution of the statement $s_1 \oplus_\rho s_2$ starts with making a probabilistic choice. With probability $\rho$ the first statement, $s_1$, is selected and executed. With the remaining probability, $1-\rho$, the second statement, $s_2$, is selected and executed.

A declaration $D$ gives the body $D(x)$ for a procedure $x$. The body of a procedure cannot be any statement, but must be a guarded statement. A guarded statement guarantees that at least one atomic action is done before some procedure is called. The recursive definition of a procedure can be used to specify an infinite process. The procedure $x$ with body $D(x) = a; x$ will produce an infinite sequence of $a$'s. The restriction to guarded statements as possible body for a procedure prevents ambiguous definitions like $D(x) = x$. The meaning of a procedure defined by $D(x) = x$ is not well-defined. Guarded recursion is a common way to deal with this problem. With guarded recursion, the body of each procedure must be a guarded statement.

When executing a program in $\mathcal{L}_p$ the declaration $D$ does not change, as is the case with all languages considered in this thesis. In the rest of this chapter one fixed declaration $D$ is assumed and $D$ is dropped from the notation, e.g. $s \in \mathcal{L}_p$ is written instead of $(D, s) \in \mathcal{L}_p$.

### 3.3.2 Transition system specification

A transition system $\mathcal{T} = (\mathit{Conf}, \mathit{Lab}, \rightarrow)$ is an abstract representation of a machine. When a program "runs" on the abstract machine, the machine takes steps between configurations in *Conf* producing some observable behavior. Which steps can be taken is given by the transition relation $\rightarrow$. The behavior of a step which is observed is an element of *Lab*.

The transition system for each statement in the language is defined using a transition system specification. A transition system specification $\mathcal{T} = (\mathit{Conf}, \mathit{Lab}, \rightarrow, \mathit{Spec})$ is a structured way of defining transition systems. The transition system specification defines the transition relation $\rightarrow$ by means of a specification *Spec* consisting of a set of axioms and rules. The transition relation specified by *Spec* consists of those transitions which can be derived from the axioms in *Spec* using the rules in *Spec*. With some abuse of terminology the transition system specification will simply be called the transition system: No distinction is made between the transition system specification and the transition system specified by the transition system specification.

The idea of a transition system specification comes from structured operational semantics [173]. The transition system specifications used in this thesis, extend the format used in structured operational semantics to be able to deal with probabilistic choice.

**Definition 3.3.2**

(a) *A transition system specification $\mathcal{T}$, from now on simply called a transition system, is a four-tuple $\mathcal{T} = (\mathit{Conf}, \mathit{Lab}, \rightarrow, \mathit{Spec})$ where*

- *Conf is a set of* configurations*, ranged over by $c$,*
- *Lab is a set of* transition labels*, ranged over by $\theta$,*
- *$\rightarrow$, the transition relation, is the subset of $\mathit{Conf} \times \mathit{Lab} \times \mathit{Conf}$ satisfying Spec as described below. An element of $\mathit{Conf} \times \mathit{Lab} \times \mathit{Conf}$ is called a transition.*
- *The specification, Spec, is a set of* axioms *and* rules*, containing at least one axiom. An axiom is a construct of the form $c \xrightarrow{\theta} c'$ (name). It specifies that the tuple $(c, \theta, c')$ belongs to $\rightarrow$.*

*A rule is a construct:*

$$\frac{c_1 \xrightarrow{\theta_1} c_1' \ \ldots \ c_k \xrightarrow{\theta_k} c_k'}{c \xrightarrow{\theta} c'} \quad \textit{(name)}$$

*It specifies that if* $(c_i, \theta_i, c_i') \in \ \rightarrow$ *$(i = 1, \ldots, k)$ then also* $(c, \theta, c') \in \ \rightarrow$. *Here* $c_1 \xrightarrow{\theta_1} c_1'$ *through* $c_k \xrightarrow{\theta_k} c_k'$ *are called premises of the rule and* $c \xrightarrow{\theta} c'$ *is called the conclusion of the rule. An axiom can be seen as a rule with no premise.*

*(b) A proof tree for a specification is a finite tree in which every node is the conclusion of a rule in Spec and its sub-nodes are the premises of this rule. The leaves of the tree are axioms. The transition at the root of a proof tree is said to be derived from the specification (by the proof tree). Proof trees are depicted with the root of the tree at the bottom.*

*(c) The transition relation* $\rightarrow$ *is the subset of Conf* $\times$ *Lab* $\times$ *Conf consisting exactly of the transitions that can be derived from the specification, i.e. for which there is a proof tree based on Spec.*

*(d) The multiplicity of a transition* $c \xrightarrow{\theta} c'$ *is the number of different proof trees that exist for the transition. That* $(c, \theta, c')$ *belongs to* $\rightarrow$ *and has multiplicity $n$ is denoted by* $c \xrightarrow{\theta}_n c'$.

The parts of a transition system specification are as usual. Often the transition relation is defined as "the smallest subset of *Conf* $\times$ *Lab* $\times$ *Conf* which satisfies all axioms and rules in *Spec*". In all cases considered this definition results in the same transition relation $\rightarrow$. The explicit reference to proof trees is needed in order to specify the multiplicity of a transition. The multiplicity of a transition can be important for probabilistic transitions, as is illustrated in example 3.3.4 below. This example also shows two simple proof trees. All proof trees in this chapter are linear. More complicated (branching) proof trees appear e.g. in section 4.3.

As each transition in the set $\rightarrow$ has a multiplicity assigned, the set $\rightarrow$ can also be seen as a multiset by including every transition with the right multiplicity, i.e. $(c, \theta, c') \in_n \ \rightarrow$ exactly when $c \xrightarrow{\theta}_n c'$. In this chapter, however, the transition relation $\rightarrow$ will be seen as a set and the multiplicities are written explicitly to clearly show where they play a role.

If several rules have the same premises, the shorthand notation

$$\begin{array}{ll} \dfrac{c_1 \xrightarrow{\theta_1} c_1' \ \ldots \ c_k \xrightarrow{\theta_k} c_k'}{\bar{c}_1 \xrightarrow{\bar{\theta}_1} \bar{c}_1'} & \text{(name 1)} \\ \qquad\quad \ldots & \\ \qquad\ \ \bar{c}_n \xrightarrow{\bar{\theta}_n} \bar{c}_n' & \text{(name } n\text{)} \end{array}$$

is used for the $n$ rules

$$\frac{c_1 \xrightarrow{\theta_1} c_1' \ \ldots \ c_k \xrightarrow{\theta_k} c_k'}{\bar{c}_i \xrightarrow{\bar{\theta}_i} \bar{c}_i'} \ \text{(name } i\text{)} \qquad (i = 1, \ldots, n)$$

### 3.3.3 The transition system $\mathcal{T}_p$

The transition system for the language $\mathcal{L}_p$ is called $\mathcal{T}_p$. All the information required to describe the state of an execution of a program in $\mathcal{L}_p$ is the part of the program that remains to be executed. A *resumption* is used to describe the remainder of a program. The set of resumptions is denoted by *Res* and ranged over by $r$. The remainder of a program is either another program or nothing, in case the execution is finished. A resumption is either a statement $s$ in *Stat* or a special symbol E denoting a finished computation:

$$r \quad ::= \quad s \mid \mathrm{E}$$

A configuration in $\mathcal{T}_p$ is a resumption $r$ together with a declaration $D$, i.e. $\mathit{Conf} = \mathit{Decl} \times \mathit{Res}$. As with programs, the declaration part is dropped from the notation as a single declaration is assumed to be fixed.

As described in the introduction the observable behavior produced by a transition in a probabilistic transition system consists of an action together with the probability that the transition occurs. The observables in the transition system $\mathcal{T}_p$ are probability-action pairs. The set of all probability-action pairs $\mathit{PAct} = [0,1] \times \mathit{Act}$ is ranged over by $\alpha$. Probabilities in $[0,1]$ are ranged over by $\rho, \sigma$ and atomic actions are ranged over by $a$. Alternatively an element of *PAct* can be written as $\rho \cdot a$, denoting action $a$ with probability $\rho$. Note that the notation $\cdot$ is used for three different operations: For concatenation of words as in $w \cdot w'$, for multiplication of numbers as in $\rho \cdot \sigma$ and for forming (probability,action)-pairs as in $\rho \cdot a$. The first two are usually suppressed in the notation; $ww'$ is written for the word $w$ followed by the word $w'$ and $\rho\sigma$ is written for the product of $\rho$ and $\sigma$.

**Definition 3.3.3** *The transition system $\mathcal{T}_p$ is given by $\mathcal{T}_p = (\mathit{Conf}, \mathit{PAct}, \rightarrow, \mathit{Spec})$. A transition $(r, \alpha, r') \in \rightarrow$ is written as $r \xrightarrow{\alpha} r'$.*
*The specification Spec consists of*

- $$a \xrightarrow{1\cdot a} \mathrm{E} \qquad \textit{(Act)}$$

- $$\frac{s_1 \xrightarrow{\alpha} r}{s_1; s_2 \xrightarrow{\alpha} r; s_2} \qquad \textit{(Seq)}$$

- $$\frac{D(x) \xrightarrow{\alpha} r}{x \xrightarrow{\alpha} r} \qquad \textit{(Rec)}$$

- $$\frac{s_1 \xrightarrow{\sigma\cdot a} s}{\begin{array}{ll} s_1 \oplus_\rho s_2 \xrightarrow{\rho\sigma\cdot a} s & \textit{(Chance 1)} \\ s_2 \oplus_\rho s_1 \xrightarrow{(1-\rho)\sigma\cdot a} s & \textit{(Chance 2)} \end{array}}$$

*where $r; s_2$ in rule (Seq) should be read as $s_2$ if $r = \mathrm{E}$.*

The axioms and rules given above are actually axiom-schemas and rule-schemas. The axiom-schema (Act) indicates that the axiom $a \xrightarrow{1\cdot a} \mathrm{E}$ (Act) is in *Spec* for every action $a \in \mathit{Act}$. Below the term "the axiom (Act)" is used when formally one should refer to the axiom (Act) for a specific action $a \in \mathit{Act}$, the action-schema (Act), or the set of axioms with name (Act). The same is done for the rule-schemas: No distinction is made between a rule-schema and the actual rules it represents.

The axiom (Act) expresses that the statement consisting of only the action $a$ takes an $a$ transition with probability 1 after which it is finished. The Rule (Rec) takes care of recursion by body replacement. To execute a procedure $x$, the body $D(x)$ of the procedure has to be executed. The rule (Seq) states that $s_1 \,;s_2$ behaves like $s_1$ until $s_1$ is done (the case that $r = \mathrm{E}$) and after that behaves like $s_2$ (since $\mathrm{E}\,; s_2 = s_2$). The statement $s_1 \oplus_\rho s_2$ acts like $s_1$ with probability $\rho$; the statement $s_2 \oplus_\rho s_1$ acts like $s_1$ with probability $1-\rho$.

With the rules for probabilistic choice there is a subtle point that has to be dealt with. It may be possible to derive the same transition more than once. The easiest example is $s = a \oplus_{\frac{1}{2}} a$. The transition $s \xrightarrow{\frac{1}{2}\cdot a} \mathrm{E}$ can be derived twice. (See example 3.3.4 below.) This is not the same as having the transition once, because the total probability of $s$ performing an action should be 1, not $\frac{1}{2}$. This requires taking the multiplicity of a transition into account when defining the operational semantics.

**Example 3.3.4** *Let $s$ be the statement $a \oplus_{\frac{1}{2}} a$. There exist two different proof trees for the specification in $\mathcal{T}_p$ deriving the same transition:*

$$\dfrac{\dfrac{}{a \xrightarrow{1\cdot a} \mathrm{E}}\ \textit{(Act)}}{a \oplus_{\frac{1}{2}} a \xrightarrow{\frac{1}{2}\cdot a} \mathrm{E}}\ \textit{(Chance 1)} \qquad\qquad \dfrac{\dfrac{}{a \xrightarrow{1\cdot a} \mathrm{E}}\ \textit{(Act)}}{a \oplus_{\frac{1}{2}} a \xrightarrow{\frac{1}{2}\cdot a} \mathrm{E}}\ \textit{(Chance 2)}$$

*Thus the multiplicity of the transition $s \xrightarrow{\frac{1}{2}\cdot a} s$ is 2, i.e. $s \xrightarrow{\frac{1}{2}\cdot a}_2 s$.*

Example 3.3.4 shows a transition with multiplicity two. Using the multiplicity of the transition is essential to prevent "losing probability" when two alternatives of a probabilistic choice are the same and happen with the same probability. The total probability of taking an action is 1 for every statement when the multiplicities are taken into account. This fact is proven in lemma 3.3.9 below.

**Example 3.3.5** *Let $D(x) = (a \oplus_\rho b)\,; c$. We have $a \xrightarrow{1\cdot a} \mathrm{E}$, $b \xrightarrow{1\cdot b} \mathrm{E}$ and $c \xrightarrow{1\cdot c} \mathrm{E}$ by axiom (Act) so*

$$\begin{array}{rcll}
(a \oplus_\rho b) & \xrightarrow{\rho\cdot a} & \mathrm{E} & \textit{rule (Chance 1)} \\
 & \xrightarrow{(1-\rho)\cdot b} & \mathrm{E} & \textit{rule (Chance 2)} \\
(a \oplus_\rho b)\,; c & \xrightarrow{\rho\cdot a} & c & \textit{rule (Seq)} \\
 & \xrightarrow{(1-\rho)\cdot b} & c & \textit{rule (Seq)} \\
x & \xrightarrow{\rho\cdot a} & c & \textit{rule (Rec)} \\
 & \xrightarrow{(1-\rho)\cdot b} & c & \textit{rule (Rec)} \\
c & \xrightarrow{1\cdot c} & \mathrm{E} & \textit{rule (Act)}
\end{array}$$

$$\begin{tikzcd}
 & x \arrow[dl, "\rho \cdot a"'] \arrow[dr, "(1-\rho)\cdot b"] & \\
c \arrow[d, "1\cdot c"'] & & c \arrow[d, "1\cdot c"] \\
\mathrm{E} & & \mathrm{E}
\end{tikzcd}$$

For a statement $s$ the transition that can be taken during the execution of $s$ can be represented as a tree. The root of the tree is $s$ itself. The first level of the tree are the transitions of $s$. The second level of the tree are the transitions of the resumptions that $s$ can reach, etc. The tree that is generated in this way is called the *transition tree* for $s$. Transition trees are depicted with the root of the tree at the top. In example 3.3.5 above the transition tree for the statement $x$ is shown. Often the names of the resumptions labeling the nodes of the tree are removed, keeping only the observable behavior of the transitions (see e.g. example 3.3.10).

**Definition 3.3.6** *The successor (multi)set of a resumption $r$ denoted by $Suc(r)$, is the multiset of possible actions that $r$ can execute, together with their corresponding resumptions. As $\mathrm{E}$ can take no actions $Suc(\mathrm{E}) = \emptyset$. For a statement $s$, $\langle \alpha, r\rangle \in_n Suc(s)$ exactly when $s \xrightarrow{\alpha}_n r$.*

The successor set $Suc(s)$ of $s$ gives all possible steps for the statement $s$ with the correct multiplicity. If $\rightarrow$ is seen as a multiset, as explained directly below definition 3.3.2, the successor set of a statement can also be described by $Suc(s) = \{\!|\, \langle \alpha, r\rangle \mid s \xrightarrow{\alpha} r \,|\!\}$.

**Example 3.3.7**

*(a) (See example 3.3.4.) Let $s = a \oplus_{\frac{1}{2}} a$. Then $s \xrightarrow{\frac{1}{2}\cdot a}_2 \mathrm{E}$ so $Suc(s) = \{\!|\, \langle \frac{1}{2}\cdot a, \mathrm{E}\rangle, \langle \frac{1}{2}\cdot a, \mathrm{E}\rangle \,|\!\}$.*

*(b) (See example 3.3.5.) Let $s = x$ with $D(x) = (a \oplus_\rho b); c$. Then $s \xrightarrow{\rho\cdot a} c$ and $s \xrightarrow{(1-\rho)\cdot b} c$ so $Suc(s) = \{\!|\, \langle \rho\cdot a, c\rangle, \langle (1-\rho)\cdot b, c\rangle \,|\!\}$.*

For a statement $s$ the successor set $Suc(s)$ is specified in definition 3.3.6 by giving its elements with their multiplicity. To show that this indeed specifies a finite multiset, the number of successors of $s$ has to be shown to be finite and each successor must have a finite multiplicity. Also, as the probability to do some step is 1 for every statement $s$, the probabilities in $Suc(s)$ should add up to 1. Lemma 3.3.9 below shows these properties. The lemma is proven by *weight induction*: In a transition $s \xrightarrow{a} s'$ the statement $s$ on the left-hand side of the arrow is referred to as the starting statement of the transition. The starting statement $D(x)$ in the premise of rule (Rec) is not syntactically simpler than the starting statement $x$ in the conclusion of the rule. Structural induction cannot be used to show facts such as: For every transition there is a limit to the height that any proof tree for the transition can have. Also for other properties, structural induction is not sufficient. The restriction to guarded recursion prevents problems with the rule (Rec). The procedure variable $x$ can still be seen as "more complex" than the guarded statement $D(x)$. The complexity of a statement can be formally defined by giving a weight to each resumption using a weight function $wgt$. Induction on the weight of a statement, called weight induction, replaces structural induction.

**Definition 3.3.8** *The function $wgt\colon Res \rightarrow \mathbb{N}$ is given by*

$$\begin{aligned} wgt(\mathrm{E}) &= 0 \\ wgt(a) &= 1 \end{aligned}$$

$$\begin{array}{rcl} wgt(x) & = & wgt(D(x)) + 1 \\ wgt(s_1\,;s_2) & = & wgt(s_1) + 1 \\ wgt(s_1 \oplus_\rho s_2) & = & wgt(s_1) + wgt(s_2) + 1 \end{array}$$

That the weight function *wgt* is well-defined is easy to see by structural induction, first on guarded statements and then on all resumptions. For every rule in $\mathcal{T}_p$ the starting resumptions of the premises of the rule are less complex, i.e. have a lower weight, than the starting resumption of the conclusion of the rule. This makes weight induction a useful way to prove properties about the transition system. Weight induction is also useful when specifying the denotational semantics and when comparing the operational and denotational semantics.

**Lemma 3.3.9** *For $\mathcal{T}_p$ the following holds.*

*(a) $\mathcal{T}_p$ is finitely branching, that is, for all resumptions $r \in Res$, $Suc(r)$ is a finite multiset.*

*(b) For all statements $s \in Stat$ the sum of the probabilities in $Suc(s)$ is 1.*

**Proof** By induction on the weight of the resumption $r$ for (a) and the weight of the statement $s$ for (b).

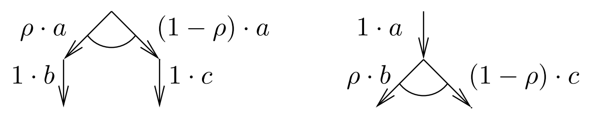
- The successor set of E is empty so this set is definitely finite.
- The statement $a$ has only one transition: $Suc(a) = \{\!|\, \langle 1 \cdot a, \mathrm{E}\rangle \,|\!\}$. Clearly $Suc(a)$ is finite and the sum of the probabilities is 1.
- The successor set of $x$, $Suc(x)$, is exactly the same as $Suc(D(x))$ and the weight of $D(x)$ is less than that of $x$. Therefore, we are done by induction.
- As is clear from rule (Seq) and the absence of any other rules for $s_1; s_2$, the successor set of $s_1; s_2$ is equal to $\{\!|\, \langle \alpha, r; s_2\rangle \mid \langle \alpha, r\rangle \in Suc(s_1) \,|\!\}$. As $wgt(s_1) < wgt(s_1; s_2)$, $Suc(s_1)$ is finite and the probabilities in $Suc(s_1)$ sum up to 1 by induction. The same properties follow immediately for $Suc(s_1; s_2)$.
- As can be seen from the rules (Chance), the successor set of $s_1 \oplus_\rho s_2$ is the union of $\{\!|\, \langle \rho\sigma \cdot a, r\rangle \mid \langle \sigma \cdot a, r\rangle \in Suc(s_1) \,|\!\}$ and $\{\!|\, \langle (1-\rho)\sigma \cdot a, r\rangle \mid \langle \sigma \cdot a, r\rangle \in Suc(s_2) \,|\!\}$. As the weights of $s_1$ and $s_2$ are both less than the weight of $s_1; s_2$, by induction both $Suc(s_1)$ and $Suc(s_2)$ are finite. But then clearly also $\{\!|\, \langle \rho\sigma \cdot a, r\rangle \mid \langle \sigma \cdot a, r\rangle \in Suc(s_1) \,|\!\}$ and $\{\!|\, \langle (1-\rho)\sigma \cdot a, r\rangle \mid \langle \sigma \cdot a, r\rangle \in Suc(s_2) \,|\!\}$ are finite and so is their union, $Suc(s_1 \oplus_\rho s_2)$. Also by induction, the probabilities sum up to 1 in $Suc(s_1)$ so they sum up to $\rho$ in $\{\!|\, \langle \rho\sigma \cdot a, r\rangle \mid \langle \sigma \cdot a, r\rangle \in Suc(s_1) \,|\!\}$ and the probabilities in $Suc(s_2)$ sum up to 1 so in $\{\!|\, \langle (1-\rho)\sigma \cdot a, r\rangle \mid \langle \sigma \cdot a, r\rangle \in Suc(s_2) \,|\!\}$ the probabilities sum up to $1-\rho$. Therefore, in $Suc(s_1 \oplus_\rho s_2)$ the probabilities sum up to $\rho + (1-\rho) = 1$. □

The successor set describes the possible first steps of a resumption. These steps are obtained from the transition system. Before the operational semantics, which describes the complete observable behavior of a program, can be given, the operational domain $\mathbb{P}_o$ is introduced in the next subsection.

### 3.3.4 The operational domain; the functor *Meas*

As described in the introduction, the transition system contains information that cannot be observed. For the operational semantics, which only gives observable behavior, this information should be removed. The configuration of the abstract machine during the execution of a program is one thing that cannot be seen. Another thing which cannot be observed is when a probabilistic choice is made. The information about the configurations can be removed by looking at *abstract transition trees.* An abstract transition tree is a transition tree in which the configurations labeling the nodes have been removed. The two statements in example 3.3.10 below produce the same observable behavior but have different abstract transition trees, showing that the abstract transition trees still contain information which is not observable.

**Example 3.3.10** *The statements $s_1 = a\,;(b \oplus_\rho c)$ and $s_2 = (a\,;c) \oplus_\rho (a\,;b)$ have the same observable behavior (sequence ac with probability $\rho$ and sequence bc with probability $1-\rho$). The abstract transition trees for both statements are different.*



$s = (a;b) \oplus_\rho (a;c)$ $\qquad$ $s = a;(b \oplus_\rho c)$

The difference between the statements in this example is the moment the probabilistic choice is made. The observable behavior of both statements is sequence $ab$ with probability $\rho$ and sequence $ac$ with probability $1-\rho$. The fact that the probabilistic choice is made later for the second statement cannot be observed from sequences of actions that are produced, i.e. the moment of choice is not part of the observable behavior of a statement. To obtain only the observable behavior, the moment of choice has to be abstracted away from. To this end the domain of behaviors is given as a *linear domain.* In a linear domain, the moment of choice is removed. A linear representation of the meaning of a program is basically a collection of sequences where each sequence corresponds to a possible execution of the program.

**Definition 3.3.11** *The set of all finite sequences of action in Act is denoted by $Act^\star$, the set all infinite sequences of action in Act is denoted by $Act^\omega$. The set $Act^\infty$ is the set of finite or infinite sequences, $Act^\infty = Act^\star \cup Act^\omega$. The distance on each of these sets is the Baire metric (see example 2.1.3).*

*Furthermore, $\epsilon$ is written for the the empty sequence, and $w \cdot w'$ for the concatenation of the sequences $w$ and $w'$. The concatenation $w \cdot S$ of a sequence $w$ with a set of sequences $S$ is given by $w \cdot S = \{\, w \cdot w' \mid w' \in S \,\}$.*

The operator $\cdot$ for the concatenation is usually omitted from the notation, e.g. $ww'$ is written instead of $w \cdot w'$.

In a probabilistic setting, the probability of the sequences should also be incorporated. This cannot be done by simply adding a probability to each sequence as can be seen from the statement $s$ in the following example.

**Example 3.3.12** *Let the statement $s$ be given by:*

$$\begin{array}{rcl} s & = & x \\ D(x) & = & (a \oplus_{\frac{1}{2}} b)\,;x \end{array}$$

*The statement $s$ can produce any infinite sequence of $a$'s and $b$'s. For a given infinite sequence, the probability to execute the first action of the sequence is $\frac{1}{2}$, the probability to execute the first two actions of the sequence is $\frac{1}{4}$, etc. Thus the probability of executing the entire infinite sequence is $0$.*

*The probability of executing some sequence starting with $ab$ is $\frac{1}{4}$, but the probability for each of these sequences separately is $0$.*

There are too many possible sequences of actions to let the probability of single sequences describe the probability of all "observable events". The next question is, what exactly is an observable event? Clearly "executing some sequence starting with *ab*" is something that can be observed, i.e. this is an observable event. The probability of this event can be found by running the program often and checking how often it starts with *ab*. An other event which can be observed is "exactly the sequence *ab* is executed". These events can be reformulated as "the sequence produced is in the set $\{\, w \in Act^{\infty} \mid w \text{ starts with } ab \,\} = abAct^{\infty}$" and "the sequence produced is in the set $\{\, ab \,\}$". The events can be described by a set of sequences. The sets $wAct^{\infty}$ and $\{\, w \,\}$ for $w \in Act^{\star}$ are exactly the open balls in $Act^{\infty}$. The event "the sequence produced does not start with *ab*" is also observable. In general, the complement of an observable event is also an observable event. Similarly one can observe the event "the sequence produced starts with *ab* or with *aa*". In general, the (countable) union of observable events is again an observable event. So the collection of observable events should contain the open balls and be closed under complement and countable union. A collection of sets that satisfies these properties are the *Borel sets* over $Act^{\infty}$.

**Definition 3.3.13** *Let $M$ be an ultrametric space.*

*(a) A $\sigma$-algebra $\mathcal{A}$ over $M$ is a collection of subsets of $M$ which is closed under complement and countable union.*

*(b) The $\sigma$-algebra over $M$ generated by a collection of subsets of $M$ is the least $\sigma$-algebra containing all the sets in the collection.*

*(c) The collection $\mathcal{B}(M)$ of Borel sets over $M$ is the $\sigma$-algebra generated by the open subsets of $M$.*

The notion of an observable event can now be made formal: an *observable event* is a Borel set. As the probability of an observable event cannot be found by adding the probabilities of all sequences in the observable event, the probability for all observable events has to be given directly. This can be done by a function from the observable events to $[0, 1]$. The function has to satisfy the calculation rules for probabilities, e.g. the probability assigned to the union of nonoverlaping events should be the sum of the probabilities assigned to the events. A function like this is called a *measure*.

In the definition below the functor *Meas* is introduced that yields the space of all measures over a given ultrametric space.

**Definition 3.3.14** *Let $M$ be an ultrametric space.*

*(a) A measure on $M$, or more precisely a Borel probability measure on $M$, is a function $\mu : \mathcal{B}(M) \to [0,1]$ such that $\mu(M) = 1$ and $\mu$ is $\sigma$-additive, i.e. $\mu(\cup_{i\in I} A_i) = \sum_{i\in I} \mu(A_i)$ for any countable collection of pairwise disjoint sets $A_i$ (for $i \in I$).*

*(b) A measure $\mu$ is said to have compact support if $\mu$ vanishes outside some compact set $K$, i.e. $\mu(M \setminus K) = 0$ thus $\mu(K) = 1$. In this case the support* $\mathrm{spt}(\mu)$ *of $\mu$ is defined as the smallest compact set $K$ for which $\mu(M \setminus K) = 0$.*

*(c) Meas$(M)$ denotes the space of all measures with compact support on $M$.*

*(d) The distance $d_{\mathit{Meas}}$ on Meas$(M)$ is defined by*

$$d_{\mathit{Meas}}(\mu, \nu) = \inf\{\, \epsilon > 0 \mid \forall x \in M : \mu(B_\epsilon(x)) = \nu(B_\epsilon(x)) \,\}$$

Throughout this chapter the term measure refers to a Borel probability measure with compact support. The support $\mathrm{spt}(\mu)$ of a measure $\mu$ is obtained as the intersection of all compact sets $K$ with $\mu(K) = 1$. That $d_{\mathit{Meas}}$ is indeed a metric on $\mathit{Meas}(M)$ is shown in lemma 3.3.16. For technical reasons it is in general not possible to give the probability for any set (see e.g. [102]). A measure only gives the probability for Borel sets.

The Borel sets over $Act^\infty$ contain all finite sets, all sets of the form $wAct^\infty$ for $w \in Act^\star$ but also all sets that can be obtained by be taking complements and unions. It is sufficient, however, to know the probabilities for the events of the form $\{\, w \,\}$ and $wAct^\infty$ for $w \in Act^\star$. (Recall that these are the open balls.) If these probabilities are known, the probabilities of the other events are fixed by general properties of probabilities that a measure must satisfy.

**Lemma 3.3.15** *If two (compact support Borel probability) measures $\mu$ and $\mu'$ coincide on the open balls of an ultrametric space $M$, they are the same.*

**Proof** It is first shown that the two measures coincide on all sets in $\mathcal{O}_\epsilon = \{\, O \subseteq M \mid x \in O \Rightarrow B_\epsilon(x) \subseteq O \,\}$ (for all $\epsilon > 0$). A basic result from measure theory can then be used to show that $\mu$ and $\mu'$ are the same.

For a given $\epsilon > 0$ take a finite collection of points $x_1, \ldots x_n$ such that both $\mu$ and $\mu'$ vanish outside $B_\epsilon(x_1) \cup \ldots \cup B_\epsilon(x_n)$. Clearly such a collection of points exists as the collection of all $\epsilon$-balls is a cover of $M$ and both $\mu$ and $\mu'$ vanish outside some compact set which must have a finite subcover. The balls $B_\epsilon(x_1), \ldots, B_\epsilon(x_n)$ can be assumed to be pairwise disjoint because if two of these balls intersect, they must be equal due to ultrametricity and one of the two can be removed. Also, for a given set $O \in \mathcal{O}_\epsilon$, a ball $B_\epsilon(x)$ is either completely inside $O$ or completely outside $O$.

$$\begin{array}{rcl}
\mu(O) & = & \mu(\cup_{i=1}^n B_\epsilon(x_i) \cap O) \\
& = & \mu(\cup\{\, B_\epsilon(x_i) \mid x_i \in O, i = 1, \ldots, n \,\}) \\
& = & \sum\{\, \mu(B_\epsilon(x_i)) \mid x_i \in O, i = 1, \ldots, n \,\} \\
& = & \sum\{\, \mu'(B_\epsilon(x_i)) \mid x_i \in O, i = 1, \ldots, n \,\} \\
& = & \mu'(\cup\{\, B_\epsilon(x_i) \mid x_i \in O, i = 1, \ldots, n \,\}) \\
& = & \mu'(\cup_{i=1}^n B_\epsilon(x_i) \cap O) \\
& = & \mu'(O)
\end{array}$$

The two sums are equal because the two measures coincide on balls.

We have now shown that the measures coincide on the ring $\mathcal{O}_* = \bigcup_{\epsilon>0} \mathcal{O}_\epsilon$. The $\sigma$-ring generated by $\mathcal{O}_*$ is the collection of all Borel sets. A basic result from measure theory states that there exists a unique extension of a measure on a ring to the $\sigma$-ring generated by the ring. (See e.g. [102].) Therefore, two measures which coincide on $\mathcal{O}_*$ must also coincide on all Borel sets. $\square$

The lemma is used to show that $\mathit{Meas}(M)$ with distance $d_{\mathit{Meas}}$ is a metric space. It also allows defining a measure by giving the probabilities of open balls only, because the probabilities of the other events are then fixed. This fact is used e.g. in section 4.3.

**Lemma 3.3.16** *For any ultrametric space $M$, the space $\mathit{Meas}(M)$ is an ultrametric space. If $M$ is complete, then so is $\mathit{Meas}(M)$.*

**Proof** That $d_{\mathit{Meas}}$ is an ultrametric is straightforward using lemma 3.3.15 above. For completeness of $\mathit{Meas}(M)$ for a complete space $M$, we follow the reasoning of [189].

Assume that $M$ is a complete ultrametric space. The notation $\mathcal{O}_\epsilon = \{ O \subseteq M \mid x \in O \Rightarrow B_\epsilon(x) \subseteq O \}$ (for $\epsilon > 0$) and $\mathcal{O}_* = \bigcup_{\epsilon>0} \mathcal{O}_\epsilon$ from lemma 3.3.15 is again used. Let $(\mu_i)_{i\in\mathbb{N}}$ be a Cauchy sequence of compact support measures on $\mathcal{O}_*$. We show that the point wise limit $\mu_\infty$ of this Cauchy sequence is again a compact support measure on $\mathcal{O}_*$. To prove this we need to show that $\mu_\infty(M) = 1$, that $\mu_\infty$ is $\sigma$-additive and that $\mu_\infty$ has compact support. That $\mu_\infty(M) = 1$ is directly clear.

First some auxiliary properties are shown followed by the compactness of the support of $\mu_\infty$ and finally the $\sigma$-additivity of $\mu_\infty$.

(a) For all closed sets sets $E, F \subseteq M$ we have that $d(E, F) \leq \epsilon$ exactly when $\forall O \in \mathcal{O}_\epsilon : E \cap O = \emptyset \iff F \cap O = \emptyset$.

(b) For all measures $\nu_1, \nu_2$ in $\mathit{Meas}(M)$ we have $d(\mathrm{spt}(\nu_1), \mathrm{spt}(\nu_2)) \leq d(\nu_1, \nu_2)$.

(c) A measure $\nu$ has compact support $\mathrm{spt}(\nu) = K$ exactly when for every open set $O$ in $\mathcal{O}_*$, $\nu(O) = 0 \iff O \cap K = \emptyset$.

(d) The function $\mu$ is finitely additive.

(e) The set $\lim_{i\to\infty} \mathrm{spt}(\mu_i)$ exists and is compact. The support of $\mu_\infty$ is equal to this set, i.e. $\mathrm{spt}(\mu) = \lim_{i\to\infty} \mathrm{spt}(\mu_i)$.

(f) The function $\mu_\infty$ is $\sigma$-additive.

Finally using property (b) the sequence $\mathrm{spt}(\mu_i)$ is a Cauchy sequence of compact sets. By completeness of the space of compact sets (cf. [38]) its limit exists and is compact.

(a) This property follows from the definition of the Hausdorff distance on closed sets.

(b) This property is straightforward from property (a).

(c) If $\nu$ is a compact support measure then clearly $\nu(O) = 0 \iff O \cap \mathrm{spt}(\nu) = \emptyset$ for every open set $O$. If $\nu(O) = 0 \iff O \cap K = \emptyset$ for every open set in $\mathcal{O}_*$ then $\nu$ vanishes outside of $K$ so $\nu$ has compact support. It is straightforward to check that $\mathrm{spt}(\nu) = K$.

(d) The finite additivity of $\mu_\infty$ is immediate from the finite additivity of each of the measures $\mu_i$.

(e) Property (b) shows that the sequence $\mathrm{spt}(\mu_i)$ is a Cauchy sequence of compact sets. By completeness of the space of compact sets (cf. [38]) its limit exists and is compact. Property (c) is used to show that $\mu_\infty$ has compact support and that the support of $\mu_\infty$ is given by $\mathrm{spt}(\mu_\infty) = \lim_{i\to\infty} \mathrm{spt}(\mu_i)$: Let $O \in \mathcal{O}_\epsilon$ and let $i > i_\epsilon$. Then

$$\begin{array}{l} O \cap (\lim_{i\to\infty} \mathrm{spt}(\mu_i)) = \emptyset \\ \iff [\text{property (a)}, i > i_\epsilon] \quad O \cap \mathrm{spt}(\mu_i) = \emptyset \\ \iff [\text{property (c)}] \quad \mu_i(O) = 0 \\ \iff [i > i_\epsilon] \quad \mu_\infty(O) = 0 \end{array}$$

(f) Let $(O_i)_{i\in\mathbb{N}}$ be a sequence of disjoint sets in $\mathcal{O}_*$ such that $\cup_{i\in\mathbb{N}} O_i$ is also a set in $\mathcal{O}_*$. Then $\cup_{i\in\mathbb{N}} O_i \in \mathcal{O}_\epsilon$ for some $\epsilon > 0$. But then $O' = M \setminus (\cup_{i\in\mathbb{N}} O_i)$ is also a set in $\mathcal{O}_\epsilon$ so in particular, this is an open set. The sets $O'$ and $\mathcal{O}_i$ for $i \in \mathbb{N}$ cover the compact support set $\mathrm{spt}(\mu_\infty)$. As the sets are disjoint, only finitely many of these sets can intersect $\mathrm{spt}(\mu_\infty)$. Let $I$ be the finite set of indices of sets $O_i$ that intersect $\mathrm{spt}(\mu_\infty)$.

$$\begin{array}{rcl} \mu_\infty(\cup_{i=1}^\infty O_i) & = & \mu_\infty(\cup_{i\notin I} O_i \cup \cup_{i\in I} O_i) \\ [\text{finite additivity}] & = & \mu_\infty(\cup_{i\notin I} O_i) + \sum_{i\in I} \mu(O_i) \\ & = & 0 + \sum_{i\in I} \mu_\infty(O_i) \\ & = & \sum_{i\notin I} \mu_\infty(O_i) + \sum_{i\in I} \mu_\infty(O_i) \\ & = & \sum_{i=1}^\infty \mu_\infty(O_i) \end{array}$$ □

As *Meas* yields a complete ultrametric space for any given complete ultrametric space, the domain *Meas*($Act^\infty$), which consists of all measures over sequences of actions, is also a complete ultrametric space. More complex domains can also be defined by using *Meas* in domain equations. To allow the use of *Meas* in domain equations, it is defined as functor on *CUMS* by giving *Meas*($f$) for each nonexpansive function $f$.

**Lemma 3.3.17** *Let $M, N$ be complete ultrametric spaces and $f\colon M \to N$ a nonexpansive function. Define Meas($f$) : Meas($M$) $\to$ Meas($N$) by*

$$\mathit{Meas}(f)(\mu)(B) = \mu(f^{-1}(B))$$

*Then Meas is a locally nonexpansive functor on CUMS.*

**Proof** Let $f, g : M \to N$ be two nonexpansive functions. If $d(f,g) \le \varepsilon$ then for all $x \in M$ we have that $f^{-1}(B_\epsilon(x)) = g^{-1}(B_\epsilon(x))$ holds. But then $\mathit{Meas}(f)(\mu)(B_\epsilon(x)) = \mu(f^{-1}(B_\epsilon(x))) = \mu(g^{-1}(B_\epsilon(x))) = \mathit{Meas}(g)(\mu)(B_\epsilon(x))$ for all $\mu \in \mathit{Meas}(M)$. This means that if $d(f,g) \le \varepsilon$ then $d(\mathit{Meas}(g), \mathit{Meas}(f) \le \varepsilon$. As this holds for all $\varepsilon > 0$ we have $d(\mathit{Meas}(g), \mathit{Meas}(f) \le d(f,g)$. □

The domain $\mathbb{P}_o$ used for the operational semantics consists of the space of measures over sequences of actions. When working with measures over the space of sequences of actions, the function $\bullet/a$ plays an important role.

**Definition 3.3.18**

*(a) The operational domain $\mathbb{P}_o$ is given by*

$$\mathbb{P}_o \quad = \quad \mathit{Meas}(\mathit{Act}^\infty)$$

*(b) The function $\bullet/a : \mathcal{B}(\mathit{Act}^\infty) \to \mathcal{B}(\mathit{Act}^\infty)$ is given by $B/a = \{\, w \mid aw \in B \,\}$.*

*(c) The function $\bullet/a : \mathit{Meas}(\mathit{Act}^\infty) \to \mathit{Meas}(\mathit{Act}^\infty)$ is given by $\mu/a(B) = \mu(B/a)$.*

If $B$ is a Borel set then $B/a$ is also a Borel set. The operation $\bullet/a$ on measures is the lifting to measures of prefixing with action $a$ as is done by *Meas*: $\mu/a = \lambda B.\mu(\{\, w \mid aw \in B \,\}) = \mathit{Meas}(\lambda w.aw)(\mu)$. The measure $\mu/a$ is called the measure along prefix $a$.

**Example 3.3.19** *Let $a, b$ be actions in Act.*

*(a) The sets $\{\, \epsilon \,\}$, $\{\, ababab\cdots \,\}$, $\{\, a, aa, aaa, \dots \,\}$ and $\{\, aw \mid w \in \mathit{Act}^\infty \,\}$ are in $\mathcal{B}(\mathit{Act}^\infty)$. Each of these sets is either open or closed. The set $\{\, ababab\cdots \,\} \cup \{\, a, aa, aaa, \dots \,\}$ is neither open nor closed but is still a Borel set because it is the union of two Borel sets (a closed and an open set).*

*(b) The sets $\{\, \epsilon \,\}/a = \emptyset$, $\{\, ababab\cdots \,\}/a = \{\, babab\cdots \,\}$, $\{\, a, aa, aaa, \dots \,\}/a = \{\, \epsilon, a, aa, \dots \,\}$ and $\{\, aw \mid w \in \mathit{Act}^\infty \,\}/a = \mathit{Act}^\infty$ are also in $\mathcal{B}(\mathit{Act}^\infty)$.*

*(c) There exists a measure $\mu$ such that for any word $w$ in $\{\, a, b \,\}^*$: $\mu(w\mathit{Act}^\infty) = \frac{1}{2}^{|w|}$ i.e. $\mu(a\mathit{Act}^\infty) = \frac{1}{2}$, $\mu(b\mathit{Act}^\infty) = \frac{1}{2}$, $\mu(aa\mathit{Act}^\infty) = \frac{1}{4}$, $\mu(ab\mathit{Act}^\infty) = \frac{1}{4}$, etc. This measure describes the meaning of the statement $x$ with $D(x) = (a \oplus_{\frac{1}{2}} b); x$.*

*(d) If a measure $\mu$ satisfies $\mu(\{\, \epsilon \,\}) = \frac{1}{2}$ and $\mu(\{\, baba\cdots \,\}) = \frac{1}{2}$, then $\mu/a$ satisfies $\mu/a(\{\, a \,\}) = \frac{1}{2}$ and $\mu/a(\{\, ababa\cdots \,\}) = \frac{1}{2}$.*

The last example above shows why the operation $\bullet/a$ can be seen as action prefixing on measures.

### 3.3.5 Operational semantics: The function $\mathcal{O}$

With the operational domain in place, the operational meaning of resumptions can be given. For an element $x \in M$ the measure $\Delta_x : \mathcal{B}(M) \to \{\, 0, 1 \,\}$ is given by

$$\Delta_x(B) \quad = \quad \begin{cases} 1 & \text{if } x \in B \\ 0 & \text{otherwise} \end{cases}$$

The measure $\Delta_x$ is called the *Dirac measure* at $x$. The Dirac measure at $x$ assigns all probability to $x$.

**Definition 3.3.20** *The operational model $\mathcal{O} : \mathit{Res} \to \mathbb{P}_o$ is given by*

$$\begin{aligned} \mathcal{O}(\mathrm{E}) \quad &= \quad \Delta_\epsilon \\ \mathcal{O}(s) \quad &= \quad \sum_{s \xrightarrow{\rho\cdot a}_n r} n \cdot \rho \cdot \mathcal{O}(r)/a \end{aligned}$$

The empty resumption produces no actions, so its observable behavior is $\epsilon$ (the empty sequence) with probability 1. The measure $\mathcal{O}(r)/a$ is the measure $\mathcal{O}(r)$ along prefix $a$. Recall that taking the measure along prefix $a$ can be seen as prefixing with action $a$, i.e. $\mathcal{O}(r)/a$ can be read as $a$ followed by the meaning of $r$. The probability that a given event $B$ occurs when executing the statement $s$, i.e. $\mathcal{O}(s)(B)$, is obtained by adding the probabilities for event $B$ for each step that $s$ can take. To obtain a sequence in $B$ when $s$ executes an action $a$ and end up with resumption $r$, the resumption $r$ must produce a sequence $w$ such that $aw$ is in $B$, i.e. $r$ must produce a sequence in $B/a$. The probability that $r$ produces such a sequence is $\mathcal{O}(r)(B/a) = (\mathcal{O}(r)/a)(B)$. If the transition $s \xrightarrow{\rho\cdot a} r$ occurs $n$ times ($s \xrightarrow{\rho\cdot a}_n r$) then this transition contributes $n{\cdot}\rho{\cdot}\mathcal{O}(r)(B/a)$ to the probability of the event $B$.

The definition can be justified by showing that it is the fixed point of a higher-order transformation:

**Lemma 3.3.21** *Let $Sem = Res \to \mathbb{P}_o$, let $S$ range over $Sem$ and let $\Phi : Sem \to Sem$ be given by*

$$\begin{array}{rcl} \Phi(S)(\mathrm{E}) & = & \Delta_\epsilon \\ \Phi(S)(s) & = & \displaystyle\sum_{s \xrightarrow{\rho\cdot a}_n r} n\cdot\rho\cdot S(r)/a \end{array}$$

*Then $\Phi$ has a unique fixed point, and therefore there is exactly one function $\mathcal{O}$ satisfying the equations in definition 3.3.20.*

**Proof** It is sufficient to show that $\Phi$ is a contractive function; using Banach's theorem this gives that $\Phi$ has a unique fixed point.

Let $\varepsilon{>}0$ and $S_1, S_2 \in Sem$ with $d(S_1,S_2){<}\varepsilon$ be given. To show that $d(\Phi(S_1),\Phi(S_2)) \leq \frac{1}{2}\varepsilon$ holds it is sufficient to show that $\Phi(S_1)(r)(B_{\frac{1}{2}\varepsilon}(w)) = \Phi(S_2)(r)(B_{\frac{1}{2}\varepsilon}(w))$ holds for all $r \in Res$ and $w \in Act^\infty$. As $\Phi(S_1)(\mathrm{E}) = \Delta_\epsilon = \Phi(S_2)(\mathrm{E})$ the case $r = \mathrm{E}$ is clear.

For the case $r = s$ with $s \in Stat$ note that

$$B_{\frac{1}{2}\varepsilon}(aw)/b = \begin{cases} \emptyset & \text{if } a \neq b \\ B_\varepsilon(w) & \text{otherwise} \end{cases}$$

For each resumption $r \in Res$ and sequence $w \in Act^\infty$, $S_1(r)(B_\varepsilon(w))$ equals $S_2(r)(B_\varepsilon(w))$ as $d(S_1,S_2) < \varepsilon$. Since also $S_1(r)(\emptyset) = 0 = S_2(r)(\emptyset)$, we have that $S_1(r)(B_{\frac{1}{2}\varepsilon}(w)/a) = S_2(r)(B_{\frac{1}{2}\varepsilon}(w)/a)$, for all $r \in Res$, $w \in Act^\infty$.

$$\begin{array}{rcl} \Phi(S_1)(s)(B_{\frac{1}{2}\varepsilon}(w)) & = & \displaystyle\sum_{s \xrightarrow{\rho\cdot a}_n r} n\cdot\rho\cdot S_1(r)(B_{\frac{1}{2}\varepsilon}(w)/a) \\ & = & \displaystyle\sum_{s \xrightarrow{\rho\cdot a}_n r} n\cdot\rho\cdot S_2(r)(B_{\frac{1}{2}\varepsilon}(w)/a) \\ & = & \Phi(S_2)(s)(B_{\frac{1}{2}\varepsilon}(w)) \end{array} \qquad \square$$

Lemma 3.3.21 illustrates the use of a higher-order function to justify the reflexive definition of the operational semantics. The higher-order characterization of the operational semantics is also used in section 3.5 to show correctness of the denotational model with respect to the operational model.

The operational semantics for $\mathcal{L}_p$ should be given for programs. The function $\mathcal{O}$, however, gives the meaning of resumptions. To remove this small discrepancy we define:

**Definition 3.3.22** *The operational semantics $\mathcal{O}[\![\bullet]\!] : \mathcal{L}_p \to \mathbb{P}_o$ for $\mathcal{L}_p$ is given by $\mathcal{O}[\![s]\!] := \mathcal{O}(s)$.*

The operational semantics $\mathcal{O}[\![\bullet]\!]$ is the same as the operational model $\mathcal{O}$ but restricted to programs in $\mathcal{L}_p$.

## 3.4 The denotational semantics of $\mathcal{L}_p$

The operational semantics gives the observable behavior of a program through use of a transition system. The transition system describes an abstract machine which runs the program. There is also a more abstract way of thinking about a program. For example the intuition behind the program $s_1; s_2$ is, first behave like $s_1$, then like $s_2$. This way of describing the meaning of statement $s_1; s_2$ works for all statements $s_1$ and $s_2$ independent of the actual transitions that these statements produce. Of course the description "behave like $s_1$ and then like $s_2$" still has to be made precise by giving the meaning of $s_1$ and $s_2$ and a way to compose these meanings.

In general one can think of the meaning of a program in terms of the meanings of the parts of the programs and a way to compose these meanings: The meaning of a program is built using the compositionality principle. In this section the denotational semantics for $\mathcal{L}_p$ is given. The denotational semantics of a program is obtained by using the compositionality principle. The meaning of a basic program is given as an element of the denotational domain $\mathbb{P}_d$ and semantical operations $;$ and $\oplus_\rho$ on $\mathbb{P}_d$ are introduced to compose these meanings. In this way the meaning can be given for programs built with the syntactical operators $;$ and $\oplus_\rho$.

The denotational domain $\mathbb{P}_d$ is a *branching domain*. The elements of $\mathbb{P}_d$ are called processes. At each point a process can probabilistically select the next action. This means that a process which makes a choice before executing the first action can be distinguished from a process which makes a choice after executing the first action even if the processes produce the same sequences with the same probabilities. On the processes in $\mathbb{P}_d$, the operations $\oplus_\rho$ and $;$ can easily be defined. The branching information is not strictly necessary for the language $\mathcal{L}_p$, a denotational semantics can also be given on a linear domain as is done in chapter 5. For extensions of the language (e.g. as in chapter 4) the branching information is really needed.

In a branching domain, the number of probabilistic options is finite at each stage. A probabilistic choice can be described by simply listing the probabilistic options together with their probability. As in the transition system, the same options may appear more than once. The statement $a \oplus_{\frac{1}{2}} a$ has two $a$ transitions to E. Similarly, in the meaning of $a \oplus_{\frac{1}{2}} a$, the option "$a$ with probability $\frac{1}{2}$" should appear twice. A probabilistic choice should therefore be modeled with multisets rather than with sets.

The set of all multisets over a given set is defined in section 3.2. When used in the denotational domain, a metric structure has to be defined on multisets. The functor $\mathcal{MP}_f$

constructs the metric space of all finite multisets over a given metric space. An important property of the functor $\mathcal{MP}_f$ is that the functor is locally nonexpansive. That $\mathcal{MP}_f$ is locally nonexpansive is important for the existence of a unique solution of the domain equations which define the denotational domain $\mathbb{P}_d$.

The next subsection introduces the functor $\mathcal{MP}_f$. The denotational domain and the semantical operations are defined in subsection 3.4.2. The denotational semantics is given in subsection 3.4.3.

### 3.4.1 Multisets over metric spaces: The functor $\mathcal{MP}_f$

In this subsection the functor $\mathcal{MP}_f$ that yield an ultrametric space $\mathcal{MP}_f(S)$ of multisets over a given ultrametric space $S$ is defined. Next it is shown that for a complete ultrametric space $S$ the space $\mathcal{MP}_f(S)$ is also complete, making $\mathcal{MP}_f$ an endo-functor on *CUMS*. Finally $\mathcal{MP}_f$ is defined on nonexpansive mappings (the arrows of *CUMS*) and shown to be locally nonexpansive.

Recall from definition 3.2.1 that a labeling with elements of a space $S$, $L \in \mathbb{L}(S)$, is a function from $\mathbb{N}$ to $S + \{ * \}$. If $S$ is a metric space equipped with metric $d$ then the metric $d_{Lab}$ on $\mathbb{L}(S)$ is given by

$$d_{Lab}(L_1, L_2) \quad = \quad \sup\{ d_*(L_1(n), L_2(n)) \mid n \in \mathbb{N} \}$$

where the metric $d_*$ on $S + \{ * \}$ is given by $d_*(x, x') = d(x, x')$, $d_*(*, *) = 0$ and $d_*(*, x) = d_*(x, *) = 1$ for $x, x' \in S$. The metric $d_*$ is the metric on $S + \{ * \}$ generated by $d$ and $d_{Lab}$ is the metric on $\mathbb{N} \to (S + \{ * \})$ generated by $d_*$. The distance of two labelings is the largest distance between the labels assigned to any $n \in \mathbb{N}$. The distance between a label in $s$ and an undefined label $*$ is one. As there are only finitely many labels not equal to $*$, the supremum used in the definition is actually a maximum.

The metric $d_{M\mathcal{P}_f}$ on $\mathcal{MP}_f(S)$ is defined as follows:

**Definition 3.4.1** *Let $L_1, L_2$ be two labelings. The distance between the multisets $\overline{L_1}$ and $\overline{L_2}$ is given by*

$$d_{M\mathcal{P}_f}(\overline{L_1}, \overline{L_2}) \quad = \quad \min\{d_{Lab}(L, L') \mid L \in \overline{L_1}, L' \in \overline{L_2}\}$$

The distance between multisets is the minimum distance between labelings representing the multisets. Equivalently we can put

$$d_{M\mathcal{P}_f}(\overline{L_1}, \overline{L_2}) \quad = \quad \min\{d_{Lab}(L_1 \circ \Phi, L_2) \mid \Phi : \mathbb{N} \to \mathbb{N} \text{ is a bijection}\}$$

where the representative for one of the multisets is fixed and a closest labeling for the other multiset is taken. Calculating the distance can be seen as pairing up the elements of both multisets (choosing $\Phi$) and taking the largest distance within a pair ($d_{Lab}$). Note that the distance between two multisets with a different number of elements is always 1.

Clearly $d_{M\mathcal{P}_f}(\overline{L_1}, \overline{L_2}) \leq 1$. Any bijection $\Phi$ which gives $d_{Lab}(L_1 \circ \Phi, L_2) < 1$ must map the support of $L_2$ to the support of $L_1$. The behavior of $\Phi$ on the rest of $\mathbb{N}$ is irrelevant for the distance $d_{Lab}(L_1 \circ \Phi, L_2)$. As there are only finitely many ways to map the finite support of $L_2$ to the finite support of $L_1$, there are only finitely many possible distances less than one and therefore the minimum used in the definition exists. Having shown that $d_{M\mathcal{P}_f}$ is a well-defined function, showing that $d_{M\mathcal{P}_f}$ is a metric is straightforward.

**Example 3.4.2** *Consider $S = \{ a, b \}^\infty$, with the Baire metric. Define the multisets $M_1$ and $M_2$ over $S$ by:*

$$\begin{aligned} M_1 &= \{\!|\, ab, ab, b \,|\!\} \\ M_2 &= \{\!|\, ab, abb, b \,|\!\} \end{aligned}$$

*then $d_{M\mathcal{P}_f}(M_1, M_2) = \frac{1}{4}$. The elements can be paired as follows: The first ab from the multiset $M_1$ is compared with ab from $M_2$, the second ab from $M_1$ is compared with abb from $M_2$ and b from $M_1$ is compared with b from $M_2$. The maximum distance that is obtained in this way is $\frac{1}{4}$.*

*Using the formal definition a representative $L_1$ is chosen for $M_1$. A labeling $L_2$ in the class $M_2$ with minimum distance to $L_1$ is then sought. The two possible choices for $L_2$ are the one below and $L_2$ with $L_2(1)$ and $L_2(2)$ reversed.*

$$\begin{array}{rcl} L_1(1) &=& ab \\ L_1(2) &=& ab \\ L_1(3) &=& b \end{array} \qquad \begin{array}{rcl} L_2(1) &=& ab \\ L_2(2) &=& abb \\ L_2(3) &=& b \end{array} \qquad \begin{array}{rccccc} L_1(x): & ab & ab & b & * \\ x: & 1 & 2 & 3 & \dots \\ L_2(x): & ab & abb & b & * \end{array}$$

*The distance between the labelings $L_1$ and $L_2$ is $\frac{1}{4}$.*

The next step is showing that $(\mathcal{MP}_f(S), d_{M\mathcal{P}_f})$ is complete and ultrametric for any complete ultrametric space $(S, d)$ in *CUMS*.

**Lemma 3.4.3** *If $(S, d)$ is a complete ultrametric space, then $(\mathcal{MP}_f(S), d_{M\mathcal{P}_f})$ is also a complete ultrametric space.*

**Proof** Showing that $d_{M\mathcal{P}_f}$ is an ultrametric is straightforward. For completeness of $(\mathcal{MP}_f(S), d_{M\mathcal{P}_f})$ consider a given Cauchy sequence of multisets $(M_i)_{i\in\mathbb{N}}$. Fix a representation $L_1$ for the first element $M_1$. For $M_2$ choose a representation $L_2$ with minimum distance to $L_1$. For $M_3$ choose a representation $L_3$ with minimum distance to $L_2$, etc. The sequence of labelings obtained by doing this is also a Cauchy sequence as $d(L_i, L_{i+1}) = d(M_i, M_{i+1})$. The Cauchy sequence of labelings must have a limit $L_\infty$ in $\mathbb{N} \to (S + \{ * \})$ as this space of labelings is complete. That $L_\infty$ again has finite support is clear from the fact that two labelings which are close must have the same support. It is easy to see that $\overline{L_\infty}$ is the limit of the sequence of multisets. □

The space of all finite sets over a space $S \in$ *CUMS* is not complete: For example the Cauchy sequence $(X_i)_{i\in\mathbb{N}}$ given in example 3.4.4 below does not have a finite set as its limit. The metric on multisets, however, is significantly different from the Hausdorff distance on sets. Two sets which are close in the Hausdorff sense can still have distance 1 if interpreted as multisets (with multiplicity 1 for each element).

**Example 3.4.4** *Take for $S$ the space $\{a\}^\infty$ consisting of all (possibly infinite) sequences of a's, then the following holds:*

$$\begin{array}{rcl} d_H(\{\, a, aa, aaa \,\}, \{\, a, aa \,\}) & = & \frac{1}{4} \\ d_{\mathcal{MP}_f}(\{\!\mid a, aa, aaa \mid\!\}, \{\!\mid a, aa \mid\!\}) & = & 1 \end{array}$$

*where $d_H$ is the Hausdorff distance on sets. The second distance is $1$ since the first multiset contains more elements than the second. Irrespective of the representation, one element of the first multiset will always be compared with $*$.*

*For $i \in \mathbb{N}$ let $X_i = \{\, a, aa, \ldots, a^i \,\}$ and $M_i = \{\!\mid a, aa, \ldots, a^i \mid\!\}$ then $(X_i)_{i \in \mathbb{N}}$ is a Cauchy sequence in $\mathcal{P}_f(S)$ but $(M_i)_{i \in \mathbb{N}}$ is not a Cauchy sequence in $\mathcal{MP}_f(S)$.*

As an aside, the following relation exists: If two multisets are close then so are the sets obtained by forgetting multiplicity. Also if two finite sets are close then there are two multisets which contain the same elements as the sets (but possibly more than once) that are also close. In example 3.4.4 above, one could take $\{\!\mid a, aa, aa \mid\!\}$ for the second multiset.

By defining $\mathcal{MP}_f$ on nonexpansive functions, $\mathcal{MP}_f$ becomes an endo-functor on *CUMS*, the category of complete ultrametric spaces.

**Definition 3.4.5** *Let $S_1, S_2$ be complete ultrametric spaces in CUMS and let $g$ be a nonexpansive function from $S_1$ to $S_2$ then $\mathcal{MP}_f(g) : \mathcal{MP}_f(S_1) \xrightarrow{1} \mathcal{MP}_f(S_2)$ is defined by*

$$\mathcal{MP}_f(g)(\overline{L}) = \overline{g^* \circ L}$$

*where $\overline{L}$ is a multiset in $\mathcal{MP}_f(S_1)$ and* $g^*(x) = \begin{cases} g(x) & \text{if } x \in S_1 \\ * & \text{if } x = * \end{cases}$

Using the notation introduced in definition 3.2.5 the definition can also be written as $\mathcal{MP}_f(g)(M) = \{\!\mid g(x) \mid x \in M \mid\!\}$ for any multiset $M \in \mathcal{MP}_f(S_1)$. This means that a function is lifted to multisets by applying the function elementwise, so for example $\mathcal{MP}_f(g)(\{\!\mid x_1, x_2, \ldots, x_n \mid\!\}) = \{\!\mid g(x_1), g(x_2), \ldots, g(x_n) \mid\!\}$. Checking that the definition does not depend on the choice of the representative $L$ is again straightforward. It is also easy to see that $\mathcal{MP}_f(g)$ is a nonexpansive function.

**Lemma 3.4.6** *The functor $\mathcal{MP}_f$ is locally nonexpansive.*

**Proof** Let $S_1, S_2 \in CUMS$, $g, h : S_1 \xrightarrow{1} S_2$ and $M \in \mathcal{MP}_f(S_1)$. Choose a representative $L$ for the multiset $M$ then:

$$\begin{array}{rcl} d(\mathcal{MP}_f(g)(M), \mathcal{MP}_f(h)(M)) & = & d(\overline{g^* \circ L}, \overline{h^* \circ L}) \;\le\; d(g^* \circ L, h^* \circ L) \\ & \le & d(g^*, h^*) \;=\; d(g, h) \end{array}$$

That $d(\mathcal{MP}_f(g), \mathcal{MP}_f(h)) \le d(g, h)$ follows since this inequality holds for every multiset $M$ in $\mathcal{MP}_f(S_1)$. □

Having this property available, the functor $\mathcal{MP}_f$ can be used in domain equations. The first place it is used is in the definition of the denotational domain $\mathbb{P}_d$ below.

Caution has to be observed when defining functions that return multisets. If such a function $f$ has to be contractive in an argument $x$, one has to see to it that the number of elements in $f(x)$ is independent of $x$. The following form of definition, not unusual when working with sets, does not result in a contractive function $f$ when using multisets.

**Example 3.4.7** *The function $g\colon \mathcal{MP}_f(\{\, a, b \,\}^\infty) \to \mathcal{MP}_f(\{\, a, b \,\}^\infty)$ given by*

$$g(M) = \{\!|\, aw \mid w \in M \,|\!\}$$

*is not contractive, since*

$$d(g(\{\!|\, a, b \,|\!\}), g(\{\!|\, a \,|\!\})) \;=\; d(\{\!|\, aa, ab \,|\!\}, \{\!|\, aa \,|\!\}) \;=\; 1 \;=\; d(\{\!|\, a, b \,|\!\}, \{\!|\, a \,|\!\})$$

Definitions of the form given in example 3.4.7 are typically used in linear domains. To describe probability in a linear domain, measures are employed, as done in section 3.3. Multisets are only used to describe probabilistic choices in a branching fashion.

### 3.4.2 A branching probabilistic domain: $\mathbb{P}_d$

In this subsection, a branching domain of probabilistic processes, $\mathbb{P}_d$, is defined. On $\mathbb{P}_d$ semantical versions of the syntactical operators ; and $\oplus_\rho$ are given. The domain $\mathbb{P}_d$ is defined by domain equations using the functor $\mathcal{MP}_f$. A probabilistic process has a choice between several alternatives. Each alternative is an action together with a probability that this action is chosen. An action may be followed by another process.

**Definition 3.4.8** *The denotational domain $\mathbb{P}_d$ is given by the following domain equations*

$$\begin{array}{rcl} \mathbb{P}_d & \simeq & \mathcal{MP}_f(\mathbb{Q}_d) \\ \mathbb{Q}_d & \simeq & PAct + PAct \times id_{\frac{1}{2}}(\mathbb{P}_d) \end{array}$$

Recall that $PAct = [0,1] \times Act$. As shown in lemma 2.2.10, a domain equation $\mathbb{P} = \mathcal{F}(\mathbb{P})$ has a unique solution up to isomorphism if the functor $\mathcal{F}$ is locally contractive. That the functor $\mathcal{MP}_f(PAct + PAct \times id_{\frac{1}{2}}(\bullet))$ is locally contractive follows directly from the fact that $\mathcal{MP}_f$ is locally nonexpansive. The domain equation for $\mathbb{P}_d$ has a unique solution up to isomorphism. On the domain $\mathbb{P}_d$ the operations $\oplus_\rho$ and ; are defined as follows.

**Definition 3.4.9** *Let $Op = \mathbb{P}_d \times \mathbb{P}_d \xrightarrow{1} \mathbb{P}_d$. The operations $\oplus_\rho$, ; $\in Op$ are given by*

$$\begin{array}{rcl} p \oplus_\rho p' & = & \rho\, p \sqcup (1-\rho)\, p' \\ p; p' & = & \{\!|\, \langle \alpha, p' \rangle \mid \alpha \in p \,|\!\} \sqcup \{\!|\, \langle \alpha, p''; p' \rangle \mid \langle \alpha, p'' \rangle \in p \,|\!\} \end{array}$$

Multiplication with a ratio $\rho$ is defined on $\mathbb{Q}_d$ by $\rho(\sigma \cdot a) = \rho\sigma \cdot a$ and $\rho\langle \sigma \cdot a, p \rangle = \langle \rho\sigma \cdot a, p \rangle$. The multiplication with $\rho$ is lifted to multisets as specified by $\mathcal{MP}_f$, i.e. by multiplying all elements of the multiset by $\rho$, thus

$$\rho\{\!|\, x_1, x_2, \ldots, x_n \,|\!\} \;=\; \{\!|\, \rho x_1, \rho x_2, \ldots, \rho x_n \,|\!\}$$

That the operation $\oplus_\rho$ is well-defined follows directly from the fact that the operations 'multiplying by $\rho$' and $\sqcup$ are nonexpansive. The definition of ; is recursive but can be shown to be correct using metric machinery, by showing that the operation is the unique fixed point of a contractive higher order function $\Omega_;\colon Op \to Op$.

**Lemma 3.4.10** *Define* $\Omega_; : Op \to Op$ *by*

$$\Omega_;(\phi)(p,p') \quad = \quad \{\!\!\{\, \langle \alpha, p' \rangle \mid \alpha \in p \,\}\!\!\} \sqcup \{\!\!\{\, \langle \alpha, \phi(p'',p') \rangle \mid \langle \alpha, p'' \rangle \in p \,\}\!\!\}$$

*then* $\Omega_;$ *has a unique fixed point and therefore there exists exactly one operation* ; *in Op satisfying the equation in definition 3.4.9.*

**Proof** Clearly any operation satisfying the equation for ; in definition 3.4.9 is a fixed point of $\Omega_;$. It is sufficient to show that $\Omega_;$ is contractive as then $\Omega_;$ has a unique fixed point by Banach's fixed point theorem.

Let two operations $\phi, \phi' \in Op$ and two processes $p, p' \in \mathbb{P}_d$ be given. As

$$\begin{aligned} \Omega_;(\phi)(p,p') \quad &= \quad \{\!\!\{\, \langle \alpha, p' \rangle \mid \alpha \in p \,\}\!\!\} \sqcup \{\!\!\{\, \langle \alpha, \phi(p'',p') \rangle \mid \langle \alpha, p'' \rangle \in p \,\}\!\!\} \text{ and} \\ \Omega_;(\phi')(p,p') \quad &= \quad \{\!\!\{\, \langle \alpha, p' \rangle \mid \alpha \in p \,\}\!\!\} \sqcup \{\!\!\{\, \langle \alpha, \phi'(p'',p') \rangle \mid \langle \alpha, p'' \rangle \in p \,\}\!\!\} \end{aligned}$$

the distance $d(\Omega_;(\phi)(p,p'), \Omega_;(\phi')(p,p'))$ is less than or equal to $d(\langle \alpha, p' \rangle, \langle \alpha, p' \rangle)$ for some $\alpha \in p$ or $d(\langle \alpha, \phi(p'',p') \rangle, \langle \alpha, \phi'(p'',p') \rangle)$ for some $\langle \alpha, p'' \rangle \in p$ and all these distances are less than or equal to $\frac{1}{2} d(\phi, \phi')$. □

The operations $\oplus_\rho$ and ; are nonexpansive by definition. Sequential composition is also contractive in its second argument.

**Lemma 3.4.11** *The operation* ; *is nonexpansive in its first argument and contractive in its second argument, i.e.*

$$d(p_1 \,;\, p_2, p_1' \,;\, p_2') \quad \leq \quad \max\{\, d(p_1, p_1'), \tfrac{1}{2} \cdot d(p_2, p_2') \,\}$$

**Proof** Take the subspace $Op'$ of $Op$ to be those operations which are nonexpansive in its first argument and contractive in its second argument, i.e. $\phi \in Op'$ if $d(\phi(p,p''), \phi(p',p''))$ $\leq d(p,p')$ and $d(\phi(p,p'), \phi(p,p'')) \leq \frac{1}{2} d(p',p'')$ for all processes $p, p', p''$.

We show that $Op'$ is a non-empty closed subset of $Op$ and that $\Omega_;$ restricts to a contractive function on $Op'$. This means that $\Omega_;$ has a fixed point within $Op'$. As ; is the only fixed point of $\Omega_;$, ; must be in $Op'$.

The operation $\phi_0$ with $\phi(p,p') = p$ is an element of $Op'$ so $Op'$ is non-empty. If $(\phi_i)_{i \in \mathbb{N}}$ is a Cauchy sequence in $Op'$ then the limit $\phi_\infty$, which exists in $Op$, is again in $Op'$ showing that $Op'$ is closed: As $d(\phi_i(p,p''), \phi_i(p',p'')) \leq d(p,p')$ for all $i \in \mathbb{N}$ and both $d(\phi_\infty(p,p''), \phi_i(p,p'') \leq d(p,p')$ and $d(\phi_\infty(p',p''), \phi_i(p',p'') \leq d(p,p')$ for sufficiently large indices $i$ we have that

$$\begin{aligned} & d(\phi_\infty(p,p''), \phi_\infty(p',p'')) \\ & \quad \leq \quad \max\{\, d(\phi_\infty(p,p''), \phi_i(p,p'')), d(\phi_i(p,p''), \phi_i(p',p'')), \\ & \qquad\qquad d(\phi_i(p',p''), \phi_\infty(p',p'')) \,\} \\ & \quad \leq \quad d(p,p') \end{aligned}$$

Similarly $d(\phi_\infty(p,p'), \phi_\infty(p,p'')) \leq \frac{1}{2} d(p',p'')$ as $d(\phi_\infty, \phi_i) \leq \frac{1}{2} d(p',p'')$ for sufficiently large indices $i$.

If $\phi \in Op'$ then $\Omega_;(\phi) \in Op'$ showing that $\Omega_;$ restricts to a function on $Op'$:

$$\begin{array}{ll} d(\Omega(\phi)(p,p'), \Omega(\phi)(p,p'')) & \\ \quad = & d(\{\!\!\{ \langle \alpha, p' \rangle \mid \alpha \in p \}\!\!\} \cup \{\!\!\{ \langle \alpha, \phi(\bar{p}, p') \rangle \mid \langle \alpha, \bar{p} \rangle \in p \}\!\!\}, \\ & \quad \{\!\!\{ \langle \alpha, p'' \rangle \mid \alpha \in p \}\!\!\} \cup \{\!\!\{ \langle \alpha, \phi(\bar{p}, p'') \rangle \mid \langle \alpha, \bar{p} \rangle \in p \}\!\!\}) \\ \quad \leq & \max(\{ d(\langle \alpha, p', \alpha, p'' \rangle) \mid \alpha \in p \} \cup \\ & \quad \{ d(\langle \alpha, \phi(\bar{p}, p') \rangle, \langle \alpha, \phi(\bar{p}, p'') \rangle) \mid \langle \alpha, \bar{p} \rangle \in p \}) \\ \quad \leq & \frac{1}{2} d(p', p'') \end{array}$$

To find a distance between two multisets, the elements of the multisets are linked. Without loss of generality both $p$ and $p'$ contain the same number of elements because otherwise directly $d(\Omega(\phi)(p,p''), \Omega(\phi)(p',p'')) \leq 1 = d(p,p')$. If, for example, an element $\langle \alpha, \bar{p} \rangle$ in $p$ is linked with $\langle \beta, \bar{p}', p'' \rangle$ in $p'$ then one can link $\langle \alpha, \phi(\bar{p}, p'') \rangle$ in $\Omega(\phi)(p,p'')$ with $\langle \beta, \phi(\bar{p}', p'') \rangle$ in $\Omega(\phi)(p',p'')$ and have that $d(\langle \alpha, \phi(\bar{p}, p'') \rangle, \langle \beta, \phi(\bar{p}', p'') \rangle) \leq d(\langle \alpha, \bar{p} \rangle, \langle \beta, \bar{p}' \rangle)$. One can do the same for the other elements of $p$ and $p'$ that are linked. In this way one obtains a linking of the elements of $\Omega(\phi)(p,p'')$ and $\Omega(\phi)(p',p'')$ with a distance between linked pairs smaller than or equal to the distance of some pair linked in $p$ and $p'$. This works for every linking of the multisets $p$ and $p'$ showing that $d(\Omega(\phi)(p,p''), \Omega(\phi)(p',p'')) \leq d(p,p')$. □

Contractiveness of ; in its second argument is required for the contractiveness of the higher-order specification of the denotational model given in the next subsection.

### 3.4.3 The denotational model $\mathcal{D}$

With the semantical operations ; and $\oplus_\rho$ in place, the definition of the denotational meaning of statements is as given by the compositionality principle. To define the meaning of a program, the semantical operations are used to combine the respective meanings of the part from which the program is constructed.

**Definition 3.4.12** *The denotational model* $\mathcal{D}\colon \mathcal{L}_p \to \mathbb{P}_d$

$$\begin{array}{rcl} \mathcal{D}(a) & = & \{\!\!\{ 1 \cdot a \}\!\!\} \\ \mathcal{D}(x) & = & \mathcal{D}(D(x)) \\ \mathcal{D}(s_1; s_2) & = & \mathcal{D}(s_1); \mathcal{D}(s_2) \\ \mathcal{D}(s_1 \oplus_\rho s_2) & = & \mathcal{D}(s_1) \oplus_\rho \mathcal{D}(s_2) \end{array}$$

A single action $a$ acts like $a$ with probability one. Recursion is handled by body replacement. As the procedure variable does not contain a substatement $D(x)$, the definition $\mathcal{D}(x) = \mathcal{D}(D(x))$ is not directly an application of the compositionality principle. However, as the body of a procedure is seen as less complex than the procedure itself (by using the weight of statements), the definition of the denotational semantics for procedures still uses the idea of decomposing a program into simpler parts. The semantical operation *op* is used to give the meaning of any statement built using the syntactic operator *op*, where *op* is either $\oplus_\rho$ or ; .

The recursive definition of the denotational semantics is justified by showing it is the fixed point of a contractive higher-order operation $\Psi$.

**Lemma 3.4.13** . *Let $Sem_D = \mathcal{L}_P \to \mathbb{P}_d$, let $S$ range over $Sem_D$ and let $\Psi : Sem_D \to Sem_D$ be given by*

$$\begin{array}{rcl} \Psi(S)(a) & = & \{\!|\, 1 \cdot a \,|\!\} \\ \Psi(S)(x) & = & \Psi(S)(D(x)) \\ \Psi(S)(s_1 ; s_2) & = & \Psi(S)(s_1) ; S(s_2) \\ \Psi(S)(s_1 \oplus_\rho s_2) & = & \Psi(S)(s_1) \oplus_\rho \Psi(S)(s_2) \end{array}$$

*Then $\Psi$ has a unique fixed point, and therefore there exists exactly one function $\mathcal{D}$ in $Sem_D$ satisfying the equations in definition 3.4.12.*

**Proof** Clearly a function $\mathcal{D}$ satisfies the equations in definition 3.4.12 exactly when it is a fixed point of $\Psi$. Banach's fixed point theorem gives that there exists a unique fixed point for $\Psi$ if $\Psi$ is contractive. That $\Psi$ is contractive, i.e. that $d(\Psi(S_1)(s), \Psi(S_2)(s)) \leq \frac{1}{2} d(S_1, S_2)$ for all $s \in \mathcal{L}_P$, is direct by induction on the weight of the statement $s$, using nonexpansiveness of $\oplus_\rho$ for the case $s = s_1 \oplus_\rho s_2$ and lemma 3.4.11 for the case $s = s_1 ; s_2$. $\square$

The following example shows how the meaning of a simple program consisting of a single procedure call is calculated and how a probabilistic process can be represented as a tree.

**Example 3.4.14** *Let $D(x) = (a \oplus_\rho b) ; c$ then*

$$\begin{array}{rcl} \mathcal{D}(x) & = & \mathcal{D}(D(x)) \;=\; \mathcal{D}((a \oplus_\rho b) ; c) \;=\; \mathcal{D}(a \oplus_\rho b) ; \mathcal{D}(c) \\ & = & (\mathcal{D}(a) \oplus_\rho \mathcal{D}(b)) ; \mathcal{D}(c) \;=\; (\{\!|\, 1 \cdot a \,|\!\} \oplus_\rho \{\!|\, 1 \cdot b \,|\!\}) ; \{\!|\, 1 \cdot c \,|\!\} \\ & = & \{\!|\, \rho \cdot a, (1-\rho) \cdot b \,|\!\} ; \{\!|\, 1 \cdot c \,|\!\} \\ & = & \{\!|\, \langle \rho \cdot a, \{\!|\, 1 \cdot c \,|\!\} \rangle, \langle (1-\rho) \cdot b, \{\!|\, 1 \cdot c \,|\!\} \rangle \,|\!\} \end{array}$$

$$\begin{tikzcd} & {} \arrow[dl, "\rho \cdot a"'] \arrow[dr, "(1-\rho) \cdot b"] & \\ {} \arrow[d, "1 \cdot c"'] & & {} \arrow[d, "1 \cdot c"] \\ {} & & {} \end{tikzcd}$$

*Note the similarity with the trees given in example 3.3.10. A branching probabilistic process in $\mathbb{P}_d$ is very similar to an abstract transition tree.*

With the denotational model in place the denotational semantics can be given. The denotational semantics $\mathcal{D}[\![\cdot]\!]$ and the model $\mathcal{D}$ coincide and are only given separately to maintain symmetry with the definition of the operational semantics.

**Definition 3.4.15** *The denotational semantics $\mathcal{D}[\![\cdot]\!] \colon \mathcal{L}_P \to \mathbb{P}_d$ is given by*

$$\mathcal{D}[\![s]\!] \;=\; \mathcal{D}(s)$$

## 3.5 Comparing the operational and denotational semantics

The denotational semantics of a statement is supposed to describe the meaning of a statement possibly with extra information required for the compositionality. The denotational semantics should be correct with respect to the operational semantics: It should be possible to recover the operational meaning of a statement by removing the extra information from the denotational meaning of the statement. If the extra information in the denotational meaning can be removed by an abstraction function *abs*, then the operational model is called an abstraction of the denotational model. In this section the operational model $\mathcal{O}[\![\cdot]\!]$ defined in section 3.3 is shown to be an abstraction of the denotational model $\mathcal{D}[\![\cdot]\!]$ defined in section 3.4.

The extra information added in the denotational semantics $\mathcal{D}$ for $\mathcal{L}_p$ consists of the moments of the probabilistic branching, present in the branching domain $\mathbb{P}_d$ but not in the linear domain $\mathbb{P}_o$. The operational meaning is obtained from the denotational meaning by abstracting away this branching information. This is done in several steps. First the denotational model $\mathcal{D}$ is extended to give a meaning to all resumptions instead of only to statements. To be able to assign a meaning to the empty resumption E the denotational domain is extended with the *empty process* $p_\epsilon$. Next an operational-like model $\mathcal{O}^*$ is defined on the denotational domain. The term operational-like model is used for a model based on the transition system, but defined on a domain other than the operational domain. The denotational model $\mathcal{D}$ is shown to coincide with the operational-like model $\mathcal{O}^*$ using the higher-order description of $\mathcal{O}^*$. Then an abstraction function *abs* which removes the branching information from meanings in $\mathbb{P}_d$ is given and the operational model $\mathcal{O}$ is shown to be an abstraction of $\mathcal{O}^*$. Finally these results are combined to show that the operational model is an abstraction of the denotational model.

$$\begin{tikzcd}
\mathbb{P}_d + \{\, p_\epsilon \,\} & \mathbb{P}_b \\
\mathcal{D} = \mathcal{O}^* \arrow[r, "abs"] & \mathcal{O}
\end{tikzcd}$$

The denotational model is extended by assigning the empty process $p_\epsilon$ as meaning to the empty resumption E. For statements $\mathcal{D}$ remains the same.

**Definition 3.5.1** *The model $\mathcal{D} : Res \to \mathbb{P}_d + \{\, p_\epsilon \,\}$ is given by*

$$\begin{aligned}
\mathcal{D}(\mathrm{E}) &= p_\epsilon \\
\mathcal{D}(a) &= \{\!|\, 1 \cdot a \,|\!\} \\
\mathcal{D}(x) &= \mathcal{D}(D(x)) \\
\mathcal{D}(s_1 \, op \, s_2) &= \mathcal{D}(s_1) \, op \, \mathcal{D}(s_2)
\end{aligned}$$

*for $op \in \{\ ;\, , \oplus_\rho \,\}$.*

The operational-like model $\mathcal{O}^*$ is defined as the fixed point of a higher-order operator $\Phi^*$.

**Definition 3.5.2** *Put Sem* $= Res \to \mathbb{P}_d$*, let* $S$ *range over Sem and define* $\mathcal{O}^*$ *as the fixed point of* $\Phi^*$*, where* $\Phi^* : Sem \to Sem$ *is given by*

$$\begin{array}{rcl} \Phi^*(S)(\mathrm{E}) & = & p_\epsilon \\ \Phi^*(S)(s) & = & \{\!|\, \langle \rho \cdot a, S(r) \rangle \mid \langle \rho \cdot a, r \rangle \in Suc(s) \,|\!\} \end{array}$$

Note that this definition uses the notation introduced in definition 3.2.5 part (c). This means that if $s \xrightarrow{\rho \cdot a}_n r$, i.e. $\langle \rho \cdot a, r \rangle \in_n Suc(s)$, then $\langle \rho \cdot a, S(r) \rangle$ is included $n$ times in $\Phi^*(S)(s)$. Note also that the only the resumption E is assigned the empty process as its meaning, all other resumptions are given a meaning in $\mathbb{P}_d$. Showing that $\Phi^*$ is a contraction on *Sem* is easy (cf. lemma 3.3.21). The following equations follow directly from this definition:

$$\begin{array}{rcl} \mathcal{O}^*(\mathrm{E}) & = & p_\epsilon \\ \mathcal{O}^*(s) & = & \{\!|\, \langle \rho \cdot a, \mathcal{O}^*(r) \rangle \mid \langle \rho \cdot a, r \rangle \in Suc(s) \,|\!\} \end{array}$$

Using the higher-order operator $\Phi^*$ we show that the operational-like model $\mathcal{O}^*$ coincides with the extended denotational model $\mathcal{D}$.

**Lemma 3.5.3** *The model* $\mathcal{D}$ *is a fixed point of* $\Phi^*$ *and thus, by uniqueness of the fixed point (Banach's theorem),* $\mathcal{D} = \mathcal{O}^*$*.*

**Proof** By induction on the weight of the resumption $\Phi^*(\mathcal{D})(r) = \mathcal{D}(r)$ is shown. The cases $r = \mathrm{E}$ and $r = a$ are immediate and the case $r = x$ is directly clear by induction. This leaves the cases $r = s_1 \oplus_\rho s_2$ and $r = s_1 ; s_2$.

$$\begin{array}{rcl} \Phi^*(\mathcal{D})(s_1 \oplus_\rho s_2) & = & \{\!|\, \langle \sigma \cdot a, \mathcal{D}(r') \rangle \mid \langle \sigma \cdot a, r' \rangle \in Suc(s_1 \oplus_\rho s_2) \,|\!\} \\ & = & \{\!|\, \langle \rho\sigma \cdot a, \mathcal{D}(r') \rangle \mid \langle \sigma \cdot a, r' \rangle \in Suc(s_1) \,|\!\} \\ & & \sqcup \{\!|\, \langle (1-\rho)\sigma \cdot a, \mathcal{D}(r') \rangle \mid \langle \sigma \cdot a, r' \rangle \in Suc(s_2) \,|\!\} \\ & = & \rho \Phi^*(\mathcal{D})(s_1) \sqcup (1-\rho)\Phi^*(\mathcal{D})(s_2) \\ \text{[ind. hyp.]} & = & \rho \mathcal{D}(s_1) \sqcup (1-\rho)\mathcal{D}(s_2) \\ & = & \mathcal{D}(s_1 \oplus_\rho s_2) \end{array}$$

as both $wgt(s_1)$ and $wgt(s_2)$ are less than $wgt(s_1 \oplus_\rho s_2)$.

$$\begin{array}{rcl} \Phi^*(\mathcal{D})(s_1 ; s_2) & = & \{\!|\, \langle \sigma \cdot a, \mathcal{D}(r') \rangle \mid \langle \sigma \cdot a, r' \rangle \in Suc(s_1 ; s_2) \,|\!\} \\ & = & \{\!|\, \langle \sigma \cdot a, \mathcal{D}(r' ; s_2) \rangle \mid \langle \sigma \cdot a, r' \rangle \in Suc(s_1) \,|\!\} \\ & = & \{\!|\, \langle \sigma \cdot a, \mathcal{D}(r') ; \mathcal{D}(s_2) \rangle \mid \langle \sigma \cdot a, r' \rangle \in Suc(s_1) \,|\!\} \\ & = & \{\!|\, \langle \sigma \cdot a, \mathcal{D}(r') \rangle \mid \langle \sigma \cdot a, r' \rangle \in Suc(s_1) \,|\!\} ; \mathcal{D}(s_2) \\ & = & \Phi^*(\mathcal{D})(s_1) ; \mathcal{D}(s_2) \\ \text{[ind. hyp.]} & = & \mathcal{D}(s_1) ; \mathcal{D}(s_2) \\ & = & \mathcal{D}(s_1 ; s_2) \end{array}$$

as $wgt(s_1) < wgt(s_1 ; s_2)$. □

To relate meanings in the branching denotational domain $\mathbb{P}_d$ with meanings in the linear operational domain $\mathbb{P}_o$, the branching information is abstracted away by applying an abstraction function *abs*.

**Definition 3.5.4** *The abstraction function* $abs : (\mathbb{P}_d + \{\, p_\epsilon \,\}) \to \mathbb{P}_o$ *is given by*

$$\begin{aligned} abs(p_\epsilon) &= \Delta_\epsilon \\ abs(\{\!\!\{\, \langle \rho_1 \cdot a_1, p_1 \rangle, \ldots, \langle \rho_n \cdot a_n, p_n \rangle \,\}\!\!\}) &= \sum_{i=1}^{n} \rho_i \,(abs(p_i)/a_i) \end{aligned}$$

The process $p_\epsilon$ will result in an empty sequence, $\epsilon$, with probability 1. Any other branching process is a multiset. Recalling that $\bullet/a$ can be seen as prefixing with action $a$ on measures the second equation in the definition becomes clear. Taking some observable event $B$ one can also reason directly: An element $\langle \rho \cdot a, p \rangle$ of the multiset adds $\rho$ times the probability that $p$ produces a sequence $w$ such that $aw \in B$ to the total probability of producing a sequence in $B$. The probability that $p$ produces a sequence $w$ such that $aw \in B$ is $abs(p)(B/a) = (abs(p)/a)(B)$.

The definition of the function *abs* is recursive but can be justified by showing that *abs* is the unique fixed point of a higher-order operator $\Omega_{abs}$. This justification is similar to that of ; (see lemma 3.4.10) and is therefore omitted.

By applying the abstraction function to a meaning in the branching denotational domain $\mathbb{P}_d$ the branching information is removed and a meaning in the linear operational domain $\mathbb{P}_b$ is obtained. As the addition of the branching information is supposed to be the only difference between the operational and the operational-like model, applying the abstraction to the operational-like model should yield the operational model. The following lemma shows that this is indeed the case.

**Lemma 3.5.5** *The operational model $\mathcal{O}$ is an abstraction of the operational-like model $\mathcal{O}^*$, i.e.*

$$\mathcal{O} = abs \circ \mathcal{O}^*$$

**Proof** This proof is very similar to the proof of lemma 3.5.3 above: It is sufficient to show that $abs \circ \mathcal{O}^*$ is a fixed point of $\Phi$ as $\mathcal{O}$ is the unique fixed point of $\Phi$. Further details are omitted. □

Combining the results in this section gives that the operational semantics $\mathcal{O}[\![\cdot]\!]$ is an abstraction of the denotational semantics $\mathcal{D}[\![\cdot]\!]$.

**Theorem 3.5.6** *For all $s \in \mathcal{L}_p$: $\mathcal{O}[\![s]\!] = abs \circ \mathcal{D}[\![s]\!]$.*

**Proof**

$$\begin{aligned} \mathcal{O}[\![s]\!] &= [\text{definition } \mathcal{O}[\![\cdot]\!]]\ \mathcal{O}(s) \\ &= [\text{lemma 3.5.5}]\ abs \circ \mathcal{O}^*(s) \\ &= [\text{lemma 3.5.3}]\ abs \circ \mathcal{D}(s) \\ &= [\text{definition } \mathcal{D}[\![\cdot]\!]]\ abs \circ \mathcal{D}[\![s]\!] \end{aligned}$$ □

## 3.6 Conclusions and bibliographical remarks

In this chapter an operational semantics and a denotational semantics for a basic process language with probabilistic choice have been given and compared. The notions of compact support measure, finite multisets and several other notions and proof methods used throughout this thesis are introduced. The work in this chapter is included here for an introductory purpose and is mainly based on [107]. It follows the general approach to comparative metric semantics set out in [38].

The construction of a metric space of compact support measures has been introduced in [190] and studied further in [188]. The note [189] provides the basis for the important proof of completeness (cf. lemma 3.3.16) of the space of compact support measures. In [28] a metric space of evaluations is introduced and compared with spaces defined using a set-theoretic and a complete partial order approach. The evaluations introduced in [28] are similar to the compact support measures used in this thesis. The work of Van Breugel and Worrell [53, 52] introduces a pseudo metric on probabilistic processes in which small variations in the probabilities lead to small distances. More details on the comparison of the pseudometric space obtained in this way with the metric space of compact support measures are given below. In [53] a real-valued logic is introduced which can be used to characterize the distance between processes. In this paper a comparison is also given with the pseudo metric introduced by Desharnais, Gupta, Jagadeesan and Panangaden in [77], the metric on compact support measures from [190] also used here and with the pseudometric introduced by Norman in his thesis [167, section 6.1].

Several techniques for modeling a language with probabilistic choice have been introduced in this chapter. The operational semantics is based on a probabilistic transition system specification. The notion of a probabilistic transition system extends Plotkin's structured operational semantics [173] to be able to deal with probabilistic choice. The quantitative information provided by the probabilities causes a technical problem: Multiple occurrences of the same transition cannot be identified with a single occurrence. The solution used in the transition systems in this thesis is to keep the multiplicities of the transitions. The paper [168] also implicitly uses multisets of transitions but does not specify how the multiplicities of transitions are found. Other solutions that have been used are to combine multiple occurrences by summing the probabilities (e.g. [154]) and calculating the probabilities separately (e.g. [10, 7]). After summing the probabilities one can use e.g. sets of labeled transitions [92], discrete probability distributions over transitions [181] to describe the probabilistic steps.

In [181] Segala introduces a notion of probabilistic automaton which is essentially also a probabilistic transition system. The discussion there is restricted to analysis of the probabilistic automata whereas here the specification of a transition system for a program in a process language is also an important step. The probabilistic automata of Segala use measures (called probability distributions in [181]) to describe the probabilistic steps instead of the finite multisets of labeled transitions used here. For finitely branching transition systems the multisets are (slightly) more expressive, but unlike measures they cannot be used to describe infinite probabilistic choice.

In [92] reactive, generative and stratified transition systems are introduced to give a meaning to a language with probabilistic choice. In a reactive system, an action is executed after which a probabilistic choice decides the next state. In a generative system, a single probabilistic choice is made to decide the next action and corresponding next state. In a stratified system, multiple probabilistic choices can be made in deciding the next step. It is shown that the stratified model is more general than the generative model in the sense that there are programs which can be distinguished by the stratified model but not by the generative model. The reverse does not hold, all programs identified by the stratified model are also identified by the generative model. The generative model in turn is more general than the reactive model. The transition system $\mathcal{T}_p$ is generative. Stratified transition systems will also be used in this thesis (e.g. section 4.4). For $\mathcal{L}_p$ a stratified transition system can be given but would yield the same operational model $\mathcal{O}$ as obtained using the generative transition system. Language constructs like e.g. hiding and failure which would be modeled differently by a generative and a stratified transition system are not present in the language $\mathcal{L}_p$. No reactive transition systems are used in this thesis as these systems do not seem to fit with $\mathcal{L}_p$ and the other languages considered.

In this chapter the operational model $\mathcal{O}$ gives the meaning of a program by removing information which is not observable, like the names of the configurations, from the transition system. Another common way to abstract away from irrelevant information of a transition system is to define a notion of bisimulation (cf. [171, 159]). For probabilistic processes the standard form of bisimulation is Larson-Skou bisimulation introduced in [149]. See sections 4.4 and 4.5 for more information on bisimulation.

The operational model $\mathcal{O}$ uses a linear domain of probabilistic processes. The linear domain is built using a functor *Meas* which constructs a complete ultrametric space of measures over a given complete ultrametric space. General measure theoretical results can be found in [102]. In [170] the application of measure theory in modeling concurrency is discussed. The functor *Meas* has been introduced by Rutten and De Vink as $\mathcal{M}'_1$ in [190] and studied further in [188]. The proof of the completeness of the space given by *Meas* is based on [189]. This functor is also used in [107, 109, 110]. In [28] a metric space of evaluations is introduced and compared with spaces defined using a set-theoretic and a complete partial order approach. The evaluations introduced in [28] are similar to the measures used here.

The metric defined on the domain of compact support measures is 'qualitative on the probabilities' rather than quantitative as the pseudo-metric defined in e.g. [53, 52]. The distance on the compact support measures does provide quantitative information by answering the question 'how far are processes apart?'. Two processes are close if they start with exactly the same behavior. The longer they behave exactly the same, the closer they are. The probabilities, however, have to be exactly the same. For example, $\frac{1}{2}\Delta_{aaa} + \frac{1}{2}\Delta_{bbb}$ and $\frac{1}{2}\Delta_{aab} + \frac{1}{2}\Delta_{bbb}$ behave the same in the first two actions. The process $\frac{1}{3}\Delta_{aaa} + \frac{2}{3}\Delta_{bbb}$, however, is considered to behave completely different from these processes as the probabilities are not the same. Van Breugel and Worrell argue that if the difference between the probabilities is small then the processes should be close. For example $\frac{1}{2}\Delta_a + \frac{1}{2}\Delta_b$ should be close to $(\frac{1}{2}+\epsilon)\Delta_a + (\frac{1}{2}-\epsilon)\Delta_b$ for small values of $\epsilon$. In the papers [53, 52] they use the Hutchinson metric to provide quantitative information about the probabilities: A pseudometric space of probabilistic processes is obtained as the final coalgebra of a functor based on the Hutchinson metric. (See e.g. [179, 131] for more on coalgebras.) In this pseudometric space the distance between probabilistic processes is small if the processes start with approximately the same behavior. One can paraphrase the comparison of two linear domains built with either the Hutchinson metric or the De Vink-Rutten metric on compact support measures as: Two processes are close in the former if they produce exactly the same sequences most of the time while processes are close in the latter if they produce approximately the same sequences all of the time. For the work of Van Breugel and Worrell mentioned above the situation is slightly different as they define a distance on transition systems, i.e. on branching structures. The Hutchinson metric is combined with the scaling functor $id_{\frac{1}{2}}$, also used in for the branching domains in this thesis. The distance obtained in this way can be described by saying that two processes are close if they produce approximately the same trees most of the time. This distance is a pseudo-metric in which transition systems have distance zero exactly when they are probabilistically bisimilar in the sense of [149]. For the linear domain $\mathbb{P}_o$ described in this chapter, it may also be possible to define a distance similar to the distance of [53, 52] by using the characterization of processes which will be given in subsection 5.3.2. On the domain $\mathbb{P}_o$ this would probably result in a real metric rather than just a pseudometric. An advantage of the approach of [53, 52] is that their metric can also be used for continuous probabilistic choices. For these choices, however, it is unclear if a version for a linear domain can also be given. (Chapter 7 of this thesis deals with infinite probabilistic choices on a linear domain.)

The denotational semantics uses a branching probabilistic domain. In the branching domain the process remaining after the first step is again a process in the branching domain. This requires a reflexive definition of the domain. Domain equations are used to specify reflexive definitions. In [38] several domain equations are shown to have a unique solution using purely metric arguments. Earlier work on domain equations can be found in [41]. The work presented here is based on [54, 51]. In [54] it is shown that equations using locally contractive functors have a unique solution (up to isomorphism). Further references to work in the field of domain equations can also be found there.

To describe probabilistic choice in a branching domain multisets are used. The functor $\mathcal{MP}_f$ which yields a metric space of finite multisets over a given metric space was first introduced in [107]. The modeling of multisets is derived from the modeling of event structures (e.g. [198, 137]) and partially ordered multisets (e.g. [85, 98]). In [196, 40] a metric is introduced on partially ordered multisets. Although the metric used here is inspired on the metric introduced in these papers, the two metrics are quite different as a partially ordered multiset is used to describe an entire structure while here a multiset is used to describe a single step.

Several papers discussing randomized algorithms and the modeling of probabilistic choice have already been discussed in chapter 1. Here we mention some further work in this area. The paper [100] discusses different techniques in the design of randomized algorithms and illustrates these with several examples such as primality testing and dining philosophers. In [136, 86] the process language *PCCS* is obtained from Milner's *SCCS* by replacing nondeterminism by probability. The paper [136] also introduces time. This work is extended in [67] where a probabilistic version of parallel composition is added. In [155, 153] (finite) probabilistic choice is added to a deterministic version of timed *CSP*. In [151] a true concurrency model and a probabilistic process algebra are given for a language with parallelism and probabilistic choice. No probabilities need be associated with the parallel composition like e.g. in [67] as, unlike in an interleaving model, the addition of parallelism does not introduce nondeterminism in a true concurrency model. Other examples of papers dealing with (purely) probabilistic process languages are [150, 103].

Further related work which deals with both probability and nondeterminism is discussed in chapter 4. Some related work dealing with full abstractness in a probabilistic setting is discussed in chapter 5. Continuous time Markov chains and stochastic process algebra are used in a setting with a continuous form of probability. See chapter 7 for a discussion of work in this area.

Chapter 4

# Combining nondeterminism and probabilistic choice

## 4.1 Introduction

The goal in this chapter is to study the control flow of a schematic language in which the notions of general nondeterminism and of probabilistic choice are combined. The main focus is on the effects of the operators on the control flow rather than on a detailed description of the primitive elements of the computation.

The introduction of probabilistic choice as done in chapter 3 creates the possibility for different flows of control starting from the same situation. A program using probabilistic choice is therefore no longer deterministic. In the literature nondeterministic choice is often removed when introducing probabilistic choice (cf. [21, 183, 168]). When modeling aspects of a system that are not deterministic, probabilistic choice instead of the nondeterministic choice is used. When using probabilistic choice in a program to model some choice in a system, this choice must satisfy the properties of a probabilistic choice: In the statement $a \oplus_\rho b$ it is not known which action will be taken. However, the choice is made according to some given probability, so the relative frequencies of the occurrences of $a$ and $b$ over multiple runs are approximately known. In $n$ runs, the action $a$ will occur approximately $\rho\, n$ times and the action $b$ will occur approximately $(1-\rho)\, n$ times.

Another property of the probabilistic choice is that the choice is unpredictable. With a probabilistic choice, one can be certain that no one will know the result beforehand. This is important e.g. when modeling a game between two players. A third aspect of the probabilistic choice is that the choice is independent. The probabilistic choice is independent in that it cannot be influenced by the current state or the outcomes of other choices in the program. This is contrary to choices a player makes while playing a game: The choices a player makes will definitely depend on the choices made by other players or other events in the game.

As the example of a game between two players already shows, there are forms of choice for which the properties of probabilistic choice do not hold. To allow modeling these kinds of choice nondeterministic choice instead of probabilistic choice is needed. The operator $\Box$ is used for nondeterministic choice. Nondeterministic choice is interpreted as "a choice that is made in some unspecified manner". No information on how the choice is made is assumed. To derive a property for a program with nondeterministic choice, the property should hold for all possible ways of making the choice. It is not always possible to remain this general in the interpretation of the nondeterministic choice, i.e. not making assumptions at all about how the nondeterministic choice is made, and still obtaining usable results. Therefore, at some points assumptions need to be made about the type of nondeterminism that is being considered.

One form of nondeterminism is caused by a choice by a user or an opponent: If in an algorithm the user has to choose between different ways to continue, this is modeled by nondeterministic choice. The program $a \Box b$, for example, gives the user the choice between the execution of action $a$ and action $b$. This type of nondeterministic choice can also be used to describe choices of an opponent when modeling a strategy for a two player game. Generally one will be interested in the best or worst case behavior.

Another form of nondeterminism is a choice in a specification: A specification is given in which one of several options should be available. The implementer of the specification chooses which one to implement. A specification $a \Box b$ can be implemented by the algorithm '$a$' or the algorithm '$b$'. (Note that the program '$a \oplus_\rho b$' also implements the specification $a \Box b$.) More generally one can think of nondeterminism as under-specification. One is not interested in which of the nondeterministic alternatives is chosen, only that 'one of these options is chosen'. This allows removing irrelevant details from a specification.

Using nondeterministic choice to model a choice which is known to be probabilistic, in a situation where this fact is not relevant or where the exact probability is unknown, is also a form of under-specification. Nondeterminism, however, is more general than only probabilistic choice with unknown probability: If for some unknown $\rho$ the program $a \oplus_\rho b$ is run many times and the outcome $a$ occurs often, then it is unlikely that running the program many times more will result in only $b$'s occurring. (Running the program often will almost surely yield an $a$.) The program $a \Box b$, on the other hand, may very well produce only $b$'s in the second series of runs.

A third form of nondeterminism is the nondeterminism caused by parallelism. When two processes are running in parallel, complex timing issues (e.g. racing conditions, relative speeds of computers, connection latencies, etc.) may influence the order in which actions happen.

When modeling choices in a system, it is clear that some choices will have to be modeled with nondeterministic choice, they cannot be assumed to satisfy the properties of the probabilistic choice. As one still wants to use the probabilistic choice where this is possible, both types of choice will have to be combined in a single language. This raises the question how to deal with programs which combine the two choices. In the program $a \Box (b \oplus_{\frac{1}{2}} c)$, for example, a choice has to be made between $a$ and the probabilistic choice between $b$ and $c$. In the program $a \oplus_{\frac{1}{2}} (b \Box c)$ a probabilistic choice has to be made between $a$ and the choice between $b$ and $c$. The question is how these programs should be interpreted. For instance, both the programs may execute the action $a$, but what can be said about the probability with which this occurs ? There are several ways to approach this question. The first option is to always make the nondeterministic choice first. The second is to always make the probabilistic choice first. A third approach is to first make the choice which “appears first”.

The first approach, where the nondeterministic choice is made first, removes the problem of interpreting statements by reducing the statements to purely probabilistic statements. The results from chapter 3 can be used to interpret the purely probabilistic statement. This approach corresponds to an information-oriented way of interpreting nondeterministic choice. The choice should be made with the correct amount of information: In $a \oplus_{\frac{1}{2}} (b \,\square\, c)$ the choice between $b$ and $c$ can be made beforehand as this will not influence the probabilistic choice. It is possible to see this program as a nondeterministic choice between $a \oplus_{\frac{1}{2}} b$ and $a \oplus_{\frac{1}{2}} c$. In other words, the statement $a \oplus_{\frac{1}{2}} (b \,\square\, c)$ is assigned the same meaning as $(a \oplus_{\frac{1}{2}} b) \,\square\, (a \oplus_{\frac{1}{2}} c)$. In $a \,\square\, (b \oplus_{\frac{1}{2}} c)$, the probabilistic choice cannot be made before the nondeterministic choice, as the outcome of the probabilistic choice should not yet be known at the time of making the nondeterministic choice. Think of the program as describing a game where the user wins one for $a$, two for $b$ and nothing for $c$. The program $(a \,\square\, b) \oplus_{\frac{1}{2}} (a \,\square\, c)$, where the probabilistic choice is made before the nondeterministic choice, describes a different game. In the first game the user has to choose between taking the sure one for $a$, or gambling and getting two for $b$ or none for case $c$. In the second game, the user will either have to choose between $a$ and $b$ or between $a$ and $c$, either choice is easy to make. The user is able to achieve higher expected winnings in the second game. Always making nondeterministic choices first is referred to as giving priority to the nondeterminism. In section 4.2, which is based mainly on [107], this approach is followed.

The second approach, where the probabilistic choice has to be made first, corresponds to a resource-oriented way of looking at the nondeterministic choice: The program is going to offer the user some of the actions $a$, $b$ or $c$. The probabilistic choice has to be made to see which actions will actually be offered. The user can then select one of the options. By first making the probabilistic choice, the remaining statement is purely nondeterministic. The usual interpretation of a nondeterministic statement can be used to find the meaning of the remaining statement. Always making probabilistic choices first is referred to as giving priority to the probability. This approach is worked out in section 4.3. This section is based on [107] together with some previously unpublished results. Recently a lot of work in the area of probabilistic process algebra has also been done in this setting, see e.g. [10].

The third approach where the choice which “appears first” is made first is an obvious choice. With the choice that appears first we mean the choice that is the highest in the parse tree or, equivalently, the choice at the front using prefix notation for the operators. See also section 4.4. This approach does not force a certain interpretation of the nondeterministic choice. However, it also does not give a direct way of interpreting statements with both types of choice. The operational processes yielded as the final result in section 4.4 have the same form as those in section 4.2 but are constructed using the third approach; choices are made in the order they are encountered. This section is mainly based on work done in [109, 110].

The operator $\square$ is added to the language to express the nondeterministic choice between two alternatives. As was mentioned above, one type of nondeterminism is the nondeterminism caused by concurrency, i.e. by programs running in parallel. Concurrency is such an important cause of nondeterminism that a special operator $\|$, called merge, is introduced to describe the parallel execution of two programs. The program $s\|s'$ describes that the programs $s$ and $s'$ run concurrently. The interleaving interpretation of parallelism is used: The actions produced by $s\|s'$ are still seen as a single sequence, but at each stage either $s$ or $s'$ may produce the next action. No information about which statement produces the next action is assumed to be available: The choice which statement produces the next action is a nondeterministic choice.

Within concurrent programs, it may be necessary for parallel components to communicate. Communication between programs running in parallel is enabled through the introduction of special synchronization actions. Two programs synchronize by executing matching synchronization actions at the same time. With the introduction of synchronization also comes the possibility of deadlock. A system or component is said to be deadlocked if it is not yet finished but is unable to perform any actions. A component can become deadlocked by trying to execute a synchronization action while no other component executes a matching synchronization action. The symbol $\delta$ is used to denote a deadlocked system.

The presence of deadlock adds yet another distinction; the interpretation of nondeterminism as local or global. The interpretations differ in the manner they deal with potential deadlock. A local nondeterministic choice is made independent of the actions of the environment (i.e. the other parallel components); the choice is made locally, without looking at actions that other components are willing to execute. Global nondeterministic choice may be influenced by the environment; the global choice will "adapt" to the actions the other components are willing to perform. If no other component is willing to synchronize, a global nondeterministic choice will avoid starting with a synchronization action if possible, as this will lead to deadlock. The nondeterminism caused by the operator merge is also interpreted as being local or global. If the nondeterminism is global, a component that wants to synchronize but is unable to at the moment, can wait and let other components execute actions first. If the nondeterminism is interpreted as local, synchronization has to be preformed immediately. If a component tries to synchronize when this is not possible, it will directly lead to deadlock.

The distinction between the local and global interpretation of nondeterminism is well known. In literature the terms internal and external or static and dynamic are also used for local and global (cf. [35, 127]) For probabilistic choice a similar distinction can be made, giving unconditional or conditional probabilistic choice. This will be explained further in section 4.2 below.

The outline of the remainder of this chapter is as follows: In section 4.2 the approach with priority for nondeterminism is worked out by giving an operational and a denotational semantics for a language $\mathcal{L}_{pnd}$ which adds nondeterministic choice and parallel composition to the language $\mathcal{L}_p$ introduced in chapter 3. The relationship between the operational and denotational semantics is also established. Both an unconditional and a conditional interpretation for probabilistic choice are considered. In section 4.3 the approach with priority for probabilistic choice is worked out for the same language $\mathcal{L}_{pnd}$ in a similar manner. Both a local and a global interpretation of nondeterminism are given. In section 4.4 the approach which does not require priority for either nondeterminism or probabilistic choice is worked out for an adapted language $\mathcal{L}^{(2)}_{pnd}$. An unconditional and a conditional interpretation for probabilistic choice are combined with a local and a global interpretation of nondeterminism.

## 4.2 Priority for nondeterminism

In this section a language $\mathcal{L}_{pnd}$ with both nondeterminism and probabilistic choice is introduced. The nondeterminism is introduced by the operator $\Box$ which denotes nondeterministic choice and by the operator $\|$, called merge, which denotes parallel composition. 'Mixed' statements containing both a nondeterministic operator and the probabilistic choice operator are dealt with by giving priority to the nondeterminism. Selecting which alternative of a nondeterministic choice to execute and finding the component in a parallel system which produces the next action is referred to as *resolving* the nondeterminism. Giving priority to the nondeterminism means that all nondeterminism in finding the first action is resolved before making any probabilistic choices or executing any actions.

As explained in the introduction of this chapter, nondeterminism can be interpreted as being local or global and similarly the probabilistic choice can be interpreted as being unconditional or conditional (to be explained in a moment). Combining the different interpretations of nondeterministic and probabilistic choice gives rise to four possible interpretations for the language $\mathcal{L}_{pnd}$. To be global, nondeterministic choice has to react to actions from the environment. However, the nondeterminism has priority and must be resolved before any probabilistic choices are made. If the actions of the environment depend on the outcome of a probabilistic choice, the actions of the environment cannot be known at the time the nondeterministic choice has to be resolved, so the nondeterministic choice cannot depend on them as is necessary for a global choice. This shows that a nondeterministic choice which has priority is not well suited for modeling a global nondeterministic choice. This observation leaves us with two interpretations, local nondeterminism combined with either unconditional or conditional probabilistic choice. For both of these interpretations an operational semantics is given in subsection 4.2.4. A single denotational model, which can be used for both interpretations despite the differences between these interpretations, is given in subsection 4.2.5. The denotational semantics is related with both operational models in subsection 4.2.6.

The distinction between unconditional or conditional probabilistic choice is similar to the distinction between local and global nondeterministic choice. Unconditional probabilistic choice is made independent of the environment. Conditional probabilistic choice, on the other hand, also takes the actions of the environment into account. The probability of a certain alternative being chosen, is a probability given that the alternative does not deadlock, i.e. the probability is a conditional probability. Clearly unconditional probabilistic choice leads to a form of local nondeterminism, while conditional probabilistic choice induces a form of global nondeterminism. The techniques for making the distinction between unconditional and conditional probability are, therefore, similar to the techniques used to describe local and global choice. All that needs to be added is a way to deal with the quantitative information provided by the probabilistic choice. When an alternative of a probabilistic choice deadlocks, normalization of the probabilities for the other alternatives is required.

The distinction between unconditional and conditional probabilistic choice is usually not made. Most papers dealing with probabilistic choice only include the unconditional form of probabilistic choice. As with global nondeterminism, however, the conditional form of probabilistic choice is also useful. For an application of a conditional probabilistic choice in specifying a system consider the following example. A server should execute tasks for two users. The tasks of the first user take twice as long as those of the second user. The server should schedule the tasks of two users in a "fair" way such that both users get approximately equal amounts of server time when both are using the server. Clearly if only one of the users is using the server, this user should get all the server time. This server can easily be specified using conditional probabilistic choice: $server = (serve - first - user \oplus_{\frac{1}{3}} serve - second - user); server$. This server will serve the first user one third and the second user two thirds of the time when both are competing for use of the server. Due to the conditionality of the choice, the server will never try to serve a user which is not requesting use of the server.

## 4.2.1 The syntax of the language $\mathcal{L}_{pnd}$

The language $\mathcal{L}_{pnd}$ extends the language $\mathcal{L}_p$ of chapter 3 with two constructs which have a nondeterministic nature: nondeterministic choice and parallel composition. To allow communication between parallel components, some of the atomic actions are assumed to be synchronization actions. The set *Act* of atomic actions is divided into two disjoint sets: A set of observable actions *OAct* ranged over by $b$ and a set of synchronization actions *Sync* ranged over by $c$.

$$Act \quad = \quad OAct \cup Sync$$

Note that $a$ is still used to range over *Act*, so an action $a$ can be an observable action as well as a synchronization action.

For each synchronization action $c \in Sync$ there is a unique complementary action $\bar{c}$ in *Sync* with which $c$ can synchronize. The complementary action for $\bar{c}$ is $c$, i.e. $\bar{\bar{c}} = c$. The synchronization of the actions $c$ and $\bar{c}$ results in a special observable action $\tau$.

A synchronization action represents an attempt at synchronization and the observable action $\tau$ represents a successful communication. The observable atomic actions (except for $\tau$) represent the observations resulting from the atomic steps in the computation and are left uninterpreted. The meaning of an action $b$ in *OAct* will simply be $b$ itself. As in the language $\mathcal{L}_p$ a set of procedure variables *PVar*, ranged over by $x$, is used for recursion.

**Definition 4.2.1**

*(a) The set of statements Stat, ranged over by s, is given by*

$$s \quad ::= \quad a \mid x \mid s\,;s \mid s \oplus_\rho s \mid s \,\Box\, s \mid s \| s$$

*where* $\rho \in (0,1)$.

*(b) The set of guarded statements GStat, ranged over by g, is given by*

$$g \quad ::= \quad a \mid g\,;s \mid g \oplus_\rho g \mid g \,\Box\, g \mid g \| g$$

*where* $\rho \in (0,1)$.

*(c) The set of declarations Decl, ranged over by $D$, is given by*

$$Decl \quad = \quad PVar \to GStat$$

*(d) The language $\mathcal{L}_{pnd}$ is given by*

$$\mathcal{L}_{pnd} \quad = \quad Decl \times Stat$$

A basic statement is an atomic action or a procedure variable. An atomic action $a$ can be either an observable action or a synchronization action. The operators $;$ and $\oplus_\rho$ are interpreted as for $\mathcal{L}_p$. The operator $;$ denotes sequential composition. The statement $s_1 \,; s_2$ behaves like $s_1$ until $s_1$ terminates after which $s_1 \,; s_2$ continues by behaving like $s_2$.

A declaration $D$ gives the body $D(x)$ for a procedure $x$. The body of a procedure must be a guarded statement. A guarded statement guarantees that at least one atomic action is done before some procedure is called. In the remainder of this section one fixed declaration $D$ is assumed and $D$ is dropped from the notation, e.g. $s \in \mathcal{L}_{pnd}$ is written instead of $(D, s) \in \mathcal{L}_{pnd}$.

The statement $s_1 \,\square\, s_2$ denotes a nondeterministic choice. The statement $s_1$ or the statement $s_2$ will be executed, but it is not known which one. As explained in the introduction of this section, the nondeterminism is local. If one of the two statements deadlocks, it can still be selected.

The operator $\oplus_\rho$ denotes probabilistic choice. The argument $\rho$ denotes the probability that the first alternative is chosen and is assumed to be between 0 and 1. The execution of the statement $s_1 \oplus_\rho s_2$ starts with making a probabilistic choice. With probability $\rho$ the first statement, $s_1$, is selected and executed. With the remaining probability, $1 - \rho$, the second statement, $s_2$, is selected and executed. As explained in the introduction, the probabilistic choice can be unconditional or conditional. If the choice is conditional and the selected statement fails, the other option is selected instead. If the choice is unconditional then it is not possible to recover from a failure of the selected statement.

The statement $s_1 \| s_2$ denotes parallel composition. Parallel composition also introduces nondeterminism: The first action may be taken by $s_1$ or it may be taken by $s_2$ or the two statements may synchronize. A failed synchronization attempt (from either $s_1$ or $s_2$) will result in deadlock. A failed synchronization attempt from $s_1$ is an action $c$ from $s_1$ without a matching action $\bar{c}$ from $s_2$.

**Example 4.2.2** *The following are all valid statements (in Stat): $(a \,\square\, b) \oplus_\rho c$, $(a \oplus_\rho c) \,\square\, (b \oplus_\rho c)$, $(b_1 \oplus_\rho c) \| (b_2 \oplus_\rho \bar{c})$, $(a \oplus_\rho b); x$ and $x; b$. All but the last are also guarded statements (in GStat) and can, therefore, be used as the body of a procedure variable, e.g. $D(x) = (a \oplus_\rho b); x$.*

*As a single declaration $D$ is assumed to be fixed, each of these statements also represents a program, e.g. $x; b$ also denotes the program $(D, x; b)$.*

A program in $\mathcal{L}_{pnd}$ can be used to describe a game against some opponent. Probabilistic choices are used to model probabilistic events within the game. The choices of the opponent are modeled by nondeterministic choices. In this way no assumptions are made on how the choices of the opponent are made. Any result derived will hold for every possible way the opponent can play.

**Example 4.2.3** *As an example of a game consider the so called 'three doors problem': In a quiz-show a prize is hidden behind one of three doors. The contestant gets to pick a door. After the contestant picks a door the quiz-master will open one of the other doors to show that the prize is not located there. Now the contestant has to choose to stick to the first selected door or to change to the other remaining closed door. The question is which strategy maximizes the probability of winning the prize.*

*To model this game, the door with the prize behind it is called $a$ and the other two doors are called $b$ and $c$. The strategy where the player first selects a random door and then sticks with this door can be described as follows: First a random door is selected giving $a \oplus_{\frac{1}{3}} (b \oplus_{\frac{1}{2}} c)$ as the description of the first step. Note that each door is selected with probability $\frac{1}{3}$. Another equivalent way of modeling this first step is e.g. $(a \oplus_{\frac{1}{2}} b) \oplus_{\frac{2}{3}} c$. In the next step one of the other doors is opened. The door opened cannot be $a$, nor can it be the door selected by the player giving $(a; (b \Box c)) \oplus_{\frac{1}{3}} ((b; c) \oplus_{\frac{1}{2}} (c; b))$. As the door is opened by the quiz-master and not by the contestant nondeterministic choice is used to model the selection of a door. Finally the contestant sticks with the first selected door giving*

$$s_1 = (a; (b \Box c); a) \oplus_{\frac{1}{3}} ((b; c; b) \oplus_{\frac{1}{2}} (c; b; c))$$

*as the complete description of this strategy.*

*The second strategy, where the player switches doors, can be described similarly. The first two steps are the same giving $(a; (b \Box c)) \oplus_{\frac{1}{3}} ((b; c) \oplus_{\frac{1}{2}} (c; b))$ as a description for these steps. Next the player selects the other closed door. The remaining closed door is the door that is not selected by the player in the first step and not selected by the quiz-master in the second step resulting in*

$$s_2 = (a; ((b; c) \Box (c; b)) \oplus_{\frac{1}{3}} ((b; c; a) \oplus_{\frac{1}{2}} (c; b; a))$$

*as the complete description of this strategy.*

This example shows how two strategies for playing a game against an opponent can be described by programs in $\mathcal{L}_{pnd}$. To be able to solve the 'three doors problem', i.e. to find which strategy is best, we will look at the semantics of these programs in example 4.2.24 below.

### 4.2.2 A transition system with priority for nondeterministic choice: $\mathcal{T}_{pnd}$

In this subsection a transition system $\mathcal{T}_{pnd}$ is given for the language $\mathcal{L}_{pnd}$ which gives priority to the nondeterministic choices. In finding which action the system will do next, first all nondeterministic choices are resolved and only then the probabilistic choices are made.

To be able to deal with parallel composition, two auxiliary operators, $\lfloor\!\lfloor$ and $|$, are added to the language (following the approach of [43]). The operator $\lfloor\!\lfloor$ is called leftmerge and $|$ is called synchronize. The extended set of statements obtained by adding these operators is denoted by $Stat^+$ and also ranged over by $s$. Thus for $Stat^+$ we extend part (a) of definition 4.2.1 by putting

$$s \quad ::= \quad \ldots \mid s \lfloor\!\lfloor s \mid s|s$$

The first action taken by the statement $s_1 \| s_2$ can be produced by $s_1$ or $s_2$ separately or by a synchronization between $s_1$ and $s_2$. The operator $\mathbin{|\!\lfloor}$ is used to describe the first situation. The statement $s_1 \mathbin{|\!\lfloor} s_2$ is like the statement $s_1 \| s_2$ except that the first step must come from $s_1$. (Synchronizing with $s_2$ is not allowed.) Similarly the statement $s_1 | s_2$ behaves like $s_1 \| s_2$ except that the first step must be a synchronization between $s_1$ and $s_2$. Combining the three possibilities for the first step of $s_1 \| s_2$ we get (as in [43])

$$s_1 \| s_2 \quad \text{is equivalent with} \quad (s_1 \mathbin{|\!\lfloor} s_2) \square (s_2 \mathbin{|\!\lfloor} s_1) \square (s_1 | s_2)$$

The information required to describe the state of an execution of a program in $\mathcal{L}_{pnd}$ is the part of the program that remains to be executed. A *resumption* is used to describe the remainder of a program. The set of resumptions is denoted by *Res* and ranged over by $r$. The remainder of a program is either another program or nothing, in case the execution is finished. A resumption, therefore, is either a statement $s$ in $\mathit{Stat}^+$ or a special symbol E denoting a finished computation

$$r \quad ::= \quad s \mid \mathrm{E}$$

A *configuration* in $\mathcal{T}_{pnd}$ is a resumption $r$ together with a declaration $D$, i.e. $\mathit{Conf} = \mathit{Decl} \times \mathit{Res}$. As with programs, the declaration part is dropped from the notation as a single declaration is assumed to be fixed.

To execute the statement $s_1 \square s_2$, the nondeterministic choice has to be made between executing $s_1$ and executing $s_2$. Recall that making this choice is referred to as resolving the nondeterministic choice or more generally as resolving the nondeterminism. Usually the nondeterminism is resolved implicitly. The transitions that $s_1 \square s_2$ can take are those of $s_1$ and those of $s_2$; the nondeterminism is implicitly resolved when one of the transitions is taken. The following example shows that simply allowing the transitions of both alternatives cannot be used to add nondeterminism to the transition system $\mathcal{T}_p$ introduced in chapter 3 (see definition 3.3.3).

**Example 4.2.4** *Consider the statement*

$$s = (a_1 \oplus_\rho a_2) \square (a_3 \oplus_\sigma a_4)$$

*The transitions system $\mathcal{T}_p$ of chapter 3 gives the transitions $(a_1 \oplus_\rho a_2) \xrightarrow{\rho \cdot a_1} \mathrm{E}$, $(a_1 \oplus_\rho a_2) \xrightarrow{(1-\rho)\cdot a_2} \mathrm{E}$, $(a_3 \oplus_\sigma a_4) \xrightarrow{\sigma \cdot a_3} \mathrm{E}$ and $(a_3 \oplus_\sigma a_4) \xrightarrow{(1-\sigma)\cdot a_4} \mathrm{E}$.*

*Taking $s \xrightarrow{\rho \cdot a_1} \mathrm{E}$, $s \xrightarrow{(1-\rho)\cdot a_2} \mathrm{E}$, $s \xrightarrow{\sigma \cdot a_3} \mathrm{E}$ and $s \xrightarrow{(1-\sigma)\cdot a_4} \mathrm{E}$ as the transitions for $s$ would give a total probability of 2 instead of 1.*

To be able to assign a probability to $a_1$ in the statement $s$ from the example above, it is necessary to first resolve the nondeterminism. If the first alternative $a_1 \oplus_\rho a_2$ is chosen then the probability of $a_1$ is $\rho$, otherwise it is 0 (assuming distinct actions $a_1, a_2, a_3, a_4$).

After resolving the nondeterminism and finding the probabilities of each action it has to be clear which actions belong to which nondeterministic alternative. The auxiliary action $\nu$ is used to distinguish between the different nondeterministic alternatives. All nondeterminism is explicitly resolved resulting in a $\nu$ transition. The $\nu$ transitions are not interpreted as real actions, only auxiliary steps in the transition system. The $\nu$ transitions, as auxiliary steps, are not part of the observable behavior of a statement. When giving the operational semantics, $\mathcal{O}[\![\bullet]\!]$, the $\nu$ transitions are removed. (Other mechanisms to bundle together transitions, instead of using auxiliary steps, are also possible, for example by using the hyper transitions systems of [48, 49].)

A statement which does not need to make anymore nondeterministic choices before the first action, is called *resolved*. As nondeterministic choices are to be made first, resolving the nondeterminism has priority and only resolved statements should be able to make probabilistic choices and produce actions.

**Definition 4.2.5** *The statement $a$ is resolved. The statement $x$ is resolved whenever $D(x)$ is resolved. The statements $s_1; s_2$ and $s_1 \,⫡\, s_2$ are resolved exactly when $s_1$ is resolved. The statements $s_1 \oplus_\rho s_2$ and $s_1|s_2$ are resolved if and only if both $s_1$ and $s_2$ are resolved.*

For $s = s_1; s_2$ and $s = s_1 \,⫡\, s_2$ the first action must come from $s_1$, so it is sufficient to resolve the nondeterminism in $s_1$ to find the possible first actions of $s$. For $s = s_1 \oplus_\rho s_2$ and $s = s_1|s_2$ both $s_1$ and $s_2$ play a role in finding the first action that is executed. Both $s_1$ and $s_2$ must be resolved for $s$ to be resolved.

The labels used in the transition system $\mathcal{T}_p$ for the language $\mathcal{L}_p$ are actions together with the probability that the action occurs. As the nondeterminism in the language $\mathcal{L}_{pnd}$ is resolved explicitly by means of $\nu$-transitions, the symbol $\nu$ is added to these labels for the transition system $\mathcal{T}_{pnd}$. Recall that $PAct = [0,1] \times Act$ and that $\alpha$ ranges over $PAct$, i.e. $\alpha = \rho \cdot a$ for some $\rho \in [0,1]$ and $a \in Act$. As $Act$ is split into $OAct$ and $Sync$, the set $PAct$ can be separated into the set $POAct$ of pairs in $[0,1] \times OAct$, ranged over by $\beta$ and the set $PSAct$ of pairs in $[0,1] \times Sync$.

$$PAct \;=\; [0,1] \times Act \;=\; [0,1] \times OAct + [0,1] \times Sync$$

The set of all transition labels, i.e. $PAct \cup \{\,\nu\,\}$, is ranged over by $\theta$, thus $\theta = \nu$ or $\theta = \alpha$ for some $\alpha \in PAct$.

**Definition 4.2.6** *The transition system $\mathcal{T}_{pnd}$ is given by $\mathcal{T}_{pnd} = (Conf, PAct \cup \{\,\nu\,\}, \to, Spec)$. A transition $(r, \theta, r') \in \to$ is written as $r \xrightarrow{\theta} r'$. The specification Spec is given below, divided into four parts.*

The first part of the specification are the axioms and rules dealing with actions, recursion and sequential composition.

- $$a \xrightarrow{1\cdot a} \mathrm{E} \qquad \text{(Act)}$$
- $$\frac{s_1 \xrightarrow{\theta} r}{s_1; s_2 \xrightarrow{\theta} r; s_2} \qquad \text{(Seq)}$$
- $$\frac{D(x) \xrightarrow{\theta} r}{x \xrightarrow{\theta} r} \qquad \text{(Rec)}$$

where $r; s_2$ in rule (Seq) should be read as $s_2$ if $r = \mathrm{E}$.

These rules have not changed from the rules in the transition system $\mathcal{T}_p$ except that $\theta$ replaces $\alpha$ as a variable denoting an arbitrary observation. The statement $a$ results in the action $a$ with probability one after which the execution is finished. To execute a procedure $x$, the body of the procedure $D(x)$ has to be executed. The statement $s_1 \,;s_2$ behaves like $s_1$ until $s_1$ is done (the case that $r = \mathrm{E}$) after which it behaves like $s_2$.

The second part of the specification are the axioms dealing with the resolution of nondeterminism.

- $s_1 \,\Box\, s_2 \xrightarrow{\nu} s_1$ (Choice 1)
  $s_1 \,\Box\, s_2 \xrightarrow{\nu} s_2$ (Choice 2)
- $s_1 \| s_2 \xrightarrow{\nu} s_1 \,\|\!\!\lfloor\, s_2$ (Intro $\|\!\!\lfloor$ 1)
  $s_1 \| s_2 \xrightarrow{\nu} s_2 \,\|\!\!\lfloor\, s_1$ (Intro $\|\!\!\lfloor$ 2)
  $s_1 \| s_2 \xrightarrow{\nu} s_1 | s_2$ (Intro $|$)

The nondeterminism is introduced by the operators $\Box$ and $\|$. The nondeterminism is resolved explicitly resulting in a $\nu$-transition. The axioms (Choice 1) and (Choice 2) are clear. The axioms (Intro $\|\!\!\lfloor$ 1), (Intro $\|\!\!\lfloor$ 2) and (Intro $|$) give the three possibilities for execution of the parallel composition using the auxiliary operators leftmerge $\|\!\!\lfloor$ and synchronize $|$.

The third part of the transition system are the rules for the auxiliary operators $\|\!\!\lfloor$ and $|$.

- $$\frac{s_1 \xrightarrow{\nu} s}{s_1 \,\|\!\!\lfloor\, s_2 \xrightarrow{\nu} s \,\|\!\!\lfloor\, s_2} \qquad \text{(Leftmerge } \nu\text{)}$$
  $$s_1 | s_2 \xrightarrow{\nu} s | s_2 \qquad \text{(Sync } \nu \text{ 1)}$$
  $$s_2 | s_1 \xrightarrow{\nu} s_2 | s \qquad \text{(Sync } \nu \text{ 2)}$$
- $$\frac{s_1 \xrightarrow{\alpha} r}{s_1 \,\|\!\!\lfloor\, s_2 \xrightarrow{\alpha} r \| s_2} \qquad \text{(Leftmerge)}$$
- $$\frac{s_1 \xrightarrow{\rho\cdot c} r_1 \quad s_2 \xrightarrow{\sigma\cdot\bar{c}} r_2}{s_1 | s_2 \xrightarrow{\rho\sigma\cdot\tau} r_1 \| r_2} \qquad \text{(Sync)}$$

where $r \| s_2$ in rule (Leftmerge) should be read as $s_2$ if $r = \mathrm{E}$ and $r_1 \| r_2$ in rule (Sync) should be read as $r_1$ if $r_2 = \mathrm{E}$ and as $r_2$ if $r_1 = \mathrm{E}$.

The statement $s_1 \,\|\!\!\lfloor\, s_2$ first resolves the nondeterminism in $s_1$. After the first step this results in $s \,\|\!\!\lfloor\, s_2$. The leftmerge is still maintained since resolving the nondeterminism has not produced any actions, only auxiliary $\nu$ steps. If all nondeterminism in $s_1$ has been resolved, $s_1 \mathbin{\|\!\!\lfloor} s_2$ takes the same action as $s_1$. After the first action, the execution continues with the parallel composition $\|$ of the resulting resumption $r$ and $s_2$. If $r$ is equal to E then the computation of the first component is finished and the parallel composition continues with executing $s_2$ by itself.

The configuration $s_1|s_2$ resolves the nondeterminism in both $s_1$ and $s_2$ and then tries to synchronize on the first step. Failure to synchronize will result in deadlock. Because of the possibility of deadlock, the probability that a statement $s_1|s_2$ takes any step may be less than one. In the operational model, $\mathcal{O}$, this "missing probability" will be interpreted as deadlock for unconditional probability. For the conditional interpretation, the missing probability is divided over the available alternatives. Recall that $\rho\sigma$ denotes the multiplication of the real numbers (probabilities) $\rho$ and $\sigma$.

The last part of the specification contains the rules for probabilistic choice.

- $$\frac{s_1 \xrightarrow{\nu} s}{\begin{array}{ll} s_1 \oplus_\rho s_2 \xrightarrow{\nu} s \oplus_\rho s_2 & \text{(Chance } \nu \text{ 1)} \\ s_2 \oplus_\rho s_1 \xrightarrow{\nu} s_2 \oplus_\rho s & \text{(Chance } \nu \text{ 2)} \end{array}}$$

- $$\frac{s_1 \xrightarrow{\sigma \cdot a} s \quad s_1, s_2 \text{ resolved}}{\begin{array}{ll} s_1 \oplus_\rho s_2 \xrightarrow{\rho\sigma \cdot a} s & \text{(Chance } \alpha \text{ 1)} \\ s_2 \oplus_\rho s_1 \xrightarrow{(1-\rho)\sigma \cdot a} s & \text{(Chance } \alpha \text{ 1)} \end{array}}$$

The statement $s_1 \oplus_\rho s_2$ first resolves the nondeterminism in $s_1$ and $s_2$ in any order. In the rules (Chance $\alpha$ 1) and (Chance $\alpha$ 2) the priority for resolving nondeterminism is stated explicitly: The probabilistic choice is only made for resolved statements. If all nondeterminism in both $s_1$ and $s_2$ has been resolved $s_1 \oplus_\rho s_2$ acts like $s_1$ with probability $\rho$ and like $s_2$ with probability $1 - \rho$.

**Example 4.2.7** *Combining the transitions obtained from the proof trees*

$$\dfrac{\dfrac{}{(a \Box b) \xrightarrow{\nu} a}\ \textit{(Choice 1)}}{(a \Box b) \oplus_\rho c \xrightarrow{\nu} a \oplus_\rho c}\ \textit{(Chance } \nu \textit{ 1)}$$

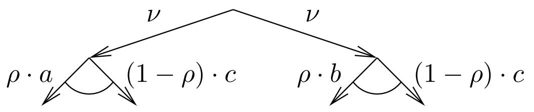
*and*

$$\dfrac{}{(a \oplus_\rho c) \Box (b \oplus_\rho c) \xrightarrow{\nu} a \oplus_\rho c}\ \textit{(Choice 1)}$$

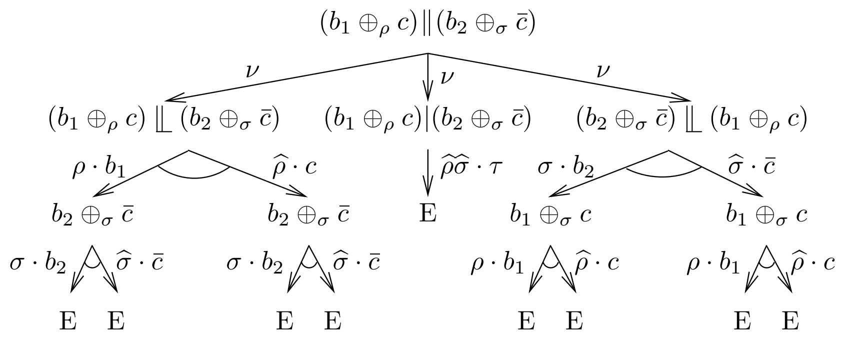
*with the transitions obtained from the symmetric case using (Choice 2) already shows that statements $(a \Box b) \oplus_\rho c$ and $(a \oplus_\rho c) \Box (b \oplus_\rho c)$ have the same transition trees except for the name of the root node. The abstract transition trees for these programs are the same (see* *the figure below) as the names of the nodes are not present in abstract transition trees. The following two proof trees give the transitions for $a \oplus_\rho c$.*

$$\dfrac{\dfrac{}{a \xrightarrow{1\cdot a} \mathrm{E}}\ \textit{(Act)}}{(a \oplus_\rho c) \xrightarrow{\rho\cdot a} \mathrm{E}}\ \textit{(Chance } \alpha \textit{ 1)} \qquad \dfrac{\dfrac{}{c \xrightarrow{1\cdot c} \mathrm{E}}\ \textit{(Act)}}{(a \oplus_\rho c) \xrightarrow{(1-\rho)\cdot c} \mathrm{E}}\ \textit{(Chance } \alpha \textit{ 2)}$$

*The transitions for $b \oplus_\rho c$ are obtained in the same way. This completes the transition tree for both $(a \,\square\, b) \oplus_\rho c$ and $(a \oplus_\rho c) \,\square\, (b \oplus_\rho c)$. The following picture shows the abstract transition tree for these statements.*



*The transition tree for the statement $(b_1 \oplus_\rho c) \,\|\, (b_2 \oplus_\sigma \bar{c})$ is shown in the following picture. In this picture $\widehat{\rho}$ is used as a shorthand for $(1-\rho)$ and similarly $\widehat{\sigma}$ is short for $(1-\sigma)$.*



$(b_1 \oplus_\rho c) \,\|\, (b_2 \oplus_\sigma \bar{c})$
$\nu$
$\nu$
$\nu$
$(b_1 \oplus_\rho c) \,\|\!\!\_\; (b_2 \oplus_\sigma \bar{c})$
$(b_1 \oplus_\rho c) | (b_2 \oplus_\sigma \bar{c})$
$(b_2 \oplus_\sigma \bar{c}) \,\|\!\!\_\; (b_1 \oplus_\rho c)$
$\rho \cdot b_1$
$\widehat{\rho} \cdot c$
$\widehat{\rho}\widehat{\sigma} \cdot \tau$
$\sigma \cdot b_2$
$\widehat{\sigma} \cdot \bar{c}$
$b_2 \oplus_\sigma \bar{c}$
$b_2 \oplus_\sigma \bar{c}$
E
$b_1 \oplus_\sigma c$
$b_1 \oplus_\sigma c$
$\sigma \cdot b_2$
$\widehat{\sigma} \cdot \bar{c}$
$\sigma \cdot b_2$
$\widehat{\sigma} \cdot \bar{c}$
$\rho \cdot b_1$
$\widehat{\rho} \cdot c$
$\rho \cdot b_1$
$\widehat{\rho} \cdot c$
E E
E E
E E
E E

*The first step is to resolve the nondeterminism introduced by the operator $\|$. The next step is taking the actions with appropriate probabilities.*

In the following example the transitions of the two programs describing the strategies for the three doors problem are given.

**Example 4.2.8** *Consider the program $s_1$, where*

$$s_1 \;=\; (a; (b \,\square\, c); a) \oplus_{\frac{1}{3}} ((b; c; b) \oplus_{\frac{1}{2}} (c; b; c))$$

*which was introduced in example 4.2.3. Using $a \xrightarrow{1\cdot a} \mathrm{E}$ obtained from axiom (Act) as premise for rule (Seq) gives the transition $a; (b \,\square\, c); a \xrightarrow{1\cdot a} (b \,\square\, c); a$. The transitions $b; c; b \xrightarrow{1\cdot b} c; b$ and $c; b; c \xrightarrow{1\cdot c} b; c$ can be derived similarly. Using rules (Chance $\alpha$ 1) and (Chance $\alpha$ 2) gives the transitions $(b; c; b) \oplus_{\frac{1}{2}} (c; b; c) \xrightarrow{\frac{1}{2}\cdot b} c; b$ and $(b; c; b) \oplus_{\frac{1}{2}} (c; b; c) \xrightarrow{\frac{1}{2}\cdot c} b; c$.*

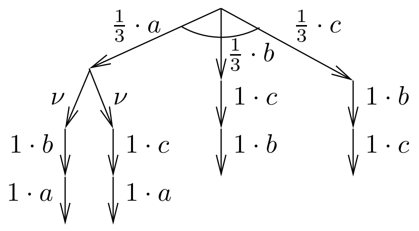
*Using these rules again gives all the first steps of the program $s_1$.*

$$\begin{aligned}
(a;(b\Box c);a)\oplus_{\frac{1}{3}}((b;c;b)\oplus_{\frac{1}{2}}(c;b;c)) &\xrightarrow{\frac{1}{3}\cdot a} (b\Box c);a\\
(a;(b\Box c);a)\oplus_{\frac{1}{3}}((b;c;b)\oplus_{\frac{1}{2}}(c;b;c)) &\xrightarrow{\frac{1}{3}\cdot b} c;b\\
(a;(b\Box c);a)\oplus_{\frac{1}{3}}((b;c;b)\oplus_{\frac{1}{2}}(c;b;c)) &\xrightarrow{\frac{1}{3}\cdot c} b;c
\end{aligned}$$

*The transitions for the program $(b \Box c);a$ are obtained by using rules (Choice 1) and (Choice 2).*

$$\begin{aligned}
(b\Box c);a &\xrightarrow{\nu} b;a\\
(b\Box c);a &\xrightarrow{\nu} c;a
\end{aligned}$$

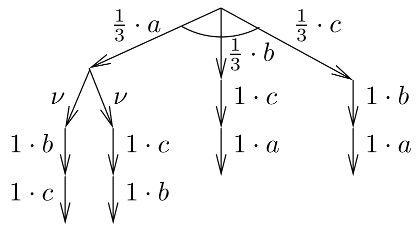
*The transitions for the programs $b;a$, $c;a$, $c;b$ and $b;c$ are found by using axiom (Act) and using rule (Seq). Combining all these transitions gives the following abstract transition tree for $s_1$*



*For the second program $s_2$ introduced in example 4.2.3,*

$$s_2 = (a;((b;c)\Box(c;b))\oplus_{\frac{1}{3}}((b;c;a)\oplus_{\frac{1}{2}}(c;b;a))$$

*the transitions can be found in a similar way resulting in the following abstract transition tree*



### 4.2.3 Properties of the transition system

In this subsection some notation is introduced and several properties of the transition system are shown. Structural induction is not applicable for the proofs of these properties.

Induction on the complexity of a resumption is used instead of induction on its syntactical structure. The complexity of a resumption is expressed by a weight function *wgt*. The weight function is an extension of the weight function used in chapter 3.

**Definition 4.2.9** *The complexity function* $wgt\colon Res \to \mathbb{N}$ *is given by*

$$\begin{array}{rcl} wgt(\mathrm{E}) & = & 0 \\ wgt(a) & = & 1 \\ wgt(x) & = & wgt(D(x)) + 1 \\ wgt(s_1\,;s_2) & = & wgt(s_1) + 1, \quad \textit{and similarly for } \lfloor\!\lfloor \\ wgt(s_1 \,\Box\, s_2) & = & 1 + wgt(s_1) + wgt(s_2), \quad \textit{and similarly for } \oplus_\rho \textit{ and } | \\ wgt(s_1 \,\|\, s_2) & = & wgt(s_1 \lfloor\!\lfloor s_2) + wgt(s_2 \lfloor\!\lfloor s_1) + wgt(s_1|s_2) + 1 \end{array}$$

That the weight function *wgt* is well-defined is easy to see by structural induction, first on guarded statements and then on all resumptions.

**Example 4.2.10** *As* $wgt((b_1 \oplus_\rho c) \lfloor\!\lfloor (b_2 \oplus_\sigma \bar{c})) = wgt(b_1 \oplus_\rho c) + 1 = 3 + 1 = 4$ *and* $wgt((b_1 \oplus_\rho c)|(b_2 \oplus_\sigma \bar{c})) = wgt(b_1 \oplus_\rho c) + wgt(b_2 \oplus_\sigma \bar{c}) = 3 + 3 = 6$ *we have that* $wgt((b_1 \oplus_\rho c) \,\|\, (b_2 \oplus_\sigma \bar{c})) = wgt((b_1 \oplus_\rho c) \lfloor\!\lfloor (b_2 \oplus_\sigma \bar{c})) + wgt((b_2 \oplus_\sigma \bar{c}) \lfloor\!\lfloor (b_1 \oplus_\rho c)) + wgt((b_1 \oplus_\rho c)|(b_2 \oplus_\sigma \bar{c})) = 4 + 4 + 6 = 14$

*As* $wgt(a \oplus_\rho b) = 3$ *we have that* $wgt((a \oplus_\rho b); x) = 4$ *and if* $D(x) = (a \oplus_\rho b); x$ *then* $wgt(x) = wgt(D(x)) + 1 = 4 + 1 = 5$.

Sometimes one is only interested in the existence of a transition with a given observation, not in the resulting configuration. The notation $s \xrightarrow{\theta}$ is used to express that there exists some resumption $r$ such that $s \xrightarrow{\theta} r$ and $s \overset{\theta}{\not\to}$ denotes that no such resumption exists.

The notation $s \overset{\nu}{\leadsto} s'$ is used to denote that $s'$ is one of the nondeterministic alternatives of $s$ and that $s'$ is resolved.

**Definition 4.2.11** *For any statements* $s$, $s'$ *in* $Stat^+$ *the following notation is introduced*

$$\begin{array}{rcl} s \xrightarrow{\nu^+} s' & \Longleftrightarrow & \exists s'' : s \xrightarrow{\nu} s'' \wedge s'' \xrightarrow{\nu^*} s' \\ s \xrightarrow{\nu^*} s' & \Longleftrightarrow & s = s' \vee s \xrightarrow{\nu^+} s' \\ s \overset{\nu}{\leadsto} s' & \Longleftrightarrow & s \xrightarrow{\nu^*} s' \wedge s' \overset{\nu}{\not\to} \end{array}$$

The notation $s \xrightarrow{\nu^+} s'$ expresses that $s$ can go to $s'$ by taking one or more $\nu$ steps. If $s$ can reach $s'$ in zero or more $\nu$ steps $s \xrightarrow{\nu^*} s'$ is written. As already mentioned above, $s \overset{\nu}{\leadsto} s'$ denotes that $s$ can reach the resolved statement $s'$ by $\nu$ steps. Correctness of the recursive definition of $\xrightarrow{\nu^+}$ and $\xrightarrow{\nu^*}$ follows directly from the fact that if $s \xrightarrow{\nu} s'$ then the weight of $s'$ is less than the weight of $s$ (see lemma 4.2.16 below).

Resolving nondeterminism is done by taking $\nu$ steps. The notion of a statement being resolved is supposed to describe that all nondeterminism before taking the first action has been resolved. Resolved statements should, therefore, be exactly those statements that do not take any $\nu$ steps. The following lemma states that this is indeed the case.

**Lemma 4.2.12** *A statement $s$ can take a $\nu$ step exactly when it is not resolved: $s \xrightarrow{\nu} \iff s$ not resolved.*

**Proof** Clear by induction on the weight of the statement. $\Box$

**Example 4.2.13** *Let, in this example, $a$, $b$, $c$, $d$ be actions in Act and $\rho$ a ratio in $(0,1)$. As the statement $a$ is resolved, it cannot take any $\nu$ steps, so $a \overset{\nu}{\rightsquigarrow} a$.*

*As the statement $a \oplus_\rho b$ is resolved it cannot take any $\nu$ steps. The transition $(a \Box b) \oplus_\rho c \xrightarrow{\nu} a \oplus_\rho c$, was derived in example 4.2.7. Combining these two gives that $(a \Box b) \oplus_\rho c \overset{\nu}{\rightsquigarrow} a \oplus_\rho c$ holds.*

*A possible sequence of transitions for the program $(a \Box b) \| (c \Box d)$ is given by $(a \Box b) \| (c \Box d) \xrightarrow{\nu} (a \Box b) \,⫫\, (c \Box d) \xrightarrow{\nu} a \,⫫\, (c \Box d)$. Therefore we have that $(a \Box b) \| (c \Box d) \xrightarrow{\nu^+} a \,⫫\, (c \Box d)$ holds and, as the statement $a \,⫫\, (c \Box d)$ is resolved, also $(a \Box b) \| (c \Box d) \overset{\nu}{\rightsquigarrow} a \,⫫\, (c \Box d)$ holds.*

The transitions for a resolved statement can be described by a multiset. For each transition a (probability, action) pair together with the resumption resulting from executing this action is included in the multiset. This multiset is called the probabilistic successor set of the statement. For a general statement it may be necessary to first resolve some nondeterminism. The successor set of a statement is therefore a set of multisets, where each element of the set is the probabilistic successor set of one of the nondeterministic alternatives of the statement.

**Definition 4.2.14** *For a resolved statement $s$ in $Stat^+$ the probabilistic successor (multi)set of $s$, denoted by $Suc'(s)$, is a multiset consisting of the pairs of the transition label and resulting resumption of all transitions that $s$ can take. The observable probabilistic successor set, denoted by $Suc'_{obs}$ is similar except that only transitions with an observable action as label are considered. The functions $Suc' : Stat_{res} \to \mathcal{MP}_f(PAct \times Res)$ and $Suc'_{obs} : Stat_{res} \to \mathcal{MP}_f(POAct \times Res)$ (with $\mathcal{MP}_f$ as introduced in definition 3.2.4) are given by*

$$\begin{aligned} Suc'(s) &= \{\!| \langle \alpha, r \rangle \mid s \xrightarrow{\alpha} r |\!\} \\ Suc'_{obs}(s) &= \{\!| \langle \beta, r \rangle \mid s \xrightarrow{\beta} r |\!\} \end{aligned}$$

*Note that, as $s \xrightarrow{\alpha} r$ is a different notation for $(s, \alpha, r) \in \to$, this definition uses the notation for multisets introduced in definition 3.2.5. As a result $\langle \alpha, r \rangle \in_n Suc'(s)$ holds exactly when the multiplicity of $s \xrightarrow{\alpha} r$ is $n$.*
*For a statement $s$ in $Stat^+$ the (observable) successor set of $s$, which is denoted by $Suc(s)$ ($Suc_{obs}(s)$), is the set containing the (observable) probabilistic successor sets of the statements that $s$ can reach by $\nu$ steps. The functions $Suc : Stat \to \mathcal{P}_f(\mathcal{MP}_f(PAct \times Res))$ and $Suc_{obs} : Stat \to \mathcal{P}_f(\mathcal{MP}_f(OAct \times Res))$ (where $\mathcal{P}_f(X)$ denotes the space of all finite subsets of $X$) are given by*

$$\begin{aligned} Suc(s) &= \{ Suc'(s') \mid s \overset{\nu}{\rightsquigarrow} s' \} \\ Suc_{obs}(s) &= \{ Suc'_{obs}(s') \mid s \overset{\nu}{\rightsquigarrow} s' \} \end{aligned}$$

The probabilistic successor set for a resolved statement $s$ is similar to the successor sets used in the previous chapter. As an aside: For statements in $\mathcal{L}_p$ the two even coincide: $Suc'(s) = Suc_p(s)$ for $s \in \mathcal{L}_p$, where $Suc_p(s)$ denotes the successor sets used in the previous chapter. As a general statement in $\mathcal{L}_{pnd}$ may have to resolve some nondeterminism before the probabilistic-action steps can be found, the successor of a statement is a set. The probabilistic successor set of any statement $s'$ that $s$ can resolve into ($s \overset{\nu}{\rightsquigarrow} s'$) is included in the successor set of $s$.

**Example 4.2.15** *Taking an observable action $b$ in OAct and a synchronization action $c$ in Sync we have that the transitions for the resolved statement $b \oplus_\rho c$ are $b \oplus_\rho c \xrightarrow{\rho \cdot b} \mathrm{E}$ and $b \oplus_\rho c \xrightarrow{(1-\rho)\cdot c} \mathrm{E}$, so the probabilistic successor set $Suc'(b \oplus_\rho c)$ is given by $Suc'(b \oplus_\rho c) = \{\!|\, \langle \rho \cdot b, \mathrm{E}\rangle, \langle (1-\rho)\cdot c, \mathrm{E}\rangle \,|\!\}$. The observable probabilistic successor set $Suc'_{obs}(b \oplus_\rho c)$ is given by $Suc'_{obs}(b \oplus_\rho c) = \{\!|\, \langle \rho \cdot b, \mathrm{E}\rangle \,|\!\}$. The successor set $Suc(b \oplus_\rho c)$ is given by $Suc(b \oplus_\rho c) = \{\, \{\!|\, \langle \rho \cdot b, \mathrm{E}\rangle, \langle (1-\rho) \cdot c, \mathrm{E}\rangle \,|\!\} \,\}$. The observable successor set $Suc_{obs}(b \oplus_\rho c)$ is given by $Suc_{obs}(b \oplus_\rho c) = \{\, \{\!|\, \langle \rho \cdot b, \mathrm{E}\rangle \,|\!\} \,\}$. The (observable) successor set for any resolved statement consists of the singleton set containing only the (observable) probabilistic successor set of the statement.*

*For observable actions $b, b'$ in OAct and a synchronization action $c$ in Sync we have that the statement $(b \Box b') \oplus_\rho c$ has two nondeterministic alternatives: As $(b \Box b') \oplus_\rho c \overset{\nu}{\rightsquigarrow} b \oplus_\rho c$ and $(b \Box b') \oplus_\rho c \overset{\nu}{\rightsquigarrow} b' \oplus_\rho c$ give the two possible nondeterministic alternatives for the statement $(b \Box b') \oplus_\rho c$, the successor set of this statement is given by $Suc((b \Box b') \oplus_\rho c) = \{\, \{\!|\, \langle \rho \cdot b, \mathrm{E}\rangle, \langle (1-\rho) \cdot c, \mathrm{E}\rangle \,|\!\}, \{\!|\, \langle \rho \cdot b', \mathrm{E}\rangle, \langle (1-\rho) \cdot c, \mathrm{E}\rangle \,|\!\} \,\}$. The observable successor set $Suc_{obs}((b \Box b') \oplus_\rho c)$ is given by $Suc_{obs}((b \Box b') \oplus_\rho c) = \{\, \{\!|\, \langle \rho \cdot b, \mathrm{E}\rangle \,|\!\}, \{\!|\, \langle \rho \cdot b', \mathrm{E}\rangle \,|\!\} \,\}$.*

The following properties of the transition system can now easily be shown to hold by weight induction.

**Lemma 4.2.16**

*(a)* $s \overset{\nu}{\not\rightarrow} \mathrm{E}$.

*(b) If* $s \xrightarrow{\nu} s'$ *then* $wgt(s') < wgt(s)$.

*(c) No infinite sequence* $s \xrightarrow{\nu} s_1 \xrightarrow{\nu} s_2 \xrightarrow{\nu} \dots$ *exists.*

*(d)* $\mathcal{T}_{pnd}$ *is finitely branching, that is*

1. *$Suc(s)$ is a finite set for all $s$ and,*
2. *$Suc'(s)$ is a finite multiset for each resolved statement $s$.*

The first property gives that a nondeterministic alternative of a program is always some program and never a finished computation. The second property gives that a single nondeterministic alternative of a program is always seen as simpler than the whole program. The third property states that there is no *internal divergence* in the transition system. It is not possible to keep taking auxiliary $\nu$ steps. This property is a direct consequence of the second property. The final property, that the transition system is finitely branching, is important for the well-definedness of the operational semantics.

### 4.2.4 The operational semantics $\mathcal{O}$

The transition system contains information which is not considered to be observable behavior. For example the auxiliary $\nu$ steps do not correspond to actual observable behavior. To obtain the operational semantics this additional information is removed. The domain of all possible behaviors is denoted by $\mathbb{P}_o$. The elements of $\mathbb{P}_o$ are called processes.

**Definition 4.2.17** *The operational domain $\mathbb{P}_o$, ranged over by $p$, is given by*

$$\begin{array}{rcl} \mathbb{P}_o & = & \mathcal{P}_{nco}(\mathbb{Q}_o) \\ \mathbb{Q}_o & = & \mathit{Meas}(\mathbb{R}_o) \\ \mathbb{R}_o & = & \mathit{OAct}^\infty_\delta \end{array}$$

*where $\mathit{OAct}^\infty_\delta = \mathit{OAct}^\star \cup \mathit{OAct}^\star \cdot \{\, \delta \,\} \cup \mathit{OAct}^\omega$.*

A sequence $r$ in $\mathbb{R}_o$ corresponds to a possible run of the system. The sequence gives the observable actions produced by a single execution of the system. The run can terminate normally, $r \in \mathit{OAct}^\star$, deadlock after executing a number of actions, $r \in \mathit{OAct}^\star \cdot \{\, \delta \,\}$, or not terminate at all, $r \in \mathit{OAct}^\omega$. The elements of $\mathbb{Q}_o$ are called probabilistic subprocesses or simply subprocesses. A probabilistic subprocess in $\mathbb{Q}_o$ gives the probability for each observable event, i.e. for each Borel set of sequences in $\mathbb{R}_o$. A special case of an observable event is a set consisting of a single sequence. It is, therefore, possible to find the probability of executing a given sequence by looking at the probability of the observable event 'singleton the sequence'.

A process in $\mathbb{P}_o$ consists of a set of possible probabilistic subprocesses in $\mathbb{Q}_o$. Each probabilistic subprocess corresponds to a single nondeterministic alternative of the system.

The following example shows some processes in $\mathbb{P}_o$ and subprocesses in $\mathbb{Q}_o$. Recall the definition of the Dirac measure $\Delta_x$ from subsection 3.3.5 in the previous chapter: Given a sequence $w$ in $\mathbb{R}_o$ the measure $\Delta_w \in \mathit{Meas}(\mathbb{R}_o)$ is given by

$$\Delta_x(B) \quad = \quad \begin{cases} 1 & \text{if } x \in B \\ 0 & \text{otherwise} \end{cases}$$

where $B$ is any Borel set of sequences in $\mathbb{R}_o$. The measure $\Delta_w$ returns 1 for a Borel set $B$ exactly when the sequence $w$ is in $B$ and 0 otherwise. In other words $\Delta_w$ describes a situation where the sequence $w$ is obtained with probability 1.

**Example 4.2.18** *The sequences $b$, $bb'$, $b\delta$ and $b^\omega$ are all elements of $\mathbb{R}_o$. The first sequence corresponds to the execution of a single $b$ followed by normal termination. The second sequence corresponds to the execution of the action $b$ followed by the execution of the action $b'$ followed by normal termination. The sequence $b\delta$ corresponds to the execution of the action $b$ followed by deadlock of the system. The sequence $b^\omega$ corresponds to forever repeating the execution of the action $b$.*

*The measures $q_1 = \Delta_b$, $q_2 = \frac{1}{4}\Delta_{b\delta} + \frac{3}{4}\Delta_{b^\omega}$ are both elements of $\mathbb{Q}_o$. The measure $q_1$ corresponds to executing the sequence $b$ with probability 1. The measure $q_2$ corresponds to executing the sequence $b\delta$ with probability $\frac{1}{4}$ and the sequence $b^\omega$ with probability $\frac{3}{4}$.*

*The measure $q_3$ with $q_3(wOAct_\delta^\infty) = \frac{1}{2}^{\text{length of } w}$ for all $w \in \{ b, b' \}^*$, is also an element of $\mathbb{Q}$. The measure $q_3$ corresponds to the execution of an infinite sequence of actions where each action of the sequence is $b$ with probability $\frac{1}{2}$ or $b'$ also with probability $\frac{1}{2}$. The probability that the sequence which is chosen starts with e.g. $bb'b$, is $\frac{1}{8}$ (as $q_3(bb'bOAct_\delta^\infty) = \frac{1}{8}$). For each infinite sequence the probability that exactly this sequence is obtained is $0$.*

*No finite sequence is obtained as $q_3(Act^\star)$ is $0$: For every $n \geq 0$ there are $2^{n+1}$ different words $w$ of length $n+1$ consisting of actions $b$ and $b'$. For each of these words, the probability, $q_3(wOAct_\delta^\infty)$, of starting with this word is $2^{-(n+1)}$. Adding these probabilities gives a probability of $1$ for starting with some word of length $n+1$. This means that the probability of executing a word of length $n$ is $0$. By using countable additivity one obtains that the probability, $q_3(Act^\star)$, of executing any finite word is $0$*

*The sets $p_1 = \{ \Delta_b \}$, $p_2 = \{ q_2, q_3 \}$ and $p_3 = \{ \Delta_b, \Delta_{bb}, \Delta_{bbb}, \ldots, \Delta_{b^\omega} \}$ are elements of $\mathbb{P}_o$. The set $p_1$ gives only one possibility namely to behave like $\Delta_b$, i.e. to execute $b$ with probability one. The set $p_2$ offers the choice between acting like $q_2$ and acting like $q_3$. The set $p_3$ allows the execution of any positive number of actions $b$, including the possibility of executing infinitely many $b$ actions. Note that the set $\{ \Delta_b, \Delta_{bb}, \Delta_{bbb}, \ldots \}$ without the possibility for infinitely many actions $b$ is not in $\mathbb{P}_o$. The sequence $\Delta_b, \Delta_{bb}, \Delta_{bbb}, \ldots$ converges to $\Delta_{b^\omega}$ so for closedness of the set and thus for compactness of the set $\Delta_{b^\omega}$ must be included.*

The operational semantics is given as a model with values in the operational domain $\mathbb{P}_o$ introduced in definition 4.2.17. It is here where the distinction between the conditional and unconditional interpretation of probabilistic choice is made. The operational model for the unconditional interpretation of probabilistic choice is denoted by $\mathcal{O}_u$ and the model for the conditional interpretation of probabilistic choice is denoted by $\mathcal{O}_c$.

**Definition 4.2.19** *Let the index $i$ be $i = u$ for the unconditional interpretation of probabilistic choice or $i = c$ for the conditional interpretation of probabilistic choice. The operational models $\mathcal{O}_i : Res \xrightarrow{1} \mathbb{P}_o$ and the auxiliary functions $\widehat{\mathcal{O}_i} : \mathcal{MP}_f(POAct \times Res) \to \mathbb{P}_o$ are given by*

$$\begin{aligned}
\mathcal{O}_i(\mathrm{E}) &= \{ \Delta_\epsilon \} \\
\mathcal{O}_i(s) &= \bigcup \{ \widehat{\mathcal{O}_i}(M) \mid M \in Suc_{obs}(s) \} \\
\widehat{\mathcal{O}_i}(\emptyset) &= \{ \Delta_\delta \} \\
\widehat{\mathcal{O}_u}(\{\!|\, &\langle \rho_1 \cdot b_1, r_1 \rangle, \ldots, \langle \rho_n \cdot b_n, r_n \rangle \,|\!\}) \\
&= \{ \ (1-\pi)\Delta_\delta + \textstyle\sum_{j=1}^n \rho_j(\mu_j/b_j) \mid \mu_1 \in \mathcal{O}(r_1), \ldots, \mu_n \in \mathcal{O}(r_n) \ \} \\
\widehat{\mathcal{O}_c}(\{\!|\, &\langle \rho_1 \cdot b_1, r_1 \rangle, \ldots, \langle \rho_n \cdot b_n, r_n \rangle \,|\!\}) \\
&= \{ \ \textstyle\sum_{j=1}^n \frac{\rho_j}{\pi}(\mu_j/b_j) \mid \mu_1 \in \mathcal{O}(r_1), \ldots, \mu_n \in \mathcal{O}(r_n) \ \}
\end{aligned}$$

*where $n$ is greater or equal to one and $\pi$ equals $\sum_{j=1}^n \rho_j$.*

Recall the definition of the measure along prefix $b$, $\mu/b(B) = \mu(B/b)$ with $B/b = \{ w \mid bw \in B \}$. The operation $/b$ plays the role of prefixing on measures, see definition 3.3.18

and example 3.3.19. Also note that $\rho\mu$ denotes the measure $\mu$ multiplied with the scalar $\rho$, so for example, if $\mu(\{ a \}) = \frac{1}{2}$ then $\frac{1}{2}(\mu/b)$ gives $\frac{1}{4}$ for the set $\{ ba \}$.

The meaning or behavior of a statement consists of a set of possible probabilistic subprocesses. The resumption E denotes a finished computation. The only possible behavior for E is to produce no actions and terminate normally with probability one. The meaning of a statement $s$ is obtained by combining the meanings of all its nondeterministic alternatives. A nondeterministic alternative is described by a probabilistic successor set $M$ in $Suc_{obs}(s)$.

A possible probabilistic subprocess for the probabilistic successor set $\{\!|\, \langle \rho_1 \cdot b_1, r_1 \rangle, \ldots, \langle \rho_n \cdot b_n, r_n \rangle \,|\!\}$ is found by choosing, for each $j$, one probabilistic subprocess $\mu_j$ from $\mathcal{O}(r_j)$. These probabilistic subprocesses are "prefixed" with the corresponding action, giving $\mu_j/b_j$, multiplied with the corresponding probability $\rho_j$ and then added. For the unconditional interpretation of probabilistic choice, any missing probability $(1-\pi)$ is interpreted as a probability of deadlock. If the probabilistic choice is interpreted as being conditional, the numbers $\rho_1, \ldots, \rho_n$ are interpreted as relative frequencies and normalized to $\frac{\rho_1}{\pi}, \ldots, \frac{\rho_n}{\pi}$ to obtain actual probabilities. Deadlock is only introduced if there are no observable steps possible at all.

**Example 4.2.20** *Let $b_1$, $b_2$ and $b_3$ be actions in OAct and let $c$, $\bar{c}$ be complementary actions in Sync. The internal successor set of the statement $s = (b_1 \Box b_2) \oplus_\rho c$ is $\{ \{\!|\, \langle \rho \cdot b_1, \mathrm{E} \rangle \,|\!\}, \{\!|\, \langle \rho \cdot b_2, \mathrm{E} \rangle \,|\!\} \}$ so the operational meaning $\mathcal{O}_i(s)$ is given by $\mathcal{O}_i(s) = \widehat{\mathcal{O}_i}(\{\!|\, \langle \rho \cdot b_1, \mathrm{E} \rangle \,|\!\}) \cup \widehat{\mathcal{O}_i}(\{\!|\, \langle \rho \cdot b_2, \mathrm{E} \rangle \,|\!\})$. For the unconditional interpretation of probabilistic choice the first term is given by $\widehat{\mathcal{O}_u}(\{\!|\, \langle \rho{\cdot}b_1, \mathrm{E} \rangle \,|\!\}) = \{ (1-\rho)\Delta_\delta + \rho\mu/b_1 \mid \mu \in \mathcal{O}(\mathrm{E}) \} = \{ (1-\rho)\Delta_\delta + \rho\Delta_\epsilon/b_1 \} = \{ (1-\rho)\Delta_\delta + \rho\Delta_{b_1} \}$. Similarly we have $\widehat{\mathcal{O}_u}(\{\!|\, \langle \rho{\cdot}b_2, \mathrm{E} \rangle \,|\!\}) = \{ (1-\rho)\Delta_\delta + \rho\Delta_{b_2} \}$. The unconditional operational meaning of $s$ is the union of these two sets, $\mathcal{O}_u(s) = \{ (1-\rho)\Delta_\delta + \rho\Delta_{b_1}, (1-\rho)\Delta_\delta + \rho\Delta_{b_2} \}$.*

*Using the conditional interpretation of probabilistic choice gives $\widehat{\mathcal{O}_c}(\{\!|\, \langle \rho \cdot b_1, \mathrm{E} \rangle \,|\!\}) = \{ \frac{\rho}{\rho}\mu/b_1 \mid \mu \in \mathcal{O}(\mathrm{E}) \} = \{ \Delta_\epsilon/b_1 \} = \{ \Delta_{b_1} \}$ for the first term and similarly $\widehat{\mathcal{O}_c}(\{\!|\, \langle \rho \cdot b_2, \mathrm{E} \rangle \,|\!\}) = \{ \Delta_{b_2} \}$ for the second term. The conditional operational meaning of $s$ is the union of these two sets, $\mathcal{O}_c(s) = \{ \Delta_{b_1}, \Delta_{b_2} \}$.*

*Independent of the interpretation of the probabilistic choice, the operational meaning of the program $(b_1 \oplus_\rho b_2) \Box b_3$ is given by $\mathcal{O}_i((b_1 \oplus_\rho b_2) \Box b_3) = \{ \rho\Delta_{b_1} + (1-\rho)\Delta_{b_2}, \Delta_{b_3} \}$.*

*The operational meaning of the statement $(b_1 \oplus_\rho c) \| (b_2 \oplus_\sigma \bar{c})$ (see example 4.2.7 for the transitions of this statement) does depend on the interpretation of probabilistic choice. For the unconditional interpretation the operational meaning is given by $\mathcal{O}_u((b_1 \oplus_\rho c) \| (b_2 \oplus_\sigma \bar{c})) = \{ \rho\sigma\Delta_{b_1 b_2} + \rho(1-\sigma)\Delta_{b_1\delta} + (1-\rho)\Delta_\delta, \sigma\rho\Delta_{b_2 b_1} + \sigma(1-\rho)\Delta_{b_2\delta} + (1-\sigma)\Delta_\delta, (1-\rho)(1-\sigma)\Delta_\tau + (\rho + \sigma - \rho\sigma)\Delta_\delta \}$. For the conditional interpretation of probabilistic choice the operational meaning is given by $\mathcal{O}_c((b_1 \oplus_\rho c) \| (b_2 \oplus_\sigma \bar{c})) = \{ \Delta_{b_1 b_2}, \Delta_{b_2 b_1}, \Delta_\tau \}$*

The last example shows an important property of the modeling of the parallel composition. The parallel composition is modeled as a scheduler. Whether the next action is produced by the left component, by the right component or by synchronization between the components is decided by resolving the nondeterminism introduced by the operator $\|$. As resolving the nondeterminism has priority over the probabilistic choice, it cannot depend on the outcome of the probabilistic choice. A subprocess of the form $\rho\Delta_{b_1 b_2} + (1-\rho)\Delta_\tau$

which allows a separate step of the lefthand process for one probabilistic option but at the same time synchronization for the other probabilistic option, is therefore not present in $\mathcal{O}_c((b_1 \oplus_\rho c) \,\|\, (b_2 \oplus_\sigma \bar{c}))$. Such a subprocess would require that the outcome of resolving the parallel composition depends on the outcome of the probabilistic choice between $b_1$ and $c$ in the first component. Note that the probability in the second component does not play a role here, if for example the first component selects $b_1$ then the second component can no longer select $\bar{c}$ due to the conditionality of the probabilistic choice.
The definition of the operational model can be justified by showing that it is the unique fixed point of a higher-order operator.

**Lemma 4.2.21** *Let $i$ be $u$ for the unconditional or $c$ for the conditional interpretation of probabilistic choice. Let $\mathit{Sem} = \mathit{Res} \xrightarrow{1} \mathbb{P}_o$ and let $\Phi_i : \mathit{Sem} \to \mathit{Sem}$ be given by*

$$\begin{array}{rcl}
\Phi_i(S)(\mathrm{E}) & = & \{\, \Delta_\epsilon \,\} \\
\Phi_i(S)(s) & = & \bigcup \{\, \widehat{\Phi}_i(S)(M) \mid M \in \mathit{Suc}_{\mathit{obs}}(s) \,\} \\
\widehat{\Phi}_i(S)(\emptyset) & = & \{\, \Delta_\delta \,\} \\
\multicolumn{3}{l}{\widehat{\Phi}_u(S)(\{\!|\, \langle \rho_1 \cdot b_1, r_1 \rangle, \ldots, \langle \rho_n \cdot b_n, r_n \rangle \,|\!\})} \\
 & = & \{\ (1-\pi)\Delta_\delta + \sum_{i=1}^n \rho_i(\mu_i / b_i) \mid \mu_1 \in S(r_1), \ldots, \mu_n \in S(r_n)\ \} \\
\multicolumn{3}{l}{\widehat{\Phi}_c(S)(\{\!|\, \langle \rho_1 \cdot b_1, r_1 \rangle, \ldots, \langle \rho_n \cdot b_n, r_n \rangle \,|\!\})} \\
 & = & \{\ \sum_{i=1}^n \frac{\rho_i}{\pi}(\mu_i / b_i) \mid \mu_1 \in S(r_1), \ldots, \mu_n \in S(r_n)\ \}
\end{array}$$

*with $\pi = \sum_{i=1}^n \rho_i$, then $\Phi_i$ has a unique fixed point. Therefore, there is exactly one function $\mathcal{O}_i$ satisfying the equations in definition 4.2.19.*

**Proof** Clearly a function $\mathcal{O}_i$ satisfies the equations given in definition 4.2.19 exactly when it is a fixed point of $\Phi_i$ (for $i = u$ or $i = c$). It is sufficient to show that $\Phi_i$ is a well-defined contractive function. Using Banach's theorem this gives that $\Phi_i$ has a unique fixed point.

To show that $\Phi_i$ is well-defined requires showing that $\Phi_i(S) \in \mathit{Sem}$ holds for all $S$ in $\mathit{Sem}$. In other words one has to show that $\Phi_i(S)(r) \in \mathbb{P}_o$ for all $S$ in $\mathit{Sem}$ and $r$ in $\mathit{Res}$. As this is directly clear for $r = \mathrm{E}$ we only need to check the case that $r = s$ for some program $s$. The transition system is finitely branching so each probabilistic successor set is finite. For a finite successor set $M$, the elements of $\widehat{\Phi}_i(S)(M)$ are clearly probabilistic subprocesses in $\mathbb{Q}_o$. Straightforward calculation shows that the function which takes $(\mu_1, \ldots, \mu_n)$ to $\{\, (1-\pi)\Delta_\delta + \sum_{i=1}^n \rho_i(\mu_i / b_i) \,\}$ is nonexpansive and thus continuous. This gives that the set $\widehat{\Phi}_i(S)(M)$ is a compact set as it is a continuous image of the product of the compact sets $S(r_1)$ through $S(r_n)$ (lemma 2.1.13). As $\Phi_i(S)(s)$ is the union of $\widehat{\Phi}_i(S)(M)$ for finitely many multisets $M$, it is also a compact set of subprocesses. Nonemptiness of $\widehat{\Phi}_i(S)(s)$ is obvious.

To show that $\Phi_i$ is contractive it is sufficient to show that

$$d(\Phi_i(S)(r), \Phi_i(S')(r)) \;\leq\; \tfrac{1}{2} d(S, S')$$

holds for all resumptions $r$. Note that $d(\widehat{\Phi}_i(S)(\emptyset), \widehat{\Phi}_i(S')(\emptyset)) = d(\Delta_\delta, \Delta_\delta) = 0 \leq \frac{1}{2} d(S, S')$.

Also, if $\mu_i \in S(r_i)$ and $\mu'_i \in S'(r_i)$ for $i = 1, \ldots, n$ then

$$\begin{array}{ll} d((1-\pi)\Delta_\delta + \sum_{i=1}^n \rho_i(\mu_i/b_i), (1-\pi)\Delta_\delta + \sum_{i=1}^n \rho_i(\mu'_i/b_i)) & \\ \quad \leq \max\{\, d(\mu_i/b_i, \mu'_i/b_i) \mid i = 1, \ldots, n \,\} & \\ \quad \leq \max\{\, \tfrac{1}{2} d(\mu_i, \mu'_i) \mid i = 1, \ldots, n \,\} & \\ \quad \leq \max\{\, \tfrac{1}{2} d(S(r_i), S'(r_i)) \mid i = 1, \ldots, n \,\} & \\ \quad \leq \tfrac{1}{2} d(S, S') & \end{array}$$

so $d(\widehat{\Phi}_u(S)(M), d(\widehat{\Phi}_u(S')(M)) \leq \frac{1}{2} d(S,S')$ for all $M$. Similarly it follows that $d(\widehat{\Phi}_c(S)(M), \widehat{\Phi}_c(S')(M)) \leq \frac{1}{2} d(S,S')$ for all $M$. This gives

$$\begin{array}{ll} d(\Phi_i(S)(s), \Phi_i(S')(s)) & \\ \quad = & d(\bigcup\{\, \widehat{\Phi}_i(S)(M) \mid M \in Suc_{obs}(s) \,\}, \bigcup\{\, \widehat{\Phi}_i(S')(M) \mid M \in Suc_{obs}(s) \,\}) \\ \quad \leq & \max\{\, d(\widehat{\Phi}_i(S)(M), \widehat{\Phi}_i(S')(M)) \mid M \in Suc_{obs}(s) \,\} \\ \quad \leq & \tfrac{1}{2} d(S, S') \end{array}$$

$\Box$

**Example 4.2.22** *Take $S$ with $S(r) = \{\, \Delta_\epsilon \,\}$ for all $r$. The function $S$ gives direct termination ignoring the program completely. Applying $\Phi_i$ to $S$ gives $\Phi_i(S)(s) = \{\, \sum\{\, \rho\Delta_b \mid \langle \rho \cdot b, r \rangle \in M \,\} \mid M \in Suc_{obs}(s) \,\}$. The function $\Phi_i(S)$ gives the first step of a program followed by termination, ignoring the remaining part of the program. The function $\Phi_i^n(S)$ gives the first $n$ steps of a program.*

The operational semantics for $\mathcal{L}_{pnd}$ should be given for programs. The function $\mathcal{O}$, however, gives the meaning of resumptions. To remove this small discrepancy we define:

**Definition 4.2.23** *The operational semantics $\mathcal{O}_i[\![\bullet]\!] : \mathcal{L}_{pnd} \to \mathbb{P}_o$ for $\mathcal{L}_{pnd}$ is given by $\mathcal{O}_i[\![s]\!] = \mathcal{O}_i(s)$, where $i = u$ for the unconditional interpretation of probabilistic choice or $i = c$ for the conditional interpretation of probabilistic choice.*

By restricting to statements instead of resumptions the operational semantics $\mathcal{O}_i[\![\bullet]\!]$ is obtained from the operational model $\mathcal{O}_i$.

**Example 4.2.24** *In this example we again consider the '3 doors problem' introduced in example 4.2.3 and find the solution.*

*The programs $s_1 = (a; (b \Box c); a) \oplus_{\frac{1}{3}} ((b; c; b) \oplus_{\frac{1}{2}} (c; b; c))$ and $s_2 = (a; ((b; c) \Box (c; b)) \oplus_{\frac{1}{3}} ((b; c; a) \oplus_{\frac{1}{2}} (c; b; a))$ describe two strategies in the quiz-show as introduced in example 4.2.3. Using the transitions derived in in example 4.2.8 the operational meaning of these programs can be found. Note that here $a$, $b$ and $c$ are all observable actions in $OAct$. As these programs do not contain synchronization, the unconditional and conditional meanings are the same. Only the unconditional meaning is given. It turns out that to find the meaning of the programs $s_1$ and $s_2$, the meanings of the sub programs $(b \Box c); a$ and $(b; c) \Box (c; b)$ are needed. The observable successor sets for these statements are*

$$\begin{array}{rcl} Suc_{obs}((b \Box c); a) & = & \{\, \{\!|\, \langle 1 \cdot b, a \rangle \,|\!\}, \{\!|\, \langle 1 \cdot c, a \rangle \,|\!\} \,\} \\ Suc_{obs}((b; c) \Box (c; b)) & = & \{\, \{\!|\, \langle 1 \cdot b, c \rangle \,|\!\}, \{\!|\, \langle 1 \cdot c, b \rangle \,|\!\} \,\} \end{array}$$

*this gives*

$$\begin{aligned}
\mathcal{O}_u((b \,\Box\, c); a) &= \widehat{\mathcal{O}_u}(\{\!|\, \langle 1 \cdot b, a\rangle \,|\!\}) \cup \widehat{\mathcal{O}_u}(\{\!|\, \langle 1 \cdot c, a\rangle \,|\!\}) \\
&= \{\, \mu/b \mid \mu \in \mathcal{O}_u(a) \,\} \cup \{\, \mu/c \mid \mu \in \mathcal{O}_u(a) \,\} \\
&= \{\, \Delta_a/b \,\} \cup \{\, \Delta_a/c \,\} \\
&= \{\, \Delta_{ba}, \Delta_{ca} \,\} \\
\mathcal{O}_u((b; c) \,\Box\, (c; b)) &= \widehat{\mathcal{O}_u}(\{\!|\, \langle 1 \cdot b, c\rangle \,|\!\}) \cup \widehat{\mathcal{O}_u}(\{\!|\, \langle 1 \cdot c, b\rangle \,|\!\}) \\
&= \{\, \Delta_{bc}, \Delta_{cb} \,\}
\end{aligned}$$

*The processes $\mathcal{O}_u(s_1)$ and $\mathcal{O}_u(s_2)$ can now be found as follows: The observable successor sets for $s_1$ and $s_2$ are*

$$\begin{aligned}
Suc_{obs}(s_1) &= \{\, \{\!|\, \langle \tfrac{1}{3} \cdot a, (b \,\Box\, c); a\rangle, \langle \tfrac{1}{3} \cdot b, c; b\rangle, \langle \tfrac{1}{3} \cdot c, b; c\rangle \,|\!\} \,\} \\
Suc_{obs}(s_2) &= \{\, \{\!|\, \langle \tfrac{1}{3} \cdot a, (b; c) \,\Box\, (c; b)\rangle, \langle \tfrac{1}{3} \cdot b, c; a\rangle, \langle \tfrac{1}{3} \cdot c, b; a\rangle \,|\!\} \,\}
\end{aligned}$$

*which gives*

$$\begin{aligned}
\mathcal{O}_u(s_1) &= \widehat{\mathcal{O}_i}(\{\!|\, \langle \tfrac{1}{3} \cdot a, (b \,\Box\, c); a\rangle, \langle \tfrac{1}{3} \cdot b, c; b\rangle, \langle \tfrac{1}{3} \cdot c, b; c\rangle \,|\!\}) \\
&= \{\, \tfrac{1}{3}(\mu/a) + \tfrac{1}{3}\Delta_{bcb} + \tfrac{1}{3}\Delta_{cbc} \mid \mu \in \mathcal{O}_u((b \,\Box\, c); a) \,\} \\
&= \{\, \tfrac{1}{3}\Delta_{aba} + \tfrac{1}{3}\Delta_{bcb} + \tfrac{1}{3}\Delta_{cbc}, \tfrac{1}{3}\Delta_{aca} + \tfrac{1}{3}\Delta_{bcb} + \tfrac{1}{3}\Delta_{cbc} \,\} \\
\mathcal{O}_u(s_2) &= \widehat{\mathcal{O}_i}(\{\!|\, \langle \tfrac{1}{3} \cdot a, (b; c) \,\Box\, (c; b)\rangle, \langle \tfrac{1}{3} \cdot b, c; a\rangle, \langle \tfrac{1}{3} \cdot c, b; a\rangle \,|\!\}) \\
&= \{\, \tfrac{1}{3}(\mu/a) + \tfrac{1}{3}\Delta_{bca} + \tfrac{1}{3}\Delta_{cba} \mid \mu \in \mathcal{O}_u((b; c) \,\Box\, (c; b)) \,\} \\
&= \{\, \tfrac{1}{3}\Delta_{abc} + \tfrac{1}{3}\Delta_{bca} + \tfrac{1}{3}\Delta_{cba}, \tfrac{1}{3}\Delta_{acb} + \tfrac{1}{3}\Delta_{bca} + \tfrac{1}{3}\Delta_{cba} \,\}
\end{aligned}$$

*The semantics $\mathcal{O}_u[\![s_j]\!]$ of $s_j$ coincides with $\mathcal{O}_u(s_j)$ (for $j = 1, 2$).*

*Having found the semantics of both programs, the next step is to give an observable event which describes 'winning the prize'. The contestant wins the prize if the door with the prize, i.e. door a, is selected at the end of the program. This means that the event we are looking for is $Act^\star a$, the set of all finite words ending in a. Observable events are Borel sets of action sequences. The set $Act^\star a$ is open and therefore indeed an observable event. (All words in $Act^\star a$ are finite and a sufficiently small open ball around a finite word w contains only the word w itself.)*

*For both strategies we can now give which probabilities are possible for winning the prize (in general the probability of winning the prize may depend on the actions of the quiz-master, i.e. on the nondeterminism). For the strategy described by $s_1$ we get:*

$$\begin{aligned}
\mathcal{O}_u(s_1)(Act^\star a) &= \{\, \tfrac{1}{3}\Delta_{aba}(Act^\star a) + \tfrac{1}{3}\Delta_{bcb}(Act^\star a) + \tfrac{1}{3}\Delta_{cbc}(Act^\star a)\,, \\
&\qquad \tfrac{1}{3}\Delta_{aca}(Act^\star a) + \tfrac{1}{3}\Delta_{bcb}(Act^\star a) + \tfrac{1}{3}\Delta_{cbc}(Act^\star a) \,\} \\
&= \{\, \tfrac{1}{3} + 0 + 0\,,\ \tfrac{1}{3} + 0 + 0 \,\} \\
&= \{\, \tfrac{1}{3} \,\}
\end{aligned}$$

*and for the strategy described by $s_2$ we get:*

$$\begin{aligned}
\mathcal{O}_u(s_2)(Act^\star\, a) &= \{\, \tfrac{1}{3}\Delta_{abc}(Act^\star\, a) + \tfrac{1}{3}\Delta_{bca}(Act^\star\, a) + \tfrac{1}{3}\Delta_{cba}(Act^\star\, a)\,, \\
&\quad\ \tfrac{1}{3}\Delta_{acb}(Act^\star\, a) + \tfrac{1}{3}\Delta_{bca}(Act^\star\, a) + \tfrac{1}{3}\Delta_{cba}(Act^\star\, a)\,\} \\
&= \{\, 0 + \tfrac{1}{3} + \tfrac{1}{3}\,,\, 0 + \tfrac{1}{3} + \tfrac{1}{3}\,\} \\
&= \{\, \tfrac{2}{3}\,\}
\end{aligned}$$

*Staying with the first selected door gives a probability of $\frac{1}{3}$ of winning the prize. Switching doors, however, gives a probability of $\frac{2}{3}$ of winning the prize.*

This solves the three doors problem:
Switching doors is the better strategy.

*At first sight people often think that the probability of winning is $\frac{1}{2}$ as in the end there is a choice between two doors, one of which must contain the prize. This reasoning actually corresponds to a different strategy in which the contestant simply forgets what happened before and randomly selects one of the remaining closed doors. This strategy can be described by the program $s_3$ given by $s_3 = (a;(b;(a \oplus_{\frac{1}{2}} c) \Box c;(a \oplus_{\frac{1}{2}} b))) \oplus_{\frac{1}{3}} ((b;c;(b \oplus_{\frac{1}{2}} a)) \oplus_{\frac{1}{2}} (c;b;(c \oplus_{\frac{1}{2}} a)))$. That the probability of winning with this strategy is indeed $\frac{1}{2}$ can be checked by looking at $\mathcal{O}_u(s_3)(Act^\star\, a)$. As we have seen, however, the strategy to always switch doors, as described by $s_2$, gives a higher probability of winning.*

### 4.2.5 Denotational semantics

When checking properties of a statement one wants to be able to decompose the statement in parts and work with the parts of the statement. As in chapter 3, section 3.4 the compositionality principle is used to make this possible; the meaning of a statement is described based on the meaning of the parts of a statement.

The denotational semantics, introduced below, gives the meaning of statements in a compositional way. The meaning of a statement is given as an element of the domain of denotational meanings, denoted by $\mathbb{P}_d$, and several semantical operations are introduced to compose these meanings.

The operational semantics is not compositional. The statements $c_1$ and $c_2$ have the same operational behavior, but within the context $\bullet \parallel \bar{c}_1$ they behave differently. (Here $\bullet$ indicates the place in the context where a statement can be substituted.) This means that the operational behavior does not contain sufficient information to be able to compose meanings. For the denotational model extra information about a statement is maintained. In the denotational meaning, unmatched synchronization actions are visible and do not result in deadlock. Also, the moments of choice are remembered in the denotational meaning. This branching information is not essential for the unconditional interpretation of probabilistic choice, only for the conditional interpretation of choice.

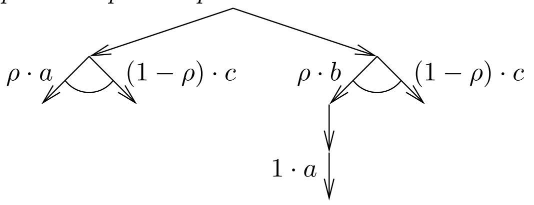
**Definition 4.2.25** *The denotational domain $\mathbb{P}_d$ is given by the following domain equations:*

$$\mathbb{P}_d \quad \simeq \quad \mathcal{P}_{nco}(\mathbb{Q}_d)$$

$$\begin{aligned}\mathbb{Q}_d &\simeq \mathcal{MP}_f(\mathbb{R}_d)\\ \mathbb{R}_d &\simeq PAct + PAct \times id_{\frac{1}{2}}(\mathbb{P}_d)\end{aligned}$$

The elements of $\mathbb{P}_d$ are called denotational processes, or simply processes if no confusion is possible. The elements of $\mathbb{Q}_d$ are called probabilistic subprocesses, or just subprocesses. A process $p$ first nondeterministically chooses a probabilistic subprocess $q$. The subprocess $q$ probabilistically chooses an action to execute. The action is possibly followed by another process. To have a single notation for all elements of $\mathbb{R}_d$ a element $\alpha$ of *PAct* is identified with $\langle \alpha, p_\epsilon \rangle$ and the (meta-)variables $\hat{p}$ and $\hat{p}'$ are used to range over $\mathbb{P}_d + \{ p_\epsilon \}$. Using this convention, $\langle \alpha, \hat{p} \rangle$ ranges over $\mathbb{R}_d$. Having a single notation for elements of $\mathbb{R}_d$ avoids the need for excessive case-distinctions in the definitions and proofs below.

**Example 4.2.26** *The process $p$ given by $p = \{ \{\!| \rho{\cdot}a, (1-\rho){\cdot}c |\!\}, \{\!| \langle \rho{\cdot}b, \{ \{\!| 1{\cdot}a |\!\} \} \rangle, (1-\rho){\cdot} c |\!\} \}$ contains two subprocesses. The first subprocess gives $a$ with probability $\rho$ and $c$ with probability $1-\rho$. The second process gives $c$ with probability $1-\rho$ and $b$ followed by another process with probability $\rho$. The process following $b$ only has one option, $a$ with probability 1. The following tree depicts the process $p$.*



*Note the similarity between the tree representation of a denotational process and the abstract transition trees as e.g. in example 4.2.7.*

For each of the syntactical operators, a semantical operation on processes is defined. The semantical operation specifies how meanings are combined to obtain the meaning of a statement built with the corresponding syntactical operator.

**Definition 4.2.27** *All denotational operations are elements of $Op = \mathbb{P}_d \times \mathbb{P}_d \xrightarrow{1} \mathbb{P}_d$, i.e. they are nonexpansive functions that take a pair of processes and yield a single process.*

*(a) The operation $\square \in Op$ is defined by*

$$p_1 \square p_2 \quad = \quad p_1 \cup p_2$$

*(b) The operation $\oplus_\rho \in Op$ is defined by*

$$\begin{aligned}p_1 \oplus_\rho p_2 &= \{ q_1 \oplus'_\rho q_2 \mid q_1 \in p_1, q_2 \in p_2 \}\\ q_1 \oplus'_\rho q_2 &= \rho q_1 \sqcup (1-\rho) q_2\\ \rho q &= \{\!| \langle \rho\sigma \cdot a, \hat{p} \rangle \mid \langle \sigma \cdot a, \hat{p} \rangle \in q |\!\}\end{aligned}$$

*(c) The operation $;\in Op$ is defined by*

$$\begin{aligned}p_1 \,; p_2 &= \{ q_1 \,;' p_2 \mid q_1 \in p_1 \}\\ q \,;' p &= \{\!| \langle \alpha, \hat{p} \,; p \rangle \mid \langle \alpha, \hat{p} \rangle \in q |\!\}\end{aligned}$$

*where* $\hat{p}\,; p = p$ *if* $\hat{p} = p_\epsilon$.

(d) *The operation* $\|\in Op$ *is defined by*

$$\begin{aligned}
p_1 \| p_2 &= p_1 \mathbin{|\!\lfloor} p_2 \cup p_1 \mathbin{|\!\lfloor} p_2 \cup p_1 | p_2 \\
p_1 \mathbin{|\!\lfloor} p_2 &= \{\, q_1 \mathbin{|\!\lfloor}' p_2 \mid q_1 \in p_1 \,\} \\
q \mathbin{|\!\lfloor}' p &= \{\!\!\{\, \langle \alpha, \hat{p} \| p \rangle \mid \langle \alpha, \hat{p} \rangle \in q \,\}\!\!\} \\
p_1 | p_2 &= \{\, q_1 |' q_2 \mid q_1 \in p_1, q_2 \in p_2 \,\} \\
q_1 |' q_2 &= \{\!\!\{\, \langle \rho\sigma \cdot \tau, \hat{p} \| \hat{p}' \rangle \mid \langle \rho \cdot c, \hat{p} \rangle \in q_1, \langle \sigma \cdot \bar{c}, \hat{p}' \rangle \in q_2 \,\}\!\!\}
\end{aligned}$$

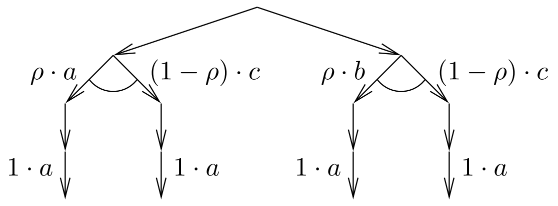
*where* $\hat{p} \| \hat{p}' = \hat{p}' \| \hat{p} = \hat{p}'$ *if* $\hat{p} = p_\epsilon$.

The definitions of several of the operations are recursive but can be shown to be correct using the metric machinery. Each operation is shown to be the unique fixed point of a contractive higher order operation $\Omega\colon Op \to Op$. The proof that the higher order operation $\Omega$ is indeed contractive is omitted as it is a straightforward extension of results in chapter 3, e.g. lemma 3.4.10, and known results (see e.g. [38]).

Note that $q_1 | q_2$ may have a total probability of less than one. It may even be the empty multiset. As in the transition system, the interpretation of this 'missing' probability depends on the interpretation of the probabilistic choice. For the unconditional interpretation of probabilistic choice, the missing probability is a probability of deadlock. The subprocess $\{\!\!\{\, \frac{2}{3} \cdot a \,\}\!\!\}$ describes a situation in which with probability $\frac{2}{3}$ an action $a$ will be taken and with probability $\frac{1}{3}$ deadlock will occur. For the conditional interpretation of probabilistic choice, the real numbers assigned to actions are seen as relative frequencies of these actions. To get probabilities the numbers have to be normalized to sum up to one. The only subprocess in which deadlock is possible is the empty multiset. The subprocesses $\{\!\!\{\, \frac{1}{4} \cdot a, \frac{1}{4} \cdot b \,\}\!\!\}$ and $\{\!\!\{\, \frac{1}{2} \cdot a, \frac{1}{2} \cdot b \,\}\!\!\}$ describes the same situation. In this situation $a$ and $b$ are equally likely to occur, i.e. both occur with probability $\frac{1}{2}$, and no deadlock is possible.

**Example 4.2.28** *Let* $p_a = \{\, \{\!\!\{\, 1 \cdot a \,\}\!\!\} \,\}$, $p_b = \{\, \{\!\!\{\, 1 \cdot b \,\}\!\!\} \,\}$ *and* $p_c = \{\, \{\!\!\{\, 1 \cdot c \,\}\!\!\} \,\}$. *Then* $p_b; p_a = \{\, \{\!\!\{\, \langle 1 \cdot b, p_a \rangle \,\}\!\!\} \,\}$, *and* $p_a \Box p_b; p_a = \{\, p_a, \{\!\!\{\, \langle 1 \cdot b, p_a \rangle \,\}\!\!\} \,\}$, *and* $(p_a \Box p_b; p_a) \oplus_\rho p_c = \{\, \{\!\!\{\, \rho \cdot a, (1-\rho) \cdot c \,\}\!\!\}, \{\!\!\{\, \langle \rho \cdot b, p_a \rangle, (1-\rho) \cdot c \,\}\!\!\} \,\}$. *See example 4.2.26 for a tree representation of this process.*

*The following tree depicts the process given by* $((p_a \Box p_b) \oplus_\rho p_c); p_a = \{\, \{\!\!\{\, \langle \rho \cdot a, p_a \rangle, \langle (1-\rho) \cdot c, p_a \rangle \,\}\!\!\}, \{\!\!\{\, \langle \rho \cdot b, p_a \rangle, \langle (1-\rho) \cdot c, p_a \rangle \,\}\!\!\} \,\}$.



In a denotational process it is still possible to interpret the probability as conditional or unconditional. Due to this, no distinction needs to be made between the conditional and unconditional interpretation of probabilistic choice when giving the denotational meaning of a statement. A single denotational model can be used for both interpretations.

The operations introduced above are nonexpansive by definition. For the operation '$;$' this can be strengthened. The operation '$;$' is nonexpansive in its first component and contractive in its second, i.e.

$$d(p_1\,;p_2, p_1'\,;p_2') \leq \max\{\, d(p_1,p_1'), \tfrac{1}{2}d(p_2,p_2')\,\}$$

The proof of this fact is again a straightforward extension of results in chapter 3 (lemma 3.4.11) and known results. To justify the following definition, $\mathcal{D}$ can be defined as the fixed point of a higher-order mapping. Contractiveness of $;$ in its second argument is required for contractiveness of this higher-order mapping.

**Definition 4.2.29** *The denotational model $\mathcal{D}\colon \mathcal{L}_{pnd} \to \mathbb{P}_d$ is given by*

$$\begin{array}{rcl} \mathcal{D}(a) & = & \{\, \{\!|\, 1\cdot a \,|\!\}\,\} \\ \mathcal{D}(x) & = & \mathcal{D}(D(x)) \\ \mathcal{D}(s_1 \;op\; s_2) & = & \mathcal{D}(s_1) \;op\; \mathcal{D}(s_2) \end{array}$$

*where op is $\Box$, $\oplus_\rho$, $;$ or $\|$.*

A single action $a$ acts like $a$ with probability one. Recursion is handled by body replacement and the semantical operation *op* is used to give the meaning of any statement built using the syntactic operator *op*.

**Example 4.2.30** *The denotational meanings of a, b and c are $\{\,\{\!|\,1\cdot a\,|\!\}\,\}$, $\{\,\{\!|\,1\cdot b\,|\!\}\,\}$ and $\{\,\{\!|\,1\cdot c\,|\!\}\,\}$ respectively, so using example 4.2.28,*

$$\begin{array}{rcl} \mathcal{D}((a \Box b; a) \oplus_\rho c) & = & \{\, \{\!|\, \rho\cdot a, (1{-}\rho)\cdot c \,|\!\}, \{\!|\, \langle \rho\cdot b, \{\,\{\!|\,1\cdot a\,|\!\}\,\}\rangle, (1{-}\rho)\cdot c \,|\!\}\,\} \\ \mathcal{D}(((a \Box b) \oplus_\rho c); a) & = & \mathcal{D}((a \Box b; a) \oplus_\rho c); \{\,\{\!|\,1\cdot a\,|\!\}\,\} \\ & = & \{\, \{\!|\, \langle \rho\cdot a, \{\,\{\!|\,1\cdot a\,|\!\}\,\}\rangle, \langle (1-\rho)\cdot c, \{\,\{\!|\,1\cdot a\,|\!\}\,\}\rangle \,|\!\}, \\ & & \;\; \{\!|\, \langle \rho\cdot b, \{\,\{\!|\,1\cdot a\,|\!\}\,\}\rangle, \langle (1-\rho)\cdot c, \{\,\{\!|\,1\cdot a\,|\!\}\,\}\rangle \,|\!\}\,\} \end{array}$$

*See example 4.2.28 for a tree representation of the process $\mathcal{D}(((a \Box b) \oplus_\rho c); a)$.*

The denotational model $\mathcal{D}$ already has the correct form. We define the denotational semantics $\mathcal{D}[\![\bullet]\!]$, which coincides with the model $\mathcal{D}$, only to maintain symmetry with the definition of the operational semantics.

**Definition 4.2.31** *The denotational semantics $\mathcal{D}[\![\bullet]\!]\colon \mathcal{L}_{pnd} \to \mathbb{P}_d$ is given by*

$$\mathcal{D}[\![s]\!] \;=\; \mathcal{D}(s)$$

Having given two operational models and a denotational model, a natural question to ask is how the models are related. In the following subsection, the operational models are compared with the denotational model.

## 4.2.6 Comparing the operational and denotational semantics

The denotational semantics contains more information than the operational semantics. As described in the previous subsection this is necessary to achieve compositionality. The branching structure and unmatched synchronization actions are still present in the denotational processes. Recall that a model based on the transition system but using a domain other than the operational domain is called an operational-like model. To compare the denotational and operational semantics a series of operational-like intermediate models are introduced. Step by step the extra information from the denotational meaning is removed by abstraction functions. For processes in the denotational domain, the probabilistic choice can still be interpreted as an unconditional or a conditional choice. The choice for unconditional or conditional probabilistic choice is already expressed in processes in the operational domain. Different abstraction functions are needed for the unconditional interpretation of probabilistic choice and for the conditional interpretation of probabilistic choice. The following graph shows the steps involved in the comparison of the operational and denotational semantics.

$$\begin{tikzcd}
\mathbb{P}_d + \{\, p_\epsilon \,\} & \mathbb{P}_b & \mathbb{P}_m & \mathbb{P}_o \\
 & & & \mathcal{O}_u \\
\mathcal{D} = \mathcal{O}^* \arrow[r, "abs_1"] & \mathcal{O}^b \arrow[r, "abs_2"] & \mathcal{O}^m \arrow[ur, "abs_{3u}"] \arrow[dr, "abs_{3c}"'] & \\
 & & & \mathcal{O}_c
\end{tikzcd}$$

The empty process $p_\epsilon$ is added to the denotational domain and the denotational model $\mathcal{D}$ is extended from a function from statements to $\mathbb{P}_d$ to a function, from resumptions to $\mathbb{P}_d + \{\, p_\epsilon \,\}$. The extended function is also called $\mathcal{D}$. An operational-like model $\mathcal{O}^*$ is defined on the denotational domain and shown to coincide with the extended denotational model. A branching domain $\mathbb{P}_b$ is introduced in which the unmatched synchronization actions are not present in the processes. An abstraction from the denotational domain to a branching domain $\mathbb{P}_b$ removes unmatched synchronization actions. An operational-like model $\mathcal{O}^b$ on the branching domain is given and shown to be an abstraction of $\mathcal{O}^*$.

Next an intermediate domain $\mathbb{P}_m$ is given. A process in the intermediate domain contains information about the probabilistic branching but does not contain nondeterministic branching. A second abstraction function removes the nondeterministic branching from a process in the branching domain $\mathbb{P}_b$ to obtain a process in the intermediate domain $\mathbb{P}_m$. An operational-like model $\mathcal{O}^m$, given on the intermediate domain, is shown to be an abstraction of $\mathcal{O}^b$.

Finally the operational models $\mathcal{O}_u$ and $\mathcal{O}_c$, which use the linear operational domain $\mathbb{P}_o$, are shown to be abstractions of the intermediate model $\mathcal{O}^m$. It is here that the difference between the unconditional and conditional interpretation of probabilistic choice results in two different abstraction functions. These results are combined to give that both operational semantics are abstractions of the denotational semantics.

The first step is to extend the denotational model to resumptions. To be able to assign a meaning to the empty resumption E, the empty meaning $p_\epsilon$ is added to the domain. The extended denotational model is also denoted by $\mathcal{D}$.

**Definition 4.2.32** *The model $\mathcal{D} : Res \to (\mathbb{P}_d + \{\, p_\epsilon \,\})$ is given by*

$$\begin{array}{rcl} \mathcal{D}(\mathrm{E}) &=& p_\epsilon \\ \mathcal{D}(a) &=& \{\, \{\!\!\{\, 1 \cdot a \,\}\!\!\}\, \} \\ \mathcal{D}(x) &=& \mathcal{D}(D(x)) \\ \mathcal{D}(s \mathbin{op} s') &=& \mathcal{D}(s) \mathbin{op} \mathcal{D}(s') \end{array}$$

*where op is* $;$ , $\oplus_\rho$ ,$\square$, $\|$, $\lfloor\!\lfloor$ *or* $|$ *and* $s, s' \in Stat^+$.

For well-definedness one should check that $\mathcal{D}(s)$ is in $\mathbb{P}_d$ for all statements $s$, as $op$ is only defined on $\mathbb{P}_d$. It is clear that the model $\mathcal{D}$ on resumptions is a conservative extension of the denotational model $\mathcal{D}$ on statements and that indeed $\mathcal{D}(s) \in \mathbb{P}_d$ holds.

The operational-like model $\mathcal{O}^*$, introduced below, also uses the domain $\mathbb{P}_d + \{\, p_\epsilon \,\}$.

**Definition 4.2.33** *Put $Sem = Res \to (\mathbb{P}_d + \{\, p_\epsilon \,\})$ and let $S$ range over $Sem$. The higher order function $\Phi^* : Sem \to Sem$ and auxiliary function $\widehat{\Phi}^*(S) : \mathcal{MP}_f(POAct \times Res) \to \mathbb{Q}_d$ are given by*

$$\begin{array}{rcl} \Phi^*(S)(\mathrm{E}) &=& p_\epsilon \\ \Phi^*(S)(s) &=& \{\, \widehat{\Phi}^*(S)(M) \mid M \in Suc(s) \,\} \\ \widehat{\Phi}^*(S)(M) &=& \{\!\!\{\, \langle \alpha, S(r) \rangle \mid \langle \alpha, r \rangle \in M \,\}\!\!\} \end{array}$$

*The operational-like model $\mathcal{O}^*$ is the unique fixed point of $\Phi^*$.*

Recall that $\langle \alpha, p_\epsilon \rangle$ is identified with $\alpha$. That $\Phi^*$ is contractive and thus has a unique fixed point is not difficult to check. The operational-like model $\mathcal{O}^*$ satisfies the following equations:

$$\begin{array}{rcl} \mathcal{O}^*(\mathrm{E}) &=& p_\epsilon \\ \mathcal{O}^*(s) &=& \{\, \widehat{\mathcal{O}}^*(M) \mid M \in Suc(s) \,\} \\ \widehat{\mathcal{O}}^*(M) &=& \{\!\!\{\, \langle \alpha, \mathcal{O}^*(r) \rangle \mid \langle \alpha, r \rangle \in M \,\}\!\!\} \end{array}$$

The definition of $\mathcal{O}^*$ as the fixed point of the higher-order operator $\Phi^*$ is exploited in the proof of the next lemma which establishes the equality of $\mathcal{O}^*$ and $\mathcal{D}$.

**Lemma 4.2.34** *The extended denotational model $\mathcal{D}$ is a fixed point of $\Phi^*$, i.e. $\Phi^*(\mathcal{D})(r) = \mathcal{D}(r)$ for all resumptions $r$. As $\mathcal{O}^*$ is the unique fixed point of $\Phi^*$ this implies that $\mathcal{O}^* = \mathcal{D}$.*

**Proof** That $\Phi^*(\mathcal{D})(r) = \mathcal{D}(r)$ can be checked by induction on the weight of the resumption $r$. A few typical cases are given below.

- As $s \,\square\, s' \overset{\nu}{\leadsto} s_0$ exactly when $s \overset{\nu}{\leadsto} s_0$ or $s' \overset{\nu}{\leadsto} s_0$ we have

$$\begin{array}{rcl} \Phi^*(\mathcal{D})(s \,\square\, s') &=& \{\, \widehat{\Phi}^*(\mathcal{D})(M) \mid M \in Suc(s \,\square\, s') \,\} \\ &=& \{\, \widehat{\Phi}^*(\mathcal{D})(M) \mid M \in Suc(s) \cup Suc(s') \,\} \\ &=& \Phi^*(\mathcal{D})(s) \cup \Phi^*(\mathcal{D})(s') \\ \text{[ind. hyp.]} &=& \mathcal{D}(s) \cup \mathcal{D}(s') \\ &=& \mathcal{D}(s \,\square\, s') \end{array}$$

Note that both $s$ and $s'$ have a lower weight that $s \,\Box\, s'$ allowing use of the induction assumption.

- If $s \xrightarrow{\nu} s_i$ for $i = 1, \ldots, n$, are all the possible steps for $s$ then $\mathcal{D}(s \oplus_\rho s') = \cup_{i=1}^{n} \mathcal{D}(s_i \oplus_\rho s')$. Using this the case for $s \oplus_\rho s'$ with $s$ not resolved is similar to the case for $s \,\Box\, s'$ above. If $s'$ is not resolved the situation is symmetrical.

  If both $s$ and $s'$ are resolved then we have that

$$Suc(s \oplus_\rho s') \quad = \quad \{\, \rho Suc'(s) \sqcup (1-\rho) Suc'(s') \,\}$$

  holds, so

$$\begin{array}{rcl}
\Phi^*(\mathcal{D})(s \oplus_\rho s') & = & \{\, \widehat{\Phi}^*(\mathcal{D})(M) \mid M \in Suc(s \oplus_\rho s') \,\} \\
& = & \{\, \widehat{\Phi}^*(\mathcal{D})(\rho Suc'(s) \sqcup (1-\rho) Suc'(s')) \,\} \\
& = & \{\, \widehat{\Phi}^*(\mathcal{D})(\rho Suc'(s)) \sqcup \widehat{\Phi}^*(\mathcal{D})((1-\rho) Suc'(s')) \,\} \\
& = & \{\, \{\!\mid \langle \alpha, \mathcal{D}(r) \rangle \mid \langle \alpha, r \rangle \in (\rho Suc'(s) \sqcup (1-\rho) Suc'(s')) \mid\!\} \,\} \\
& = & \{\ \rho \{\!\mid \langle \alpha, \mathcal{D}(r) \rangle \mid \langle \alpha, r \rangle \in Suc'(s) \mid\!\} \sqcup \\
& & \quad (1-\rho) \{\!\mid \langle \alpha, \mathcal{D}(r) \rangle \mid \langle \alpha, r \rangle \in Suc'(s')) \mid\!\}\ \} \\
& = & \{\, \widehat{\Phi}^*(\mathcal{D})(Suc'(s)) \oplus'_\rho \widehat{\Phi}^*(\mathcal{D})(Suc'(s')) \,\} \\
& = & \{\, \widehat{\Phi}^*(\mathcal{D})(Suc'(s)) \,\} \oplus_\rho \{\, \widehat{\Phi}^*(\mathcal{D})(Suc'(s')) \,\} \\
& = & \Phi^*(\mathcal{D})(s) \oplus_\rho \Phi^*(\mathcal{D})(s') \\
\text{[ind. hyp.]} & = & \mathcal{D}(s) \oplus_\rho \mathcal{D}(s') \\
& = & \mathcal{D}(s \oplus_\rho s')
\end{array}$$

- If $s \xrightarrow{\nu} r$ then $s; s' \xrightarrow{\nu} r; s'$ and if $s; s' \xrightarrow{\nu} r'$ then $r' = r; s'$ and $s \xrightarrow{\nu} r$ which gives that

$$Suc(s; s') = \{\, Suc'(s_0; s') \mid s \overset{\nu}{\rightsquigarrow} s_0 \,\}$$

  If $s_0$ is a resolved statement then

$$\begin{array}{rcl}
\widehat{\Phi}^*(\mathcal{D})(\{\!\mid \langle \alpha, r \rangle \mid s_0; s' \xrightarrow{\alpha} r \mid\!\}) & = & \{\!\mid \langle \alpha, \mathcal{D}(r) \rangle \mid s_0; s' \xrightarrow{\alpha} r \mid\!\} \\
& = & \{\!\mid \langle \alpha, \mathcal{D}(r'; s') \rangle \mid s_0 \xrightarrow{\alpha} r' \mid\!\} \\
& = & \{\!\mid \langle \alpha, \mathcal{D}(r'); \mathcal{D}(s') \rangle \mid s_0 \xrightarrow{\alpha} r' \mid\!\} \\
& = & \{\!\mid \langle \alpha, \mathcal{D}(r') \rangle \mid s_0 \xrightarrow{\alpha} r' \mid\!\} ;' \mathcal{D}(s')
\end{array}$$

  In other words $\widehat{\Phi}^*(\mathcal{D})(Suc'(s_0; s')) = \widehat{\Phi}^*(\mathcal{D})(Suc'(s_0)) ;' \mathcal{D}(s')$.

  Putting these results together gives that

$$\begin{array}{rcl}
\Phi^*(\mathcal{D})(s; s') & = & \{\, \widehat{\Phi}^*(\mathcal{D})(M) \mid M \in Suc(s; s') \,\} \\
& = & \{\, \widehat{\Phi}^*(\mathcal{D})(Suc'(s_0; s')) \mid s \overset{\nu}{\rightsquigarrow} s_0 \,\} \\
& = & \{\, \widehat{\Phi}^*(\mathcal{D})(Suc'(s_0)) ;' \mathcal{D}(s') \mid s \overset{\nu}{\rightsquigarrow} s_0 \,\} \\
& = & \{\, \widehat{\Phi}^*(\mathcal{D})(Suc'(s_0)) \mid s \overset{\nu}{\rightsquigarrow} s_0 \,\} ; \mathcal{D}(s')
\end{array}$$

$$\begin{array}{rcl} & = & \Phi^*(\mathcal{D})(s)\,;\mathcal{D}(s') \\ \text{[ind. hyp.]} & = & \mathcal{D}(s)\,;\mathcal{D}(s') \\ & = & \mathcal{D}(s;s') \end{array}$$

Note that the induction assumption is only used for $s$, not for $s'$. (Recall the complexity $wgt(s')$ of the statement $s'$ is not necessarily less than the complexity $wgt(s;s')$ of the statement $s;s'$.) The second and third equation follow from the properties derived above. The other equations are clear from the definitions of $\Phi^*$, ; and $\mathcal{D}$. $\square$

In the operational meaning of a statement, unmatched synchronization actions are not present. Compared to the denotational domain, the branching domain $\mathbb{P}_b$ removes the possibility of synchronization actions. Removing the synchronization actions may cause 'missing probability', i.e. a total probability of less than one. As for denotational processes, this missing probability is seen as a probability of deadlock for an unconditional interpretation of choice. For a conditional interpretation the ratios have to be normalized to obtain the actual probabilities.

**Definition 4.2.35** *The branching domain $\mathbb{P}_b$ is given by*

$$\begin{array}{rcl} \mathbb{P}_b & \simeq & \mathcal{P}_{nco}(\mathbb{Q}_b) + \{\, p_\epsilon \,\} \\ \mathbb{Q}_b & \simeq & \mathcal{MP}_f(\mathbb{R}_b) \\ \mathbb{R}_b & \simeq & POAct \times id_{\frac{1}{2}}(\mathbb{P}_b) \end{array}$$

The branching domain $\mathbb{P}_b$ is the same as the denotational domain except that synchronization actions are no longer present. An abstraction function is used to relate the denotational domain with the branching domain. All that is required is to remove the synchronization actions from a process.

**Definition 4.2.36** *The functions* $abs_1 \colon \mathbb{P}_d + \{\, p_\epsilon \,\} \to \mathbb{P}_b$ *and* $abs_1' \colon \mathbb{Q}_d \to \mathbb{Q}_b$ *are defined by*

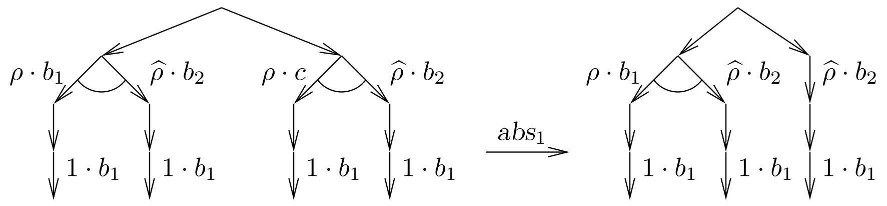
$$\begin{array}{rcl} abs_1(p_\epsilon) & = & p_\epsilon \\ abs_1(p) & = & \{\, abs_1'(q) \mid q \in p \,\} \\ abs_1'(q) & = & \{\!\mid \langle \beta, abs_1(\hat{p}) \rangle \mid \langle \beta, \hat{p} \rangle \in q \mid\!\} \end{array}$$

*where $\hat{p}$ ranges over $\mathbb{P}_d + \{\, p_\epsilon \,\}$ and $\langle \beta, \hat{p} \rangle$ should be read as $\beta$ for $\hat{p} = p_\epsilon$. Recall that $\beta = \rho \cdot b$ is an* observable *action $b$ in OAct labeled with a probability $\rho$ in* $[0, 1]$.

The function $abs_1'$ removes synchronization actions from a subprocess in $\mathbb{Q}_d$ and keeps only the observable actions. The function $abs_1$ applies the function $abs_1'$ to every subprocess in a given process from $\mathbb{Q}_d$.

**Example 4.2.37** *Let in this example $b_1, b_2$ be actions in OAct and $c$ and action in Sync. The abstraction of the process* $\{\, \{\!\mid \langle \rho {\cdot} b_1, \{\, \{\!\mid 1 {\cdot} b_1 \mid\!\} \,\} \rangle, \langle (1-\rho) {\cdot} b_2, \{\, \{\!\mid 1 {\cdot} b_1 \mid\!\} \,\} \rangle \mid\!\}, \{\!\mid \langle \rho {\cdot} c, \{\, \{\!\mid 1 {\cdot}$

$b_1 \}\!\!\}\, \} \rangle, \langle (1-\rho)\cdot b_2, \{ \{\!\!\{ 1\cdot b_1 \}\!\!\}\, \} \rangle \}\!\!\}\, \}$ *is the process* $\{ \{\!\!\{ \langle \rho\cdot b_1, \{ \{\!\!\{ \langle 1\cdot b_1, p_\epsilon \rangle \}\!\!\}\, \} \rangle, \langle (1-\rho)\cdot b_2, \{ \{\!\!\{ \langle 1\cdot b_1, p_\epsilon\rangle \}\!\!\}\, \} \rangle \}\!\!\}, \{\!\!\{ \langle (1-\rho)\cdot b_2, \{ \{\!\!\{ \langle 1\cdot b_1, p_\epsilon\rangle \}\!\!\}\, \} \rangle \}\!\!\}\, \}$. *In the tree representation of the branching process the trailing* $p_\epsilon$ *at each leaf is omitted and in both trees the shorthand* $\widehat{\rho}$ *is used for* $(1-\rho)$.



The operation-like model $\mathcal{O}^b$ yields branching processes in the domain $\mathbb{P}_b$.

**Definition 4.2.38** *The operational-like model* $\mathcal{O}^b : Res \to \mathbb{P}_b$ *is given by*

$$\begin{aligned} \mathcal{O}^b(\mathrm{E}) &= p_\epsilon \\ \mathcal{O}^b(s) &= \{ \widehat{\mathcal{O}}^b(M) \mid M \in Suc_{obs}(s) \} \\ \widehat{\mathcal{O}}^b(M) &= \{\!\!\{ \langle \beta, \mathcal{O}^b(r) \rangle \mid \langle \beta, r\rangle \in M \}\!\!\} \end{aligned}$$

The meaning of the empty resumption E is the empty process $p_\epsilon$. The meaning of a statement $s$ can be found by looking at which observable steps $s$ can take. The steps that $s$ can take are collected in the successor set $Suc_{obs}(s)$ of $s$.

The operational-like model $\mathcal{O}^b$ yielding processes in the branching domain $\mathbb{P}_b$ is an abstraction of the operational-like model $\mathcal{O}^*$ with values in the denotational domain $\mathbb{P}_d$. If from a process $\mathcal{O}^*(s)$ the synchronization actions are removed the process $\mathcal{O}^b(s)$ is obtained.

**Lemma 4.2.39** *The operational-like branching model is an abstraction of the operational-like model* $\mathcal{O}^*$: $\mathcal{O}^b = abs_1 \circ \mathcal{O}^*$.

**Proof** This proof uses the same idea as the proof of lemma 4.2.34. One shows that $abs_1 \circ \mathcal{O}^*$ is a fixed point of the higher-order function $\Phi^b$ which was used implicitly in the definition of $\mathcal{O}^b$. In other words $abs \circ \mathcal{O}^*$ satisfies the same equations. By uniqueness of the fixed point, $\mathcal{O}^b$ and $abs_1 \circ \mathcal{O}^*$ have to coincide. Note that

$$\begin{aligned} \widehat{\Phi}^b(abs_1 \circ \mathcal{O}^*)(M) &= \{\!\!\{ \langle \beta, abs_1(\mathcal{O}^*(r))\rangle \mid \langle \beta, r\rangle \in M \}\!\!\} \\ &= abs_1'(\{\!\!\{ \langle \beta, \mathcal{O}^*(r)\rangle \mid \langle \beta, r\rangle \in M \}\!\!\}) \\ &= abs_1'(\{\!\!\{ \langle \alpha, \mathcal{O}^*(r)\rangle \mid \langle \alpha, r\rangle \in M \sqcup M' \}\!\!\}) \end{aligned}$$

for all multisets $M'$ containing only pairs starting with a synchronization action. The second equation holds as $abs_1'$ removes all pairs starting with a synchronization action. Using these two equalities we obtain

$$\begin{aligned} \Phi^b(abs_1 \circ \mathcal{O}^*)(\mathrm{E}) &= p_\epsilon \;=\; abs_1(p_\epsilon) \;=\; abs_1(\mathcal{O}^*(\mathrm{E})) \\ \Phi^b(abs_1 \circ \mathcal{O}^*)(s) &= \{ \widehat{\Phi}^b(abs_1 \circ \mathcal{O}^*)(M) \mid M \in Suc_{obs}(s) \} \\ &= \{ abs_1'(\{\!\!\{ \langle \beta, \mathcal{O}^*(r)\rangle \mid \langle \beta, r\rangle \in M \}\!\!\}) \mid M \in Suc_{obs}(s) \} \\ &= \{ abs_1'(\{\!\!\{ \langle \alpha, \mathcal{O}^*(r)\rangle \mid \langle \alpha, r\rangle \in M \}\!\!\}) \mid M \in Suc(s) \} \\ &= abs_1(\mathcal{O}^*(s)) \end{aligned}$$

That a multiset $M$ from $Suc_{obs}(s)$ can be replaced by a multiset from $Suc(s)$ is clear, as only pairs starting with a synchronization action are added to $M$. $\square$

The branching operational-like model $\mathcal{O}^m$ is an abstraction of the operational-like model $\mathcal{O}^*$. This implies that the the model $\mathcal{O}^b$ identifies all statements that $\mathcal{O}^*$ identifies. The following example shows that the reverse does not hold.

**Example 4.2.40** *Let for this example $c_1$, $c_2$ be two different actions in Sync. The programs $c_1$ and $c_2$ are identified by $\mathcal{O}^b$ but not by $\mathcal{O}^*$*

$$\begin{array}{rcl} \mathcal{O}^b(c_1) & = & \{\,\emptyset\,\} = \mathcal{O}^b(c_2) \\ \mathcal{O}^*(c_1) & = & \{\,\{\!\!\{\, 1 \cdot c_1 \,\}\!\!\}\,\} \\ \mathcal{O}^*(c_2) & = & \{\,\{\!\!\{\, 1 \cdot c_2 \,\}\!\!\}\,\} \end{array}$$

The next step is to introduce an operational-like model on an intermediate domain and show that this is an abstraction of the operational-like model on the branching domain. The intermediate domain is a 'mixed' domain: Processes in the intermediate domain do not contain any nondeterministic branching, but can have probabilistic branching.

**Definition 4.2.41** *The intermediate domain $\mathbb{P}_m$ is given by*

$$\mathbb{P}_m \;=\; \mathcal{P}_{nco}(\mathbb{Q}_m)$$

*where $\mathbb{Q}_m$ is given by the domain equations:*

$$\begin{array}{rcl} \mathbb{Q}_m & \simeq & \mathcal{MP}_f(\mathbb{R}_m) + \{\, q_\epsilon \,\} \\ \mathbb{R}_m & \simeq & POAct \times id_{\frac{1}{2}}(\mathbb{Q}_m) \end{array}$$

*with $q_\epsilon$ a fresh symbol denoting an empty subprocess.*

The domains $\mathbb{Q}_m$ and $\mathbb{R}_m$ are defined as the unique solution (up to isomorphism) of the given domain equations. The domain $\mathbb{P}_m$ consists of the nonempty compact subsets of $\mathbb{Q}_m$ and does not need to be defined using domain equations as its definition is not recursive.

In a process in the branching domain, an action can be followed by another process $p$. In the intermediate domain, an action can only be followed by a probabilistic subprocess $q$. The new symbol $q_\epsilon$ denotes an empty subprocess. The singleton set containing only the empty subprocess is used to denote the process that does nothing, replacing the symbol $p_\epsilon$. As with the denotational domain and the branching domain, a process can still be interpreted in two ways, corresponding to an unconditional or a conditional probabilistic choice. The abstraction function $abs_2$ relates the branching domain with the intermediate domain.

**Definition 4.2.42** *The abstraction function $abs_2 \colon \mathbb{P}_b \to \mathbb{P}_m$ and the auxiliary function $abs_2' \colon \mathbb{Q}_b \to \mathbb{P}_m$ are defined by*

$$\begin{array}{rcl} abs_2(p_\epsilon) & = & \{\, q_\epsilon \,\} \\ abs_2(p) & = & \bigcup \{\, abs_2'(q) \mid q \in p \,\} \\ abs_2'(\emptyset) & = & \{\, \emptyset \,\} \\ \multicolumn{3}{l}{abs_2'(\{\!\!\{\, \langle \beta_1, p_1 \rangle, \ldots, \langle \beta_n, p_n \rangle \,\}\!\!\})} \\ & = & \{\, \{\!\!\{\, \langle \beta_1, q_1 \rangle, \ldots, \langle \beta_n, q_n \rangle \,\}\!\!\} \mid q_1 \in abs_2(p_1), \ldots, q_n \in abs_2(p_n) \,\} \end{array}$$

In the branching domain, a process can choose nondeterministically between probabilistic subprocesses after each action. In the intermediate domain all nondeterministic choices are collected in a single choice between probabilistic subprocesses. Selecting a probabilistic subprocess from $abs_2(p_i)$ corresponds to making all nondeterministic choices in the process $p_i$. For a probabilistic subprocess $q$, $abs_2'(q)$ collects all nondeterministic choices for all probabilistic alternatives in $q$. All nondeterministic choices in $abs_2(p)$ are collected by joining all choices for all probabilistic subprocesses $q$ in $p$. If there are no actions at all in the probabilistic subprocesses, the statement deadlocks. In both the branching and the intermediate domain this is modeled by the empty multiset $\emptyset$.

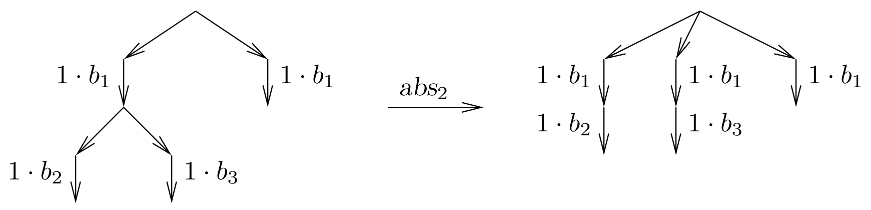
**Example 4.2.43** *Let for this example $b_1$, $b_2$ and $b_3$ be actions in OAct. The branching process*

$$\{\, \{\!\!\{ \langle 1\cdot b_1, \{\, \{\!\!\{ \langle 1\cdot b_2, p_\epsilon\rangle, \langle 1\cdot b_3, p_\epsilon\rangle \}\!\!\} \,\}\rangle \}\!\!\}, \{\!\!\{ \langle 1\cdot b_1, p_\epsilon\rangle \}\!\!\} \,\}$$

*is mapped to the intermediate process*

$$\{\, \{\!\!\{ \langle 1\cdot b_1, \{\!\!\{ \langle 1\cdot b_2, q_\epsilon\rangle \}\!\!\}\rangle \}\!\!\}, \{\!\!\{ \langle 1\cdot b_1, \{\!\!\{ \langle 1\cdot b_3, q_\epsilon\rangle \}\!\!\}\rangle \}\!\!\}, \{\!\!\{ \langle 1\cdot b_1, q_\epsilon\rangle \}\!\!\} \,\}$$

*by the abstraction function* $abs_2$.



*In the tree representation of the branching process the trailing $p_\epsilon$ at each leaf is omitted. In the tree representation of the intermediate process the trailing $q_\epsilon$ at each leaf is omitted.*

The operational-like model $\mathcal{O}^m$ yields values in the intermediate domain $\mathbb{P}_m$ introduced in definition 4.2.41.

**Definition 4.2.44** *The intermediate operational-like model $\mathcal{O}^m$ is given by*

$$\begin{aligned}
\mathcal{O}^m(\mathrm{E}) &= \{\, q_\epsilon \,\} \\
\mathcal{O}^m(s) &= \bigcup\{\, \widehat{\mathcal{O}}^m(M) \mid M \in Suc_{obs}(s) \,\} \\
\widehat{\mathcal{O}}^m(\emptyset) &= \{\, \emptyset \,\} \\
\widehat{\mathcal{O}}^m(\{\!\!\{ \langle \beta_1, r_1\rangle, \ldots, \langle \beta_n, r_n\rangle \}\!\!\}) & \\
&= \{\, \{\!\!\{ \langle \beta_1, q_1\rangle, \ldots, \langle \beta_n, q_n\rangle \}\!\!\} \mid q_1 \in \mathcal{O}^m(r_1), \ldots, q_n \in \mathcal{O}^m(r_n) \,\}
\end{aligned}$$

In the intermediate domain actions may not be followed by a process, only by a probabilistic subprocess. The construction $\langle \beta, \mathcal{O}^b(r)\rangle$, used in definition 4.2.38, cannot be used here. A probabilistic subprocess $q$ is selected from $\mathcal{O}^m(r)$ and $\langle \beta, q\rangle$ is used instead. In this way the nondeterministic choices are collected as is done in the abstraction from the branching domain to the intermediate domain. As the intermediate operational-like model removes the nondeterministic branching in the same way as the abstraction function $abs_2$ does, the following lemma is not surprising.

**Lemma 4.2.45** *The intermediate operational-like model $\mathcal{O}^m$ is an abstraction from the branching model $\mathcal{O}^b$: $\mathcal{O}^m = abs_2 \circ \mathcal{O}^b$.*

The same technique as used in lemmas 4.2.34 and 4.2.39 can be applied to prove this lemma. The intermediate operational-like model $\mathcal{O}^m$ is an abstraction of the branching operational-like model $\mathcal{O}^b$. The gives that the model $\mathcal{O}^m$ identifies all statements that $\mathcal{O}^b$ identifies. The following example shows that the reverse does not hold.

**Example 4.2.46** *Let for this example $b_1$, $b_2$ and $b_3$ be actions in OAct. The programs $b_1; (b_2 \,\Box\, b_3)$ and $b_1; b_2 \,\Box\, b_1; b_3$ are identified by the model $\mathcal{O}^m$ but not by the model $\mathcal{O}^b$*

$$\begin{array}{rcl}
\mathcal{O}^m(b_1; (b_2 \,\Box\, b_3)) & = & \{\, \{\!\!\{\, \langle 1 \cdot b_1, \{\!\!\{\, \langle 1 \cdot b_2, q_\epsilon \rangle \,\}\!\!\} \,\}\!\!\}, \{\!\!\{\, \langle 1 \cdot b_1, \{\!\!\{\, \langle 1 \cdot b_3, q_\epsilon \rangle \,\}\!\!\} \,\}\!\!\} \,\} \\
 & = & \mathcal{O}^m(b_1; b_2 \,\Box\, b_1; b_3) \\
\mathcal{O}^b(b_1; (b_2 \,\Box\, b_3)) & = & \{\, \{\!\!\{\, \langle 1 \cdot b_1, \{\, \{\!\!\{\, \langle 1 \cdot b_2, p_\epsilon \rangle \,\}\!\!\}, \{\!\!\{\, \langle 1 \cdot b_3, p_\epsilon \rangle \,\}\!\!\} \,\} \rangle \,\}\!\!\} \,\} \\
\mathcal{O}^b(a; b \,\Box\, a; c) & = & \{\, \{\!\!\{\, \langle 1 \cdot b_1, \{\, \{\!\!\{\, \langle 1 \cdot b_2, p_\epsilon \rangle \,\}\!\!\} \,\} \rangle \,\}\!\!\}, \{\!\!\{\, \langle 1 \cdot b_1, \{\, \{\!\!\{\, \langle 1 \cdot b_3, p_\epsilon \rangle \,\}\!\!\} \,\} \rangle \,\}\!\!\} \,\}
\end{array}$$

The final step is to remove the probabilistic branching structure from the intermediate domain. A linear meaning in the operational domain should be obtained. As a process in the domain $\mathbb{P}_o$ has only one interpretation, while a process in the intermediate domain $\mathbb{P}_m$ has two possible interpretations, two different abstraction functions are given. The first abstraction function $abs_{3u}$ yields processes in $\mathbb{P}_o$ corresponding to an unconditional interpretation of probabilistic choice. The second abstraction function $abs_{3c}$ yields processes in $\mathbb{P}_o$ corresponding to a conditional interpretation of probabilistic choice.

**Definition 4.2.47** *The abstraction functions $abs_{3i}\colon \mathbb{P}_m \to \mathbb{P}_o$ and the auxiliary functions $abs'_{3i}\colon \mathbb{Q}_m \to \mathbb{Q}_o$ are defined by*

$$\begin{array}{rcl}
abs_{3i}(p) & = & \{\, abs'_{3i}(q) \mid q \in p \,\} \\
abs'_{3i}(q_\epsilon) & = & \Delta_\epsilon \\
abs'_{3i}(\emptyset) & = & \Delta_\delta \\
abs'_{3u}(\{\!\!\{\, \langle \rho_1 \cdot b_1, q_1 \rangle, \ldots, \langle \rho_n \cdot b_n, q_n \rangle \,\}\!\!\}) & = & (1-\pi)\Delta_\delta + \sum_{j=1}^n \rho_j (abs'_{3u}(q_j)/b_j) \\
abs'_{3c}(\{\!\!\{\, \langle \rho_1 \cdot b_1, q_1 \rangle, \ldots, \langle \rho_n \cdot b_n, q_n \rangle \,\}\!\!\}) & = & \sum_{j=1}^n \frac{\rho_j}{\pi} (abs'_{3c}(q_j)/b_j)
\end{array}$$

*where $\pi = \sum_{j=1}^n \rho_j$ and $i$ is $u$ for the unconditional or $c$ for the conditional interpretation of probabilistic choice.*

Recall that the operation $\bullet/b$ on measures plays the role of prefixing with action $b$. The function $abs'_{3u}$ is actually only well-defined for subprocesses where the total probability ($\pi$) is less than or equal to 1. This is not a problem because all subprocesses of interest satisfy this constraint. (To make $abs'_{3u}$ everywhere well-defined one can use an arbitrary measure as outcome for subprocesses with total probability greater than one.) The following lemma can again be proven by using the technique of lemmas 4.2.34 and 4.2.39.

**Lemma 4.2.48** *Both operational models are abstractions of the intermediate operational-like model: $\mathcal{O}_i(s) = abs_{3i}(\mathcal{O}^m(s))$ where $i$ is $u$ for the unconditional or $c$ for the conditional interpretation of probabilistic choice.*

**Example 4.2.49** *Let for this example $b$ be an action in OAct and $c$ an action in Sync. For the program $b \oplus_{\frac{1}{2}} c$ the intermediate operation-like model gives $\mathcal{O}^m(b \oplus_{\frac{1}{2}} c) = \{ \{\!| \langle \frac{1}{2} \cdot b, q_\epsilon \rangle |\!\} \}$. The abstractions for the unconditional and the conditional interpretation of this process are $\{ \frac{1}{2}\Delta_b + \frac{1}{2}\Delta_\delta \}$ and $\{ \Delta_b \}$ respectively.*

These results can now be combined to show that the the operational semantics is an abstraction of the denotational semantics.

**Theorem 4.2.50** *There exist abstraction functions $abs_i : \mathbb{P}_d \to \mathbb{P}_o$ such that $\mathcal{O}_i[\![\bullet]\!] = abs_i \circ \mathcal{D}[\![\bullet]\!]$ where $i$ is $u$ for the unconditional or $c$ for the conditional interpretation of probabilistic choice.*

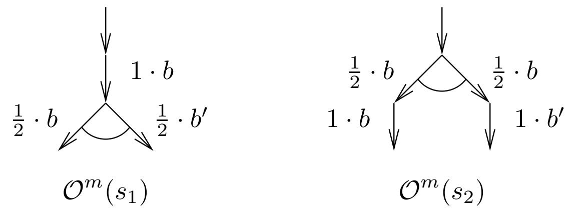
**Proof** Take $abs_i = abs_{3i} \circ abs_2 \circ abs_1$ then

$$\begin{array}{rcll} \mathcal{O}_i[\![s]\!] & = & \text{[lemma 4.2.48]} & abs_{3i}(\mathcal{O}^m(s)) \\ & = & \text{[lemma 4.2.45]} & abs_{3i}(abs_2(\mathcal{O}^b(s)) \\ & = & \text{[lemma 4.2.39]} & abs_{3i}(abs_2(abs_1(\mathcal{O}^*(s))) \\ & = & \text{[lemma 4.2.34]} & abs_{3i}(abs_2(abs_1(\mathcal{D}(s))) \\ & = & \text{[definition 4.2.31]} & abs_i(\mathcal{D}[\![s]\!]) \end{array}$$ $\Box$

The result obtained in theorem 4.2.50 implies that $\mathcal{D}$ is correct with respect to both operational semantics, which means that if $\mathcal{D}$ identifies two statements then $\mathcal{O}_u$ and $\mathcal{O}_c$ will also identify them. The following example shows that the reverse does not hold. Both operational semantics identify more statements than the denotational semantics. The unconditional and conditional operational semantics are incomparable; there exist statements which are identified by the unconditional, but not by the conditional model but there also exist statements which are identified by the conditional but not by the unconditional model.

**Example 4.2.51** *Take $s_1 = b; (b \oplus_{\frac{1}{2}} b')$, $s_2 = b; b \oplus_{\frac{1}{2}} b; b'$. These statements are identified by both operational semantics, but not by the operational-like model $\mathcal{O}^m$, and also not by the denotational semantics*

$$\begin{array}{rcl} \mathcal{O}_i(s_1) & = & \{ \frac{1}{2}\Delta_{bb} + \frac{1}{2}\Delta_{bb'} \} \;=\; \mathcal{O}_i(s_2) \\ \mathcal{O}^m(s_1) & = & \{ \{\!| \langle 1 \cdot b, \{\!| \langle \frac{1}{2} \cdot b, q_\epsilon \rangle, \langle \dfrac{1}{2} \cdot b', q_\epsilon \rangle |\!\} \rangle |\!\} \} \\ & \neq & \{ \{\!| \langle \frac{1}{2} \cdot b, \{\!| \langle 1 \cdot b, q_\epsilon \rangle |\!\} \rangle, \langle \frac{1}{2} \cdot b, \{\!| \langle 1 \cdot b', q_\epsilon \rangle |\!\} \rangle |\!\} \} \;=\; \mathcal{O}^m(s_2) \end{array}$$



*As the operational-like model $\mathcal{O}^m$ already gives different meanings for programs $s_1$ and $s_2$, the denotational model definitely gives different meanings for these programs.*

*Take* $s_3 = b$ *and* $s_4 = b \oplus_{\frac{1}{2}} c$ *then* $\mathcal{O}_c(s_3) = \{\, \Delta_b \,\} = \mathcal{O}_c(s_4)$ *but* $\mathcal{O}_u(s_3) = \{\, \Delta_b \,\} \neq \{\, \frac{1}{2}\Delta_b + \frac{1}{2}\Delta_\delta \,\} = \mathcal{O}_u(s_4)$.
*Finally take* $s_5 = b; (b \oplus_{\frac{1}{2}} c)$ *and* $s_6 = b; b \oplus_{\frac{1}{2}} b; c$ *then* $\mathcal{O}_u(s_5) = \{\, \frac{1}{2}\Delta_{bb} + \frac{1}{2}\Delta_{b\delta} \,\} = \mathcal{O}_u(s_6)$ *but* $\mathcal{O}_c(s_5) = \{\, \Delta_{bb} \,\} \neq \{\, \frac{1}{2}\Delta_{bb} + \frac{1}{2}\Delta_{b\delta} \,\} = \mathcal{O}_c(s_6)$.

The following summarizes the results of this subsection.

$$\begin{tikzcd}
& & & \mathcal{O}_u \arrow[dl, shift left, "/" marking, "\text{ex 4.2.51}"] \arrow[dd, leftrightarrow, bend left=60, "/" marking, "\text{ex 4.2.51}"] \\
\mathcal{E} = \mathcal{O}^* \arrow[r, shift left, "abs_1"] \arrow[urrr, bend left, "abs_u"] \arrow[drrr, bend right, "abs_c"'] & \mathcal{O}^b \arrow[l, shift left, "/" marking, "\text{ex 4.2.40}"] \arrow[r, shift left, "abs_2"] & \mathcal{O}^m \arrow[l, shift left, "/" marking, "\text{ex 4.2.46}"] \arrow[ur, shift left, "abs_{3u}"] \arrow[dr, shift right, "abs_{3c}"'] & \\
& & & \mathcal{O}_c \arrow[ul, shift right, "/" marking, "\text{ex 4.2.51}"']
\end{tikzcd}$$

Each model is an abstraction of the model to its left. Therefore, the further to the right a model is, the more programs it identifies. For each abstraction there is an example that shows that strictly more statements are identified after the abstraction.

## 4.3 Priority for probability

In this section the language $\mathcal{L}_{pnd}$ with both nondeterminism and probabilistic choice is again considered. Although the language remains the same, the interpretation of statements is different from the interpretation in the previous section. In this section priority is given to probability. The probabilistic choices are always made first when executing a 'mixed' program in which both probabilistic choices and nondeterministic decisions have to be made to find the next action of the program. Giving priority to probability over nondeterminism corresponds to a 'resource oriented' view of nondeterminism. The probabilistic choice has to be made first to see which nondeterministic options, i.e. which resources, the system will offer.

In the introduction to this chapter the local and the global interpretation of nondeterminism have been explained. The unconditional and conditional interpretation of probabilistic choice are also mentioned in the introduction and further explained in section 4.2. A local nondeterministic choice and an unconditional probabilistic choice are made independently of the environment. In contrast a global nondeterministic choice and a conditional probabilistic choice will 'adapt' to the environment, restricting the choice to the alternatives which will not fail within this environment.

In section 4.2 it was concluded that a setting with priority for nondeterminism is not well suited for a global nondeterministic choice. For the same reasons as given there, a setting with priority for probability is not well suited for a conditional probabilistic choice. To be conditional, the probabilistic choice has to react to actions from the environment. However, the probabilistic choice has priority and must be resolved before any nondeterministic choices are made. If the actions of the environment depend on the outcome of a nondeterministic choice, the actions of the environment cannot be known at the time the probabilistic choice has to be resolved, so the probabilistic choice cannot depend on them as is necessary for a conditional choice.

The conditional interpretation of probabilistic choice is not considered in this section. This leaves two possible interpretations, an unconditional interpretation of probabilistic choice combined with either a local or a global interpretation of nondeterminism. Both interpretations are dealt with in the same framework.

Making a probabilistic choice or nondeterministic choice can be represented by a transition in the transition system. If this is done, the choice is said to be resolved explicitly. If resolving a choice is always directly followed by the execution of an action, one can combine the transition for resolving the choice with the transition for executing the action. The single transition that resolves the choice and executes the action is said to implicitly resolve the choice by executing the action.

When giving priority to nondeterminism, the probabilistic choice is made right before executing an action. The probabilistic choice is, therefore, a choice between actions to execute. The making of the probabilistic choice and the execution of the action can be combined by associating a probability with the action. In this way the probabilistic choice is resolved implicitly when executing the action.

In contrast, if priority is given to probabilistic choice, a probabilistic choice may have to be made between alternatives which still have to make nondeterministic decisions before it is clear which action will be executed. The probabilistic choice is not a choice between actions but a choice between programs. The probabilistic choice cannot be made implicitly combined with the execution of an action, as the choice has to be made before the next action is decided. Instead of choosing from actions to execute, the probabilistic choice is a choice between programs to execute.

In chapter 3 the notions generative and stratified were introduced and in sections 3.3 and 4.2 generative transition systems were given. A generative transition system uses an implicit probabilistic choice between actions whereas a stratified transition system uses a probabilistic choice between programs. In this section a stratified transition system $\mathcal{T}_{pnd}^{(2)}$ with priority for probabilistic choice is given. As the probabilistic choice cannot be made implicitly, it will be modeled explicitly in the transition system by introducing an auxiliary step labeled with the probability that this step is taken. With priority for probability, the nondeterminism is resolved right before the execution of the action. The nondeterminism can be resolved implicitly with the execution of an action, removing the need for the auxiliary $\nu$ step used to resolve the nondeterminism in the transition system $\mathcal{T}_{pnd}$ of the previous section.

In the next subsection the stratified transition system $\mathcal{T}_{pnd}^{(2)}$ with priority for probabilistic choice is given. After deriving some properties of the transition system $\mathcal{T}_{pnd}^{(2)}$ in subsection 4.3.2 the operational semantics for the local and global interpretation of nondeterminism are defined in subsection 4.3.3. The operational domain is a branching domain. In subsection 4.3.4 a denotational model is constructed, which is compared with both operational models in subsection 4.3.5. How the branching operational domain can be linearized is shown in subsection 4.3.6.

### 4.3.1 A transition system with priority for probabilistic choice: $\mathcal{T}^{(2)}_{pnd}$

To describe systems offering resources, the same syntax is used as for programs describing games. The programs in the language $\mathcal{L}_{pnd}$ may contain both nondeterminism and probabilistic choice. The interpretation of statements of the language is adapted to fit the 'resource oriented' description of systems.

**Example 4.3.1** *Consider a vending machine offering both tea and coffee. Using atomic actions* tea, coffee *to describe receiving a cup of beverage, the following describes a machine offering both tea and coffee*

$$tea \,\square\, coffee$$

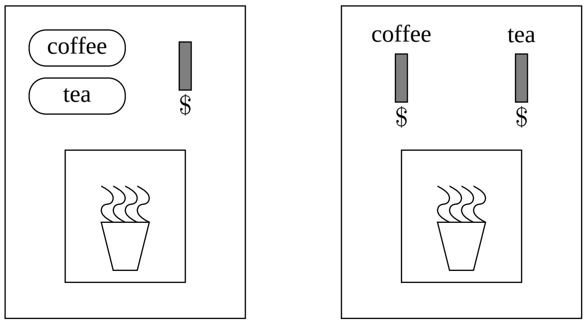
*Adding the atomic action* coin *to describe the insertion of a coin into the machine, one can also describe that one must first pay for the tea or coffee.*

$$coin; (tea \,\square\, coffee)$$

*Note that this is a different machine than the machine described by*

$$(coin; tea) \,\square\, (coin; coffee)$$

*where one first has to choose for tea or coffee before inserting the coin.*



$coin; (tea \,\square\, coffee)$ $\quad$ $(coin; tea) \,\square\, (coin; coffee)$

*The program*

$$coin; ((tea \oplus_\rho coin') \,\square\, (coffee \oplus_\sigma coin'))$$

*describes a vending machine that may have run out of beverages: Tea is only available with probability $\rho$ and coffee with probability $\sigma$. If a beverage is not available, the machine offers a refund modeled by the atomic action $coin'$.*

*All of these machines only deliver a beverage once. Recursion can be used to describe that, after providing the beverage, the machine returns to its original state. The program $x$ with the body of $x$ given by $D(x) = s; x$ where $s$ is any of the programs above, describes such a machine.*

As explained in the introduction of this section, resolving the probabilistic choice is made explicit. Resolving a probabilistic choice will lead to an auxiliary probabilistic step in the transition system. The label associated with an auxiliary probabilistic step will be the probability that this step is taken. Thus, the set of probabilities $[0, 1]$ is part of the set of transition labels. As the probability is resolved explicitly, there is no need to assign a probability to actions. The set $Act$ of actions is also part of the set of transition labels. These are the only two types of labels used in $\mathcal{T}^{(2)}_{pnd}$. No auxiliary label for resolving nondeterminism is required as nondeterminism is resolved implicitly in $\mathcal{T}^{(2)}_{pnd}$.

$$Lab \quad = \quad [0, 1] \cup Act$$

Transitions with a label in $[0, 1]$ are called probabilistic transitions and transitions with a label in $Act$ are called action transitions. Recall that the set of actions $Act$ is divided into the set of observable actions $OAct$ ranged over by $b$ and the set of synchronization action $Sync$ ranged over by $c$.

In the previous section the set of statements was extended to $Stat^+$ by including the auxiliary operators $\mathbin{|\!|\!\_}$ and $|$. There is no need to extend $Stat$ with these operators here. The nondeterminism is not resolved explicitly, so the program $s \,\|\, s'$ will produce actions directly without first resolving into $s \mathbin{|\!|\!\_} s'$, $s' \mathbin{|\!|\!\_} s$, or $s|s'$.

The information required to describe the state of an execution of a program in $\mathcal{L}_{pnd}$ is the part of the program that remains to be executed. As before, a resumption describes the remainder of a program. A resumption is either a statement or the empty resumption E:

$$r \quad ::= \quad s \mid \mathrm{E}$$

Configurations are also defined as before. A configurations in $\mathcal{T}^{(2)}_{pnd}$ is a resumption together with a declaration, i.e. $Conf = Decl \times Res$. The declaration part is dropped from the notation as a single declaration is assumed to be fixed.

In section 4.2 the nondeterminism has to be resolved first and probabilistic choices and actions are only possible for (nondeterministically) resolved statements. The situation in this section is similar, except that first all probabilistic choices have to be made and nondeterministic choices and actions are only possible for *probabilistically resolved* statements. The following definition makes the notion of a probabilistically resolved statement precise.

**Definition 4.3.2** *The following statements are called probabilistically resolved: The statement $a$ is probabilistically resolved. The statement $s_1; s_2$ is probabilistically resolved exactly when $s_1$ is probabilistically resolved. The statement $x$ is probabilistically resolved if and only if $D(x)$ is probabilistically resolved. The statements $s_1 \Box s_2$ and $s_1 \,\|\, s_2$ are probabilistically resolved if both $s_1$ and $s_2$ are probabilistically resolved. The set of all resolved statements is denoted by $Stat_{res}$.*

Well-definedness of $Stat_{res}$ is clear by weight induction using the function $wgt$ introduced in definition 4.2.9. In the remainder of this section, the term resolved refers to probabilistically resolved. That $a$ is resolved is clear. For $s = s_1; s_2$ the first action must come from $s_1$, so it is sufficient to resolve the probability in $s_1$ to find the possible first actions of $s$. For $s = s_1 \,\Box\, s_2$ and $s = s_1 \| s_2$ both $s_1$ and $s_2$ may produce the first action that is executed. Both $s_1$ and $s_2$ must be resolved for $s$ to be resolved. The statement $s_1 \oplus_\rho s_2$ is clearly not resolved (for any statements $s_1$ and $s_2$). Other examples of statements that are not resolved are $(a \oplus_\rho b); c$, $(a \oplus_\rho b) \,\Box\, c$ and $x$ with $D(x) = a \oplus_\rho b$.

Having introduced the configurations *Conf*, transition labels *Lab* and the notion of a resolved statement, the transition system $\mathcal{T}^{(2)}_{pnd}$ can be given.

**Definition 4.3.3** *The transition system $\mathcal{T}^{(2)}_{pnd}$ is given by $\mathcal{T}^{(2)}_{pnd} = (Conf, Lab, \rightarrow, Spec)$. A transition $(r, \theta, r') \in \rightarrow$ is written as $r \xrightarrow{\theta} r'$. Spec is given in parts below.*

The first part of the specification consists of the axioms and rules dealing with actions, recursion and sequential composition.

- $$a \xrightarrow{a} \mathrm{E} \qquad \text{(Act)}$$
- $$\frac{s_1 \xrightarrow{\theta} r}{s_1; s_2 \xrightarrow{\theta} r; s_2} \qquad \text{(Seq)}$$
- $$\frac{D(x) \xrightarrow{\theta} r}{x \xrightarrow{\theta} r} \qquad \text{(Rec)}$$

where $r; s_2$ in rule (Seq) should be read as $s_2$ if $r = \mathrm{E}$.

Only the transition for the statement $a$ has changed compared to the definition of $\mathcal{T}_{pnd}$ in section 4.2. The statement $a$ results in the action $a$ after which the execution is finished. No probability is added to $a$ as probabilities are resolved explicitly and are no longer associated with actions. To execute a procedure $x$, the body of the procedure $D(x)$ has to be executed. The statement $s_1 ; s_2$ behaves like $s_1$ until $s_1$ is done (the case that $r = \mathrm{E}$) after which it behaves like $s_2$.

The second part of the specification consists of the axioms and rules dealing with the resolving of probabilistic choices.

- $$s_1 \oplus_\rho s_2 \xrightarrow{\rho} s_1 \qquad \text{(Chance 1)}$$
  $$s_1 \oplus_\rho s_2 \xrightarrow{1-\rho} s_2 \qquad \text{(Chance 2)}$$
- $$\frac{s_1 \xrightarrow{\rho} s'}{\begin{array}{ll} s_1 \,\Box\, s_2 \xrightarrow{\rho} s' \,\Box\, s_2 & \text{(Choice } \rho \text{ 1)} \\ s_1 \| s_2 \xrightarrow{\rho} s' \| s_2 & \text{(Merge } \rho \text{ 1)} \end{array}}$$
- $$\frac{s_1 \xrightarrow{\rho} s' \quad s_2 \text{ resolved}}{\begin{array}{ll} s_2 \,\Box\, s_1 \xrightarrow{\rho} s_2 \,\Box\, s' & \text{(Choice } \rho \text{ 2)} \\ s_2 \| s_1 \xrightarrow{\rho} s_2 \| s' & \text{(Merge } \rho \text{ 2)} \end{array}}$$

A probabilistic choice is resolved explicitly by a transition labeled with a probability, giving $s \oplus_\rho s' \xrightarrow{\rho} s$ and $s \oplus_\rho s' \xrightarrow{1-\rho} s'$. As probability has priority, it is resolved before resolving any nondeterminism. In a statement $s \Box s'$, the probabilistic choices in $s$ and $s'$ are resolved before making the nondeterministic choice. The probability in $s$ is resolved first by use of the rule (Choice $\rho$ 1) followed by the probability in $s'$ by use of rule (Choice $\rho$ 2). The last part of example 4.3.4 below illustrates that by first resolving the probability in $s$ and then in $s'$ gives the probabilistic options one would expect. The choice to first resolve the probability in the first component and then in the second component is an arbitrary one. Reversing the order or combining probabilistic steps of both components into a single probabilistic step would yield the same options with the same probability. To be exact, the notion $\overset{\rho}{\leadsto}$ to be introduced in definition 4.3.6 below would be the same. The last statement in example 4.3.4 below illustrates the use of the rules (Choice $\rho$ 1) and (Choice $\rho$ 2).

The last part of the specification contains the rules for the action transitions of the nondeterministic constructs $\Box$ and $\|$.

- $$\frac{s_1 \xrightarrow{a} r \quad s_2 \text{ resolved}}{\begin{array}{l} s_1 \Box s_2 \xrightarrow{a} r \qquad \text{(Choice 1)} \\ s_2 \Box s_1 \xrightarrow{a} r \qquad \text{(Choice 2)} \end{array}}$$

- $$\frac{s_1 \xrightarrow{a} r \quad s_2 \text{ resolved}}{\begin{array}{l} s_1 \| s_2 \xrightarrow{a} r \| s_2 \qquad \text{(Merge 1)} \\ s_2 \| s_1 \xrightarrow{a} s_2 \| r \qquad \text{(Merge 2)} \end{array}}$$

- $$\frac{s_1 \xrightarrow{c} r_1 \quad s_2 \xrightarrow{\bar{c}} r_2}{s_1 \| s_2 \xrightarrow{\tau} r_1 \| r_2} \qquad \text{(Sync)}$$

where $r \| s_2$ and $s_2 \| r$ in rules (Merge 1) and (Merge 2) should be read as $s_2$ if $r = \mathrm{E}$ and $r_1 \| r_2$ in rule (Sync) should be read as $r_1$ if $r_2 = \mathrm{E}$ and as $r_2$ if $r_1 = \mathrm{E}$.
If all probabilistic choices have been resolved, the actions possible for $s \Box s'$ are the actions possible for $s$ and the actions possible for $s'$. As there may be more than one action possible for a statement, the execution of an action implicitly resolves nondeterminism. Only probabilistically resolved statements can produce actions, so there is no need to separately require that $s_1$ is resolved.

A parallel composition of two statements can either start with an action of one of the statements as described by rules (Merge 1) and (Merge 2) or it may synchronize if the statements are able to produce complementary synchronization actions as described by rule (Sync).

The following example shows the transition trees for several statements.

**Example 4.3.4** *As $c \xrightarrow{c} \mathrm{E}$ and $\bar{c} \xrightarrow{\bar{c}} \mathrm{E}$ by axiom (Act) and both $c$ and $\bar{c}$ are resolved, there are three transitions for $c\|\bar{c}$*

$c\|\bar{c} \xrightarrow{c} \bar{c}$ *by rule (Merge 1)*
$c\|\bar{c} \xrightarrow{\bar{c}} c$ *by rule (Merge 2)*
$c\|\bar{c} \xrightarrow{\tau} \mathrm{E}$ *by rule (Sync)*

$$\begin{tikzcd}
 & c\|\bar{c} \arrow[dl, "c"'] \arrow[d, "\tau"] \arrow[dr, "\bar{c}"] & \\
\bar{c} \arrow[d, "\bar{c}"'] & \mathrm{E} & c \arrow[d, "c"] \\
\mathrm{E} & & \mathrm{E}
\end{tikzcd}$$

*The two parallel components $c$ and $\bar{c}$ may synchronize to give $\tau$ or one may take an independent step, allowing synchronization with other components in the environment.*

*The transitions for $b \oplus_{\frac{1}{2}} \bar{c}$ are $b \oplus_{\frac{1}{2}} \bar{c} \xrightarrow{\frac{1}{2}} b$ and $b \oplus_{\frac{1}{2}} \bar{c} \xrightarrow{\frac{1}{2}} \bar{c}$. As $c$ is resolved we have*

$c\|(b \oplus_{\frac{1}{2}} \bar{c}) \xrightarrow{\frac{1}{2}} (c\|b)$ *by rule (Merge $\rho$ 1)*
$c\|(b \oplus_{\frac{1}{2}} \bar{c}) \xrightarrow{\frac{1}{2}} (c\|\bar{c})$ *by rule (Merge $\rho$ 2)*

$$\begin{tikzcd}
 & & & c\|(b \oplus_{\frac{1}{2}} \bar{c}) \arrow[dll, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & & \\
 & c\|\bar{c} \arrow[dl, "c"'] \arrow[d, "\tau"] \arrow[dr, "\bar{c}"] & & & c\|b \arrow[dl, "c"'] \arrow[dr, "b"] & \\
\bar{c} \arrow[d, "\bar{c}"'] & \mathrm{E} & c \arrow[d, "c"] & b \arrow[d, "b"'] & & c \arrow[d, "c"] \\
\mathrm{E} & & \mathrm{E} & \mathrm{E} & & \mathrm{E}
\end{tikzcd}$$

*Probabilistic choices are made before nondeterminism is resolved. The $\xrightarrow{\frac{1}{2}}$ step needs to be taken before it is possible to see if the components can synchronize. Synchronization is only possible if the outcome of the probabilistic choice is $\bar{c}$, not if the outcome is $b$.*

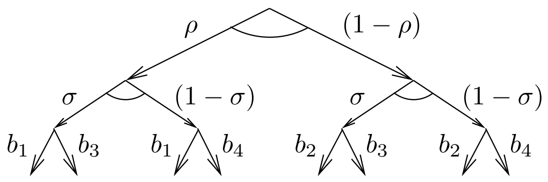
$c\,;(b \oplus_{\frac{1}{2}} \bar{c}) \xrightarrow{c} (b \oplus_{\frac{1}{2}} \bar{c})$ *by axiom (Act) and rule (Seq)*
$(b \oplus_{\frac{1}{2}} \bar{c}) \xrightarrow{\frac{1}{2}} b$ *by axiom (Chance 1)*
$(b \oplus_{\frac{1}{2}} \bar{c}) \xrightarrow{\frac{1}{2}} \bar{c}$ *by axiom (Chance 2)*

$$\begin{tikzcd}
 & c\,;(b \oplus_{\frac{1}{2}} \bar{c}) \arrow[d, "c"'] & \\
 & b \oplus_{\frac{1}{2}} \bar{c} \arrow[dl, "\frac{1}{2}"'] \arrow[dr, "\frac{1}{2}"] & \\
b \arrow[d, "b"'] & & \bar{c} \arrow[d, "\bar{c}"] \\
\mathrm{E} & & \mathrm{E}
\end{tikzcd}$$

*The first action produced is always $c$. The probabilistic choice $b \oplus_{\frac{1}{2}} \bar{c}$ is made after the first action is produced.*

*The transitions for $b_1 \oplus_\rho b_2$ are $b_1 \oplus_\rho b_2 \xrightarrow{\rho} b_1$ and $b_1 \Box b_2 \xrightarrow{1-\rho} b_2$. The transitions for $b_3 \oplus_\sigma b_4$ are $b_3 \oplus_\sigma b_4 \xrightarrow{\sigma} b_3$ and $b_3 \oplus_\sigma b_4 \xrightarrow{1-\sigma} b_4$. Combining this gives*

$$\begin{array}{rcll}
(b_1 \oplus_\rho b_2) \Box (b_3 \oplus_\sigma b_4) & \xrightarrow{\rho} & b_1 \Box (b_3 \oplus_\sigma b_4) & \textit{by rule (Choice } \rho \textit{ 1)} \\
(b_1 \oplus_\rho b_2) \Box (b_3 \oplus_\sigma b_4) & \xrightarrow{1-\rho} & b_2 \Box (b_3 \oplus_\sigma b_4) & \textit{by rule (Choice } \rho \textit{ 1)} \\
b_1 \Box (b_3 \oplus_\sigma b_4) & \xrightarrow{\sigma} & b_1 \Box b_3 & \textit{by rule (Choice } \rho \textit{ 2)} \\
b_1 \Box (b_3 \oplus_\sigma b_4) & \xrightarrow{1-\sigma} & b_1 \Box b_4 & \textit{by rule (Choice } \rho \textit{ 2)}
\end{array}$$



*The two probabilistic choices are resolved first resulting in four options, $b_1 \Box b_3$ with probability $\rho\sigma$, $b_1 \Box b_4$ with probability $\rho(1-\sigma)$, $b_2 \Box b_3$ with probability $(1-\rho)\sigma$ and $b_2 \Box b_4$ with probability $(1-\rho)(1-\sigma)$.*

### 4.3.2 Properties of the transition system $\mathcal{T}^{(2)}_{pnd}$

In this subsection some notation is introduced and several properties of the transition system are shown. Structural induction is not applicable for the proofs of these properties. Induction on the complexity of a resumption is used instead of induction on its syntactical structure. The complexity of a resumption is expressed by a weight function *wgt*. The weight function is the same as the weight function used in the previous section except that it is restricted to the resumptions used in this section.

Resolving probability is done by taking probabilistic $\xrightarrow{\rho}$ steps. The notion of a statement being resolved is supposed to describe that all probabilistic choices before taking the first action have been resolved. Resolved statements should therefore, be exactly those statements that do not take any probabilistic $\xrightarrow{\rho}$ steps. The following lemma states that this is indeed the case.

**Lemma 4.3.5** *A statement $s$ can take a probabilistic step exactly when it is not resolved: $s$ is not resolved $\iff \exists r \in Res, \rho \in [0,1] : s \xrightarrow{\rho} r$.*

**Proof** Clear by induction on the weight of the statement. $\Box$

The lemma states that being able to take a probabilistic step is the same as not being resolved, in other words, being resolved is the same as not being able to take any probabilistic steps.

The explicit resolving of the probabilistic choices is not assumed to be part of the observable behavior of a program. In the observable behavior of a program, the probabilistic steps are combined so that only the resulting probabilities can be seen and not the probabilistic steps themselves. To allow combining probabilistic steps the notation $\overset{\rho}{\leadsto}$ is introduced. For two statements $s, s' \in Stat$, $s \overset{\rho}{\leadsto} s'$ is also referred to as a (probabilistic) transition. This transition denotes that $s'$ is a resolved probabilistic alternative of $s$ which is selected with probability $\rho$. As for all probabilistic transitions, the multiplicity of a transition $s \overset{\rho}{\leadsto} s'$ is also important, therefore $\overset{\rho}{\leadsto}$ is a multiset.

**Definition 4.3.6** *For statements $s$, $s'$, $s''$ in Stat the following notation is introduced*

$$\begin{aligned} s \xrightarrow{\rho}{}^{+} s' &\iff \exists s'', \sigma, \sigma' : \rho = \sigma \cdot \sigma' \wedge s \xrightarrow{\sigma} s'' \wedge s'' \xrightarrow{\sigma'}{}^{*} s' \\ s \xrightarrow{\rho}{}^{*} s' &\iff (s = s' \wedge \rho = 1) \vee s \xrightarrow{\rho}{}^{+} s' \\ s \overset{\rho}{\leadsto} s' &\iff s \xrightarrow{\rho}{}^{*} s' \wedge s' \text{ resolved} \end{aligned}$$

*Each transition $s \xrightarrow{\rho} s'$ with multiplicity $n$ contributes $n$ to the multiplicity of $s \xrightarrow{\rho}{}^{+} s'$. Similarly each pair of transitions $s \xrightarrow{\sigma} s''$ and $s'' \xrightarrow{\sigma'}{}^{+} s'$ with multiplicities $n$ and $m$ contributes $n$ times $m$ to the multiplicity of $s \xrightarrow{\sigma\sigma'}{}^{+} s'$.*

*The multiplicity of $s \xrightarrow{1}{}^{*} s$ is one, and the multiplicity of $s \xrightarrow{\rho}{}^{*} s'$ for $s \neq s'$ is the multiplicity of $s \xrightarrow{\rho}{}^{+} s'$.*

*When $s \overset{\rho}{\rightsquigarrow} s'$ holds, the multiplicity of this transition is the same as the multiplicity of $s \xrightarrow{\rho}{}^{*} s'$.*

That $s$ can reach $s'$ by a number of probabilistic transitions, with accumulated probability $\rho$, is denoted by $s \overset{\rho}{\rightsquigarrow} s'$. The addition of probabilistic information is the main difference between the definition of $\overset{\rho}{\rightsquigarrow}$ and that of $\overset{\nu}{\rightsquigarrow}$ given in definition 4.2.11. The probabilistic information that is added consists of a probability label and of the multiplicity of a transition $s \overset{\rho}{\rightsquigarrow} s'$.

The definitions of $\xrightarrow{\rho}{}^{+}$ and $\xrightarrow{\rho}{}^{*}$ are recursive but can easily be shown to be correct by weight induction using part (a) of lemma 4.3.10 below.

**Example 4.3.7** *The multiplicity of the transition $a \overset{1}{\rightsquigarrow} a$ is one.*

*The multiplicity of the transition $a \oplus_{\frac{1}{2}} a \xrightarrow{\frac{1}{2}} a$ is two so the multiplicity of the transition $a \oplus_{\frac{1}{2}} a \overset{\frac{1}{2}}{\rightsquigarrow} a$ is also two.*

*For $(a \oplus_{\frac{1}{2}} a) \oplus_{\frac{2}{3}} a$ there are two transitions: $(a \oplus_{\frac{1}{2}} a) \oplus_{\frac{2}{3}} a \xrightarrow{\frac{2}{3}} (a \oplus_{\frac{1}{2}} a)$ and $(a \oplus_{\frac{1}{2}} a) \oplus_{\frac{2}{3}} a \xrightarrow{\frac{1}{3}} a$ both with multiplicity one. As seen above the multiplicity of the transition $a \oplus_{\frac{1}{2}} a \overset{\frac{1}{2}}{\rightsquigarrow} a$ is two, therefore, the first transition contributes two to the multiplicity of $(a \oplus_{\frac{1}{2}} a) \oplus_{\frac{2}{3}} a \overset{\frac{1}{3}}{\rightsquigarrow} a$. The second transition contributes one to the multiplicity of $(a \oplus_{\frac{1}{2}} a) \oplus_{\frac{2}{3}} a \overset{\frac{1}{3}}{\rightsquigarrow} a$. The total multiplicity of $(a \oplus_{\frac{1}{2}} a) \oplus_{\frac{2}{3}} a \overset{\frac{1}{3}}{\rightsquigarrow} a$ is three.*

*For the statement $(a \oplus_{\frac{1}{2}} a) \oplus_{\frac{1}{2}} (a \oplus_{\frac{1}{2}} a)$ the only transition is $(a \oplus_{\frac{1}{2}} a) \oplus_{\frac{1}{2}} (a \oplus_{\frac{1}{2}} a) \xrightarrow{\frac{1}{2}} (a \oplus_{\frac{1}{2}} a)$ with multiplicity two. The multiplicity of the transition $a \oplus_{\frac{1}{2}} a \overset{\frac{1}{2}}{\rightsquigarrow} a$ is also two. The product of these multiplicities is the multiplicity of the transition $(a \oplus_{\frac{1}{2}} a) \oplus_{\frac{1}{2}} (a \oplus_{\frac{1}{2}} a) \overset{\frac{1}{4}}{\rightsquigarrow} a$. In other words the multiplicity of $(a \oplus_{\frac{1}{2}} a) \oplus_{\frac{1}{2}} (a \oplus_{\frac{1}{2}} a) \overset{\frac{1}{4}}{\rightsquigarrow} a$ is four.*

This example already shows that by counting the multiplicities, the total probability remains one. Part (d) of lemma 4.3.10 below states that is always the case.

Now that we have introduced a way to combine probabilistic transitions, the notion of a *successor set* can be given.

**Definition 4.3.8** *The nondeterministic successor set $Suc'(s)$ of a (probabilistically) resolved statement $s$ is given by*

$$Suc'(s) \quad = \quad \{ \langle a, r \rangle \mid s \xrightarrow{a} r \}$$

*The successor set $Suc(s)$ of a statement $s$ is given by*

$$Suc(s) \quad = \quad \{\!|\, \rho \cdot Suc'(s') \mid s \overset{\rho}{\rightsquigarrow} s' \,|\!\}$$

Note that the definition of *Suc* uses the notation of multisets introduced in definition 3.2.5 part (c). As a result the pair $\rho \cdot Suc'(s')$ is included $n$ times in the successor set for a transition $s \overset{\rho}{\rightsquigarrow} s'$ with multiplicity $n$. An action transition for a statement can be described by a pair consisting of the action taken and the resulting resumption. All transitions for a resolved statement can be described by a set of such pairs, the nondeterministic successor set. For a general statement the nondeterministic successor sets for all probabilistic alternatives of the statement are included in its successor set. Note that if the statement $s$ happens to be resolved this gives $Suc(s) = \{\!|\, 1 \cdot Suc'(s) \,|\!\}$: The successor set of a resolved statement is a multiset containing one element.

**Example 4.3.9** *Using the transitions derived in examples 4.3.4 and 4.3.7 we have*

$$\begin{array}{rcl}
Suc(a \oplus_{\frac{1}{2}} a) & = & \{\!|\, \frac{1}{2} \cdot \{\, \langle a, \mathrm{E} \rangle \,\}\,,\, \frac{1}{2} \cdot \{\, \langle a, \mathrm{E} \rangle \,\} \,|\!\} \\
Suc(c \| \bar{c}) & = & \{\!|\, 1 \cdot \{\, \langle c, \bar{c} \rangle, \langle \tau, \mathrm{E} \rangle, \langle \bar{c}, c \rangle \,\} \,|\!\} \\
Suc(c \| (b \oplus_{\frac{1}{2}} \bar{c})) & = & \{\!|\, \frac{1}{2} \cdot \{\, \langle c, b \rangle, \langle b, c \rangle \,\}\,,\, \frac{1}{2} \cdot \{\, \langle c, \bar{c} \rangle, \langle \tau, \mathrm{E} \rangle, \langle \bar{c}, c \rangle \,\} \,|\!\} \\
Suc(c \,; (b \oplus_{\frac{1}{2}} \bar{c})) & = & \{\!|\, 1 \cdot \{\, \langle c, b \oplus_{\frac{1}{2}} \bar{c} \rangle \,\} \,|\!\}
\end{array}$$

$$\begin{array}{l}
Suc((b_1 \oplus_\rho b_2) \,\Box\, (b_3 \oplus_\sigma b_4)) \;= \\
\qquad \{\!|\; \rho\sigma \cdot \{\, \langle b_1, \mathrm{E} \rangle, \langle b_3, \mathrm{E} \rangle \,\}\,,\, \rho(1-\sigma) \cdot \{\, \langle b_1, \mathrm{E} \rangle, \langle b_4, \mathrm{E} \rangle \,\}\,, \\
\qquad\quad (1-\rho)\sigma \cdot \{\, \langle b_2, \mathrm{E} \rangle, \langle b_3, \mathrm{E} \rangle \,\}\,,\, (1-\rho)(1-\sigma) \cdot \{\, \langle b_2, \mathrm{E} \rangle, \langle b_4, \mathrm{E} \rangle \,\} \;|\!\}
\end{array}$$

The following lemma states some properties of the transition system $\mathcal{T}^{(2)}_{pnd}$. The first property is very useful for proofs which use weight induction and is an important part in proving the other properties. The other properties are general properties of the transition system which are useful when defining the semantics.

**Lemma 4.3.10**

*(a) If $s \xrightarrow{\rho} s'$ then $wgt(s') < wgt(s)$, with wgt as in definition 4.2.9.*

*(b) No infinite sequence $s \xrightarrow{\rho_1} s_1 \xrightarrow{\rho_2} \ldots$ exists.*

*(c) The transition system $\mathcal{T}^{(2)}_{pnd}$ is finitely branching, that is, for all $s \in \mathcal{L}_{pnd}$:*

1. *$Suc(s)$ is a finite multiset and,*
2. *$Suc'(s)$ is a finite set for each resolved statement $s$.*

*(d) The sum of probabilities of all probabilistic alternatives for each statement is one*

$$\sum \{\!|\, \rho \mid s \overset{\rho}{\rightsquigarrow} s' \,|\!\} = 1$$

The first property is directly clear by inspecting the rules of the transition system. A formal proof can be given by induction on the weight of the statement $s$ or by induction on the height of the proof tree for $s \xrightarrow{\rho} s'$. The second property is a direct consequence of the first property and states that the transition system is free of *internal divergence*. It it not possible to keep taking auxiliary $\rho$ steps. At some point the system will either stop or produce a real action. The facts that $\mathcal{T}^{(2)}_{pnd}$ is finitely branching and that the probabilities always sum up to one can easily be shown to hold by weight induction.

### 4.3.3 Operational semantics

The transition system contains information which is not considered to be observable behavior. For example the auxiliary probabilistic transition steps do not correspond to actual observable behavior. To obtain the operational semantics this additional information is removed. The domain of all possible behaviors is denoted by $\mathbb{P}_o$. The elements of $\mathbb{P}_o$ are called processes.

Giving priority to probabilistic choice over nondeterminism corresponds to a 'resource oriented' view of nondeterminism, as mentioned in the introduction of this chapter. When it is clear for the user of a system which resources are offered at which time, the moment of nondeterministic choice can be deduced. In such a case the operational model should be a branching model.

**Example 4.3.11** *Consider the following two statements which both describe a vending machine*

$$\begin{array}{rcl} s_1 & = & coin; (tea \,\Box\, coffee) \\ s_2 & = & (coin; tea) \,\Box\, (coin; coffee) \end{array}$$

*The sequences produced by these two statements are the same:* coin coffee *and* coin tea*. However, the vending machines they describe are different (see example 4.3.1). The difference in the moment of choice is observable for the user of the vending machine.*

The operational domain $\mathbb{P}_o$ is a branching domain. In subsection 4.3.6 it is shown how a model on a linear domain can be obtained by removing the branching information from processes in the operational domain $\mathbb{P}_o$.

**Definition 4.3.12** *The operational domain $\mathbb{P}_o$ is given as the unique solution of the following domain equations*

$$\begin{array}{rcl} \mathbb{P}_o & \simeq & \mathcal{MP}_f([0,1] \times \mathbb{Q}_o) + \{\, p_\epsilon \,\} \\ \mathbb{Q}_o & \simeq & \mathcal{P}_{nco}(\mathbb{R}_o) \\ \mathbb{R}_o & \simeq & OAct \times id_{\frac{1}{2}}(\mathbb{P}_o) + \{\, \delta \,\} \end{array}$$

*where $p_\epsilon$ is the empty process and $\delta$ represents deadlock as before.*

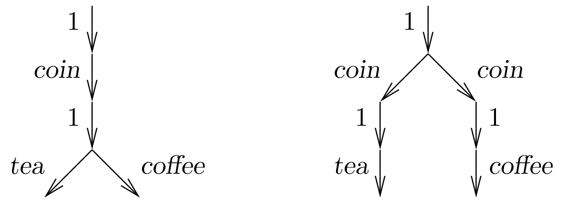
As $\mathcal{MP}_f$ and $\mathcal{P}_{nco}$ are locally nonexpansive, the functor $\{\, p_\epsilon \,\} + \mathcal{MP}_f([0,1] \times \mathcal{P}_{nco}(OAct \times id_{\frac{1}{2}}(\bullet) + \{\, \delta \,\}))$ is locally contractive. Using lemma 2.2.10 gives that the domain equations have a unique solution. The elements of the domain $\mathbb{P}_o$ are called processes. The elements of the domain $\mathbb{Q}_o$ are called subprocesses. Similar to the processes in the domain $\mathbb{P}_d$, introduced in definition 4.2.25, the processes in the domain $\mathbb{P}_o$ can be represented as trees.

**Example 4.3.13** *Let in this example tea, coffee and coin be observable actions in OAct.*

*The pair $\langle \mathit{tea}, p_\epsilon \rangle$ is an element of $\mathbb{R}_o$ and the set $\{ \langle \mathit{tea}, p_\epsilon \rangle, \langle \mathit{coffee}, p_\epsilon \rangle \}$ is a subprocess in $\mathbb{Q}_o$. The processes $p_1$ and $p_2$ given by*

$$\begin{aligned} p_1 &= \{\!\mid 1 \cdot \{ \langle \mathit{coin}, \{\!\mid 1 \cdot \{ \langle \mathit{tea}, p_\epsilon \rangle, \langle \mathit{coffee}, p_\epsilon \rangle \} \mid\!\} \rangle \} \mid\!\} \\ p_2 &= \{\!\mid 1 \cdot \{ \langle \mathit{coin}, \{\!\mid 1 \cdot \{ \langle \mathit{tea}, p_\epsilon \rangle \} \mid\!\} \rangle, \langle \mathit{coin}, \{\!\mid 1 \cdot \{ \langle \mathit{coffee}, p_\epsilon \rangle \} \mid\!\} \rangle \} \mid\!\} \end{aligned}$$

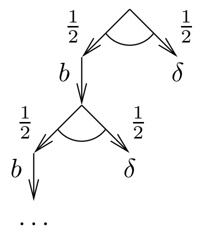
*are elements of $\mathbb{P}_o$. They can be represented by the following trees*



$p_1$ $p_2$

*In these trees the trailing $p_\epsilon$ at each leave of the tree is omitted.*

*For $b$ an action in OAct, the process $p_3$ which satisfies $p_3 = \{\!\mid \frac{1}{2} \cdot \{ \delta \}, \frac{1}{2} \cdot \{ \langle b, p_3 \rangle \} \mid\!\}$ is an element of $\mathbb{P}_o$. This process can be represented by the following tree*



$p_3$

Even if it can be observed at any moment which resources are offered, the abstract transition trees still contain information which is not observable. In an operational process this information is no longer present. There are two main differences between the processes in $\mathbb{P}_o$ and the abstract transition trees for programs (cf. example 4.3.4). As it is impossible to tell how many probabilistic choices are made before a resource is offered, all probabilistic choices before an action are combined into a single choice in the domain $\mathbb{P}_o$. Secondly, only observable actions are present in processes in $\mathbb{P}_o$. Unmatched synchronization actions are not present. Each failed synchronization leads to deadlock and it cannot be observed which synchronization action caused the deadlock. Instead the symbol $\delta$ is used to model deadlock.

**Example 4.3.14** *Consider a vending machine which requires a component that supplies coffee beans. This machine can be modeled by $s_c = c_{bean}; x_{coffee}$ where $c_{bean}$ in* Sync *and the procedure $x_{coffee}$ describes the further operation of the machine. A similar machine requiring tea leaves to serve tea can be described by $s_t = c_{leaf}; x_{tea}$ where $c_{leaf}$ in* Sync *and the procedure $x_{tea}$ describes the further operation of the machine.*

*Without the components that provide the beans and the leaves these machines behave the same: They do nothing. In the domain $\mathbb{P}_o$ this is modeled by $\delta$.*

Recall from the introduction that two different interpretations of nondeterminism are considered: a local interpretation and a global interpretation. For probabilistic choice only the unconditional interpretation is used. Therefore, two operational models are given on the operational domain $\mathbb{P}_o$.

**Example 4.3.15** *Consider the following part of a vending machine $s_{ct}$ that uses the two components $s_c$ and $s_t$ of the previous example*

$$s_{ct} = coin; ((c_{bean}; x_{coffee} \,\square\, c_{leaf}; x_{tea}) \,\square\, coin')$$

*where $coin'$ is an action in OAct modeling a refund. If the nondeterministic choice is local, then this machine will deadlock if coffee is selected when there are no coffee beans available. With a global interpretation of nondeterminism the machine will allow the selection of tea or a refund instead.*

The operational models $\mathcal{O}_l$ and $\mathcal{O}_g$ give the meanings of statements using the local and global interpretation of nondeterminism respectively.

**Definition 4.3.16** *The operational models $\mathcal{O}_i : Res \to \mathbb{P}_o$ and the auxiliary functions $\widehat{\mathcal{O}_i} : Stat_{res} \to \mathbb{Q}_o$ are given by*

$$\begin{array}{rcl}
\mathcal{O}_i(\mathrm{E}) & = & p_\epsilon \\
\mathcal{O}_i(s) & = & \{\!|\, \rho \cdot \widehat{\mathcal{O}_i}(s') \mid s \overset{\rho}{\rightsquigarrow} s' \,|\!\} \\
\widehat{\mathcal{O}_l}(s) & = & \{\, \langle b, \mathcal{O}_l(r) \rangle \mid s \xrightarrow{b} r \,\} \cup \{\, \delta \mid s \xrightarrow{c} r \,\} \\
\widehat{\mathcal{O}_g}(s) & = & \left\{ \begin{array}{ll} \{\, \langle b, \mathcal{O}_g(r) \rangle \mid s \xrightarrow{b} r \,\} & \textit{if } \exists b, r : s \xrightarrow{b} r \\ \{\, \delta \,\} & \textit{otherwise} \end{array} \right.
\end{array}$$

*where $i = l$ for the local interpretation of nondeterminism or $i = g$ for the global interpretation of nondeterminism.*

The operational models $\mathcal{O}_l$ and $\mathcal{O}_g$ differ in the way they deal with potential deadlock. In the local model each unmatched synchronization action will indeed lead to deadlock. In the global model, the synchronization will only be selected if it is successful. Deadlock is only obtained if all alternatives fail.

**Example 4.3.17** *Let $D(x) = (\bar{c} \oplus_{\frac{1}{2}} b); x$ then*

$$\begin{array}{rcl}
\mathcal{O}_l(x) & = & \{\!|\, \frac{1}{2} \cdot \widehat{\mathcal{O}_l}(\bar{c}; x), \frac{1}{2} \cdot \widehat{\mathcal{O}_l}(b; x) \,|\!\} \\
& = & \{\!|\, \frac{1}{2} \cdot \{\, \delta \,\}, \frac{1}{2} \cdot \{\, \langle b, \mathcal{O}_l(x) \rangle \,\} \,|\!\}
\end{array}$$

*A tree representation of the unique process in $\mathbb{P}_o$ that satisfies this equation is given in example 4.3.13.*

*Let $s = (c_{bean}; x_{coffee} \Box c_{leaf}; x_{tea}) \Box coin'$ then*

$$\begin{aligned}
\mathcal{O}_l(s \| \bar{c}_{bean}) &= \{\!\mid 1 \cdot \widehat{\mathcal{O}_l}(s \| \bar{c}_{bean}) \mid\!\} \\
&= \{\!\mid 1 \cdot \{ \langle \tau, \mathcal{O}_l(x_{coffee}) \rangle, \delta, \langle coin', \{\!\mid 1 \cdot \{ \delta \} \mid\!\} \rangle \} \mid\!\} \\
\mathcal{O}_g(s \| \bar{c}_{bean}) &= \{\!\mid 1 \cdot \widehat{\mathcal{O}_g}(s \| \bar{c}_{bean}) \mid\!\} \\
&= \{\!\mid 1 \cdot \{ \langle \tau, \mathcal{O}_g(x_{coffee}) \rangle, \langle coin', \{\!\mid 1 \cdot \{ \delta \} \mid\!\} \rangle \} \mid\!\}
\end{aligned}$$

*With a local interpretation of nondeterminism, deadlock is possible. Deadlock cannot occur (in the first step) with the global interpretation of nondeterminism as there are other options available.*

The operational semantics for $\mathcal{L}_{pnd}$ should be given for programs. The functions $\mathcal{O}_i$, however, give the meaning of resumptions. To remove this small discrepancy we define $\mathcal{O}[\![\bullet]\!]$.

**Definition 4.3.18** *The operational semantics $\mathcal{O}_i[\![\bullet]\!] : \mathcal{L}_{pnd} \to \mathbb{P}_o$ for $\mathcal{L}_{pnd}$ is given by $\mathcal{O}_i[\![s]\!] = \mathcal{O}_i(s)$, where $i = l$ for the local interpretation of nondeterminism or $i = g$ for the global interpretation of nondeterminism.*

By restricting to statements instead of resumptions the operational semantics $\mathcal{O}_i[\![\bullet]\!]$ is obtained from the operational model $\mathcal{O}_i$.

**Example 4.3.19** *The statements $s_t$ and $s_c$ given by*

$$\begin{aligned}
s_t &= (c_{leaf}; x_{tea}) \Box coin' \\
s_c &= (c_{bean}; x_{coffee}) \Box coin'
\end{aligned}$$

*are equivalent, $\mathcal{O}_i(s_t) = \mathcal{O}_i(s_c)$. The vending machines described by these statements can be interchanged; they offer the same services. (In this case, only a refund as no coffee beans or tea leaves are available.)*

*The statement $s_0 = coin'$ is equivalent to $s_c$ using the global interpretation of nondeterminism but not using the local interpretation, $\mathcal{O}_g(s_c) = \mathcal{O}_g(s_0)$ and $\mathcal{O}_l(s_c) \neq \mathcal{O}_l(s_0)$. Using the global interpretation both statements describe a machine that can only give a refund. Using the local interpretation, the machine $s_c$ will deadlock when coffee is selected, no longer offering a refund.*

## 4.3.4 Denotational semantics

As with the models with priority for nondeterminism a model based on the compositionality principle is useful for simplifying the task of checking properties of a program. The denotational semantics, introduced below, gives the meaning of statements in a compositional way. The meaning of a statement is given as an element of the domain of denotational meanings, denoted by $\mathbb{P}_d$, and several semantical operations are introduced to compose these meanings.

The operational semantics is not compositional as can be seen from example 4.3.20. This means that the operational behavior does not contain sufficient information to be able to compose meanings. For the denotational model extra information about a statement is maintained.

**Example 4.3.20** *Recall the vending machine components $s_c = c_{bean}; s$ and $s_t = c_{leaf}; s'$ from example 4.3.14 requiring components that supply coffee beans and tea leaves respectively. Without these components the two machine behave the same: They do nothing, $\mathcal{O}_i[\![s_c]\!] = \{ 1 \cdot \{ \delta \} \} = \mathcal{O}_i[\![s_t]\!]$. If, however, a component that supplies coffee beans (but not tea leaves) is added, the two machines behave differently: $\mathcal{O}_i[\![s_c \| \bar{c}_{bean}]\!] \neq \{ 1 \cdot \{ \delta \} \} = \mathcal{O}_i[\![s_t \| \bar{c}_{bean}]\!]$.*

*Two operationally equivalent machines, i.e. two machines with the same operational behavior, can be interchanged as a whole, but to replace a single component of a machine with another component, the two components should have the same denotational semantics.*

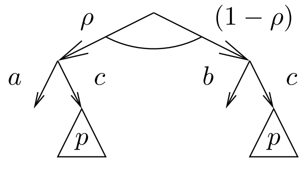
In the denotational meaning, unmatched synchronization actions are visible and do not result in deadlock. The denotational domain which contains all denotational meanings is given in the following definition.

**Definition 4.3.21** *The denotational domain $\mathbb{P}_d$ is given as the unique solution of the following domain equations*

$$\begin{array}{rcl}
\mathbb{P}_d & \simeq & \mathcal{MP}_f([0,1] \times \mathbb{Q}_d) \\
\mathbb{Q}_d & \simeq & \mathcal{P}_{nco}(\mathbb{R}_d) \\
\mathbb{R}_d & \simeq & Act + Act \times id_{\frac{1}{2}}(\mathbb{P}_d)
\end{array}$$

Again the existence of a unique solution is guaranteed by lemma 2.2.10. The elements of $\mathbb{P}_d$ are called (denotational) processes and the elements of $\mathbb{Q}_d$ are called nondeterministic subprocesses or simply subprocesses. Recall that multisets are used to model probabilistic choices in a branching domain. A process in $\mathbb{P}_d$ gives a (finite) number of possible subprocesses, each with the probability that the subprocess is selected. A subprocess in $\mathbb{Q}_d$ gives a set consisting of the actions that are possible. After an action the execution can be finished or continue with another process in $\mathbb{P}_d$. As a denotational meaning may still be composed with another process, it is not possible to see if a synchronization action will fail or not. Therefore, in a denotational process, unmatched synchronization actions are not interpreted as deadlock but are included as part of the outcome.

**Example 4.3.22** *The process $\{\!| \rho \cdot \{ a, \langle c, p \rangle \}, (1-\rho) \cdot \{ b, \langle c, p \rangle \} |\!\}$ contains two subprocesses. The first subprocess is selected with probability $\rho$ and will offer action $a$ and action $c$. With probability $1-\rho$ the second subprocess is selected and $b$ and $c$ are the available alternatives. If $c$ is selected in either of these subprocesses, this action is followed by another process $p$.*



*Note the similarity between the tree representation of a denotational process and the abstract transition trees as in e.g. example 4.3.4.*

For each of the syntactical constructs $\Box$, $\oplus_\rho$, $;$ and $\|$, a semantical operation on processes is defined. The semantical operation specifies how processes should be combined to obtain the meaning of a statement built with the corresponding syntactical operator.

To have a single notation for all elements of $\mathbb{R}_d$ an action $a$ is identified with $\langle a, p_\epsilon \rangle$ and the (meta-)variables $\hat{p}$ and $\hat{p}'$ are used to range over $\mathbb{P}_d + \{ p_\epsilon \}$. Using this convention, $\langle a, \hat{p} \rangle$ ranges over $\mathbb{R}_d$.

**Definition 4.3.23** *All denotational operations are elements of $Op = \mathbb{P}_d \times \mathbb{P}_d \xrightarrow{1} \mathbb{P}_d$, i.e. they are nonexpansive functions that take a pair of processes and yield a single process.*

*(a) The operator $\oplus_\rho \in Op$ is defined by*

$$p_1 \oplus_\rho p_2 \quad = \quad \rho p_1 \sqcup (1-\rho) p_2$$

*where $\rho p$ is equal to $\{\!|\, \rho\sigma \cdot q \mid \sigma \cdot q \in p \,|\!\}$.*

*(b) The operator $\Box \in Op$ is defined by*

$$p_1 \Box p_2 \quad = \quad \{\!|\, \rho\sigma \cdot (q_1 \cup q_2) \mid \rho \cdot q_1 \in p_1, \sigma \cdot q_2 \in p_2 \,|\!\}$$

*(c) The operator $;\in Op$ and the auxiliary function $;' : \mathbb{Q}_d \times \mathbb{P}_d \to \mathbb{Q}_d$ are defined by*

$$\begin{array}{rcl} p_1 ; p_2 & = & \{\!|\, \rho \cdot q ;' p_2 \mid \rho \cdot q \in p_1 \,|\!\} \\ q ;' p & = & \{ \langle a, \hat{p} ; p \rangle \mid \langle a, \hat{p} \rangle \in q \} \end{array}$$

*where $\hat{p} ; p = p$ if $\hat{p} = p_\epsilon$.*

*(d) The operator $\| \in Op$ and the auxiliary functions $\|' : \mathbb{Q}_d \times \mathbb{Q}_d \to \mathbb{Q}_d$, $\Lsh\!\!\lfloor : \mathbb{Q}_d \times \mathbb{Q}_d \to \mathbb{Q}_d$, $\|'' : \mathbb{P}_d \times \mathbb{Q}_d \to \mathbb{P}_d$ and $| : \mathbb{Q}_d \times \mathbb{Q}_d \to (\mathbb{Q}_d \cup \{ \emptyset \})$ are defined by:*

$$\begin{array}{rcl} p_1 \| p_2 & = & \{\!|\, \rho\sigma \cdot (q_1 \|' q_2) \mid \rho \cdot q_1 \in p_1, \sigma \cdot q_2 \in p_2 \,|\!\} \\ q_1 \|' q_2 & = & q_1 \| \!\! \lfloor q_2 \cup q_2 \| \!\! \lfloor q_1 \cup q_1 | q_2 \\ q_1 \| \!\! \lfloor q_2 & = & \{ \langle a, \hat{p} \|'' q_2 \rangle \mid \langle a, \hat{p} \rangle \in q_1 \} \\ p_1 \|'' q_2 & = & \{\!|\, \rho \cdot (q_1 \|' q_2) \mid \rho \cdot q_1 \in p_1 \,|\!\} \\ q_1 | q_2 & = & \{ \langle \tau, \hat{p} \| \hat{p}' \rangle \rangle \mid \langle c, \hat{p} \rangle \in q_1, \langle \bar{c}, \hat{p}' \rangle \in q_2 \} \end{array}$$

*where $\hat{p} \|'' q = \{\!|\, 1 \cdot q \,|\!\}$ for $\hat{p} = p_\epsilon$ and also $\hat{p} \| \hat{p}' = \hat{p}' \| \hat{p} = \hat{p}'$ when $\hat{p} = p_\epsilon$.*

Except for the reversal of the levels of nondeterministic choice and probabilistic choice, this definition closely resembles definition 4.2.27 in the previous section. The recursive definitions can again be justified by showing that an operation *op* satisfies the equations in this definition exactly when the operation is a fixed point of a contractive higher order operation $\Omega_{op}$. Uniqueness of the fixed point of this higher order operation gives that there is exactly one function satisfying these equations.

In the process $p \oplus_\rho p'$, the process $p$ is chosen with probability $\rho$. A subprocess $q$ that is chosen with probability $\sigma$ in process $p$ is, therefore, chosen with probability $\rho\sigma$ in $p \oplus_\rho p'$. Similarly a subprocess $q$ that is chosen with probability $\sigma$ in process $p'$ is chosen with probability $(1-\rho)\sigma$ in $p \oplus_\rho p'$.

The subprocess $q \cup q'$ offers the actions offered by $q$ and the actions offered by $q'$. To obtain a probabilistic alternative of $p \,\square\, p'$ an alternative $q$ for $p$ and an alternative $q'$ for $p'$ are selected and combined to $q \cup q'$. The probability of this alternative is the probability that $q$ and $q'$ are selected, i.e. the product of the probability of $q_1$ and the probability of $q_2$.

Sequential composition is as usual. For the sequential composition the actions of the first process are taken until it is finished (the case $\hat{p} = p_\epsilon$) after which the second process is executed. (Recall the convention that $a$ is identified with $\langle a, p_\epsilon\rangle$.)

A probabilistic alternative of a parallel composition $p_1 \| p_2$ is obtained by combining an alternative $q_1$ of $p_1$ with an alternative $q_2$ of $p_2$ to give $q_1 \,\|'\, q_2$. The probability of the combined subprocess is the probability that $q_1$ and $q_2$ are selected. The first step of the combined subprocess $q_1 \,\|'\, q_2$ can come from the subprocess $q_1$ ($q_1 \,\|\!\!\_\, q_2$), from $q_2$ ($q_2 \,\|\!\!\_\, q_1$) or from the synchronization between these two subprocesses ($q_1 | q_2$). After the first step the remaining process is the parallel composition of what remains of $q_1$ and what remains of $q_2$. If the first action is produced by $q_1$ the remainder of $q_1$ will be some process $\hat{p}$ while the remainder of $q_2$ will be the subprocess $q_2$ itself. To give the parallel composition of a process and a subprocess, the operator $\|''$ is used. Note that $p \,\|''\, q = p \| \{\!|\, 1 \cdot q \,|\!\}$.

**Example 4.3.24** *Let $c_{bean}, \bar{c}_{bean}, x_{coffee}$ and coin$'$ be as before and let no_beans be an additional atomic action in OAct. The action no_beans models a light that burns to indicate that the component of the vending machine that supplies the coffee beans is empty.*

*The process $\{\!|\, 1 \cdot \{\, \bar{c}_{bean} \,\} \,|\!\}$ describes a component that is able to supply coffee beans. The process $\{\!|\, 1 \cdot \{\, \mathit{no\_beans} \,\} \,|\!\}$ describes a similar component that is empty. By using the operation $\oplus_\rho$ these two processes can be combined to a process describing a component that still has coffee beans with probability $\rho$*

$$\{\!|\, 1 \cdot \{\, \bar{c}_{bean} \,\} \,|\!\} \oplus_\rho \{\!|\, 1 \cdot \{\, \mathit{no\_beans} \,\} \,|\!\} \quad = \quad \{\!|\, \rho \cdot \{\, \bar{c}_{bean} \,\}, (1-\rho) \cdot \{\, \mathit{no\_beans} \,\} \,|\!\}$$

*The process $\{\!|\, 1 \cdot \{\, c_{bean} \,\} \,|\!\}$ that checks for coffee beans and the process $p_{coffee}$ that makes the coffee can be combined using the operation ; resulting in the process $\{\!|\, 1 \cdot \{\, \langle c_{bean}, p_{coffee}\rangle \,\} \,|\!\}$ that first checks for beans and then makes coffee. Note that the exact form of the process $p_{coffee}$ does not influence the way these processes are composed. The operator $\square$ can be used to allow a choice. For example the process $\{\!|\, 1 \cdot \{\, \langle c_{bean}, p_{coffee}\rangle \,\} \,|\!\} \,\square\, \{\!|\, 1 \cdot \{\, coin' \,\} \,|\!\} = \{\!|\, 1 \cdot \{\, \langle c_{bean}, p_{coffee}\rangle, coin' \,\} \,|\!\}$ offers the choice between coffee and a refund.*

*The operator $\|$ is used to compose separate components. We have that*

$$\begin{aligned} &\{\!|\, \rho \cdot \{\, \bar{c}_{bean} \,\}, (1-\rho) \cdot \{\, \mathit{no\_beans} \,\} \,|\!\} \,\|\, \{\!|\, 1 \cdot \{\, \langle c_{bean}, p_{coffee}\rangle, coin' \,\} \,|\!\} \\ &\quad = \quad \{\!|\, \rho \cdot (\{\, \bar{c}_{bean} \,\} \,\|'\, \{\, \langle c_{bean}, p_{coffee}\rangle, coin' \,\}), \\ &\qquad\qquad (1-\rho) \cdot (\{\, \mathit{no\_beans} \,\} \,\|'\, \{\, \langle c_{bean}, p_{coffee}\rangle, coin' \,\}) \,|\!\} \end{aligned}$$

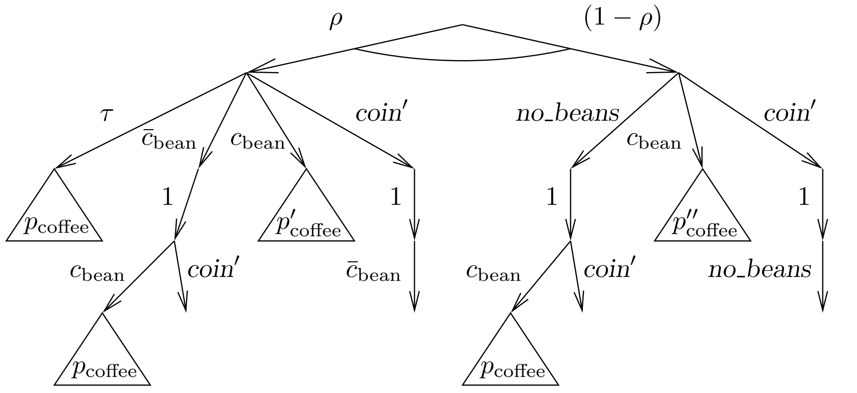
$$\begin{aligned}
= \quad & \{\!|\, \rho \cdot \{ \langle \bar{c}_{bean}, \{\!|\, 1 \cdot \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \,|\!\} \rangle, \langle c_{bean}, p_{coffee} \parallel'' \{ \bar{c}_{bean} \} \rangle, \\
& \qquad \langle coin', \{\!|\, 1 \cdot \{ \bar{c}_{bean} \} \,|\!\} \rangle, \langle \tau, p_{coffee} \rangle \}, \\
& (1-\rho) \cdot \{ \langle no\_beans, \{\!|\, 1 \cdot \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \,|\!\} \rangle, \\
& \qquad \langle c_{bean}, p_{coffee} \parallel'' \{ no\_beans \} \rangle, \langle coin', \{\!|\, 1 \cdot \{ no\_beans \} \,|\!\} \rangle \} \,|\!\}
\end{aligned}$$

*because*

$$\begin{aligned}
& \{ \bar{c}_{bean} \} \parallel' \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \\
& \quad = \{ \bar{c}_{bean} \} \mathbin{|\!\lfloor} \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \cup \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \mathbin{|\!\lfloor} \{ \bar{c}_{bean} \} \\
& \qquad \cup \{ \bar{c}_{bean} \} \mid \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \\
& \quad = \{ \left\langle \bar{c}_{bean}, \{\!|\, 1 \cdot \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \,|\!\} \right\rangle, \left\langle c_{bean}, p_{coffee} \parallel'' \{ \bar{c}_{bean} \} \right\rangle, \\
& \qquad \left\langle coin', \{\!|\, 1 \cdot \{ \bar{c}_{bean} \} \,|\!\} \right\rangle, \left\langle \tau, p_{coffee} \right\rangle \}
\end{aligned}$$

*and*

$$\begin{aligned}
& \{ no\_beans \} \parallel' \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \\
& \quad = \{ no\_beans \} \mathbin{|\!\lfloor} \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \\
& \qquad \cup \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \mathbin{|\!\lfloor} \{ no\_beans \} \\
& \qquad \cup \{ no\_beans \} \mid \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \\
& \quad = \{ \langle no\_beans, \{\!|\, 1 \cdot \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \,|\!\} \rangle, \\
& \qquad \langle c_{bean}, p_{coffee} \parallel'' \{ no\_beans \} \rangle, \langle coin', \{\!|\, 1 \cdot \{ no\_beans \} \,|\!\} \rangle \}
\end{aligned}$$



with $p'_{\text{coffee}} = p_{\text{coffee}} \parallel'' \{ \bar{c}_{\text{bean}} \}$ and $p''_{\text{coffee}} = p_{\text{coffee}} \parallel'' \{ no\_beans \}$

Denotational process describing the vending machine

*Note that in this composition the possibility is left open that other components may be added which supply or use coffee beans.*

The operations introduced above are nonexpansive. For the operation '$;$' this can be strengthened. The operator '$;$' is nonexpansive in its first component and contractive in its second, i.e.

$$d(p_1 \,;p_2, p_1' \,; p_2') \leq \max\{ d(p_1, p_1'), \tfrac{1}{2} d(p_2, p_2') \}$$

The proof of this fact is again a straightforward extension of results in chapter 3 (lemma 3.4.11) and known results (cf. [38]). The justification of the following definition of the denotational model $\mathcal{D}$ below can be based on the usual fixed point method (see e.g. lemma 3.4.13). Contractiveness of ; in its second argument is required for contractiveness of the higher-order mapping used in this method.

The definition of the denotational semantics uses the compositionality principle. The meaning of a basic statement consisting of a single action is given. For a statement built with a certain syntactical operation the corresponding semantical operation is used to find the meaning of the statement from the meanings of the parts of the statement.

**Definition 4.3.25** *The denotational model $\mathcal{D}: \mathcal{L}_{pnd} \to \mathbb{P}_d$ is given by*

$$\begin{array}{rcl} \mathcal{D}(a) & = & \{\!|\, 1 \cdot \{ a \} \,|\!\} \\ \mathcal{D}(x) & = & \mathcal{D}(D(x)) \\ \mathcal{D}(s_1 \ op \ s_2) & = & \mathcal{D}(s_1) \ op \ \mathcal{D}(s_2) \end{array}$$

*where op is $\Box$, $\oplus_\rho$, ; or $\|$.*

A single action $a$ acts like $a$ with probability one. Recursion is handled by body replacement and to give the meaning of any statement built using the syntactic operator *op* the corresponding denotational operation *op* is used.

**Example 4.3.26** *The meaning $\mathcal{D}(coin)$ of the atomic action coin is given by $\mathcal{D}(coin) = \{\!|\, 1 \cdot \{ coin \} \,|\!\}$. The processes obtained for atomic actions can be composed as in example 4.3.24 to obtain*

$$\begin{array}{rcl} \mathcal{D}(\bar{c}_{bean} \oplus_\rho no\_beans) & = & \{\!|\, \rho \cdot \{ \bar{c}_{bean} \}, (1-\rho) \cdot \{ no\_beans \} \,|\!\} \\ \mathcal{D}(c_{bean}; x_{coffee} \Box coin') & = & \{\!|\, 1 \cdot \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \,|\!\} \end{array}$$

$$\begin{array}{l} \mathcal{D}((\bar{c}_{bean} \oplus_\rho no\_beans) \,\|\, (c_{bean}; x_{coffee} \Box coin')) \ = \\ \qquad \{\!|\, \rho \cdot \{ \langle \bar{c}_{bean}, \{\!|\, 1 \cdot \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \,|\!\} \rangle, \langle c_{bean}, p_{coffee} \,\|'' \{ \bar{c}_{bean} \} \rangle, \\ \qquad\qquad \langle coin', \{\!|\, 1 \cdot \{ \bar{c}_{bean} \} \,|\!\} \rangle, \langle \tau, p_{coffee} \rangle \}, \\ \qquad\quad (1-\rho) \cdot \{ \langle no\_beans, \{\!|\, 1 \cdot \{ \langle c_{bean}, p_{coffee} \rangle, coin' \} \,|\!\} \rangle, \\ \qquad\qquad\qquad \langle c_{bean}, p_{coffee} \,\|'' \{ no\_beans \} \rangle, \langle coin', \{\!|\, 1 \cdot \{ no\_beans \} \,|\!\} \rangle \} \,|\!\} \end{array}$$

*where $p_{coffee} = \mathcal{D}(x_{coffee})$.*

To obtain the operational semantics $\mathcal{O}[\![\bullet]\!]$ from the operational model $\mathcal{O}$ a restriction to programs was required. As the denotational model $\mathcal{D}$ is already defined on programs, this restriction is not required here. We separately defined the denotational semantics $\mathcal{D}[\![\bullet]\!]$, which coincides with the model $\mathcal{D}$, only to maintain symmetry with the definition of the operational semantics and with definitions of denotational semantics in later chapters.

**Definition 4.3.27** *The denotational semantics $\mathcal{D}[\![\bullet]\!]: \mathcal{L}_{pnd} \to \mathbb{P}_d$ is given by $\mathcal{D}[\![s]\!] = \mathcal{D}(s)$.*

Having given two operational models and a denotational model, a natural question to ask is how the models are related. In the following subsection, the operational models are compared with the denotational model.

### 4.3.5 Comparing the operational and denotational semantics

The denotational semantics contains more information than the operational semantics. As explained above, this information is needed to achieve compositionality. The unmatched synchronization actions are still present in the denotational processes. In this subsection it is shown that both operational models are abstractions from the denotational model. The operational meaning can be obtained from the denotational meaning by removing the extra information.

A similar approach to that of section 4.2 is used to show that the operational models are abstractions of the denotational model. An operational-like model $\mathcal{O}^*$ yielding denotational processes is introduced and shown to coincide with the denotational model $\mathcal{D}$. Next two abstraction functions are introduced and both operational models are shown to be abstractions of the operational-like model $\mathcal{O}^*$. The nondeterminism in processes in the denotational domain, can still be interpreted as local or global. Different abstraction functions are needed for the local and global interpretations of nondeterministic choice. Both the operational domain $\mathbb{P}_o$ and the denotational domain $\mathbb{P}_d$ are branching domains, allowing for relatively simple abstraction functions (compared to the abstraction functions $abs_i$ from section 4.2) from the denotational to the operational domain. An abstraction which removes the branching information from processes in $\mathbb{P}_o$ is given in subsection 4.3.6 below. The following graph shows the steps involved in the comparison of the operational and denotational semantics.

$$\begin{tikzcd}
\mathbb{P}_d + \{\, p_\epsilon \,\} & \mathbb{P}_b \\
 & \mathcal{O}_l \\
\mathcal{D} = \mathcal{O}^* \arrow[ur, "abs_l"] \arrow[dr, "abs_g"'] & \\
 & \mathcal{O}_g
\end{tikzcd}$$

To be able to assign a meaning to all resumptions, and not only to statements, the denotational domain $\mathbb{P}_d$ is extended with the empty process giving $\mathbb{P}_d + \{\, p_\epsilon \,\}$. The denotational model is extended to resumptions by assigning the empty process as the meaning of the empty resumption, $\mathcal{D}(\mathrm{E}) = p_\epsilon$.

**Definition 4.3.28** *The model $\mathcal{D} : Res \to \mathbb{P}_d + \{\, p_\epsilon \,\}$ is given by*

$$\begin{array}{rcl}
\mathcal{D}(\mathrm{E}) & = & p_\epsilon \\
\mathcal{D}(a) & = & \{\!|\, 1 \cdot \{\, a \,\} \,|\!\} \\
\mathcal{D}(x) & = & \mathcal{D}(D(x)) \\
\mathcal{D}(s_1 \, op \, s_2) & = & \mathcal{D}(s_1) \, op \, \mathcal{D}(s_2)
\end{array}$$

*for $op \in \{\, ; , \oplus_\rho, \Box, \| \,\}$.*

Next the operational-like model $\mathcal{O}^*$ with values in the extended denotational domain is given.

**Definition 4.3.29** *The operational-like model $\mathcal{O}^* : \mathcal{L}^+_{pnd} \to \mathbb{P}_d + \{\, p_\epsilon \,\}$ and the auxiliary function $\widehat{\mathcal{O}}^* : Stat_{res} \to \mathbb{Q}_d$ are given by*

$$\begin{array}{rcl}
\mathcal{O}^*(\mathrm{E}) & = & p_\epsilon \\
\mathcal{O}^*(s) & = & \{\!|\, \rho \cdot \widehat{\mathcal{O}}^*(s') \mid s \overset{\rho}{\rightsquigarrow} s' \,|\!\} \\
\widehat{\mathcal{O}}^*(s) & = & \{\, \langle a, \mathcal{O}^*(r) \rangle \mid s \xrightarrow{a} r \,\}
\end{array}$$

*where $\langle a, p_\epsilon \rangle$ is identified with $a$.*

As usual this definition can be justified by showing that $\mathcal{O}^*$ is the unique fixed point of a higher-order operator. The higher-order operator $\Phi^*$ is assumed to be clear.

**Example 4.3.30** *Let, as in previous examples (see e.g. 4.3.26), $\bar{c}_{bean}$ be a synchronization action in Sync and no_beans an observable action in OAct.*

$$\begin{array}{rcl}
\mathcal{O}^*(\bar{c}_{bean} \oplus_\rho no\_beans) & = & \{\!|\, \rho \cdot \widehat{\mathcal{O}}^*(\bar{c}_{bean}), (1-\rho) \cdot \widehat{\mathcal{O}}^*(no\_beans) \,|\!\} \\
& = & \{\!|\, \rho \cdot \{\, \bar{c}_{bean} \,\}, (1-\rho) \cdot \{\, no\_beans \,\} \,|\!\}
\end{array}$$

Although the process that is assigned to the statement $\bar{c}_{\mathrm{bean}} \oplus_\rho no\_beans$ is calculated in a different way, the operational-like model $\mathcal{O}^*$ yields the same process as $\mathcal{D}$ (cf. example 4.3.26). This is the case for all statements.

**Lemma 4.3.31** *The operational-like model $\mathcal{O}^*$ and the denotational model $\mathcal{D}$ coincide.*

**Proof** The denotational model $\mathcal{D}$ is shown to be a fixed point of the contractive higher-order operator $\Phi^*$ that can be used to justify the definition of $\mathcal{O}^*$. As $\mathcal{O}^*$ is the unique fixed point of this operator, the two models coincide. That $\Phi^*(\mathcal{D})(r) = \mathcal{D}(r)$ is shown by induction on the weight of the resumption $r$. A few cases are given below.

- For the basic resumptions E and $a$, we have

  $$\Phi^*(\mathcal{D})(\mathrm{E}) \;=\; p_\epsilon \;=\; \mathcal{D}(\mathrm{E})$$

  and

  $$\Phi^*(\mathcal{D})(a) \;=\; \{\!|\, 1 \cdot \{\, a \,\} \,|\!\} \;=\; \mathcal{D}(a)$$

- The axioms (Chance 1) and (Chance 2) give the transitions $s \oplus_\rho s' \xrightarrow{\rho} s$ and $s \oplus_\rho s' \xrightarrow{1-\rho} s'$. Inspection of the transition system $\mathcal{T}^{(2)}_{pnd}$ shows that these are the only transitions for $s \oplus_\rho s'$. This means that $s \oplus_\rho s' \overset{\sigma}{\rightsquigarrow} s''$ exactly when $s \overset{\sigma'}{\rightsquigarrow} s''$ and $\sigma = \rho\sigma'$ or $s' \overset{\sigma'}{\rightsquigarrow} s''$ and $\sigma = (1-\rho)\sigma'$. Using this gives

$$\begin{array}{rcll}
\Phi^*(\mathcal{D})(s \oplus_\rho s') & = & [\text{def. } \Phi^*] & \{\!|\, \sigma \cdot \widehat{\Phi}^*(\mathcal{D})(s'') \mid (s \oplus_\rho s') \overset{\sigma}{\rightsquigarrow} s'' \,|\!\} \\
& = & [\text{def. } \overset{\rho}{\rightsquigarrow}] & \{\!|\, \rho\sigma' \cdot \widehat{\Phi}^*(\mathcal{D})(s'') \mid s \overset{\sigma'}{\rightsquigarrow} s'' \,|\!\} \\
& & & \sqcup \{\!|\, (1-\rho)\sigma' \cdot \widehat{\Phi}^*(\mathcal{D})(s'') \mid s' \overset{\sigma'}{\rightsquigarrow} s'' \,|\!\} \\
& = & [\text{def. } \Phi^*] & \Phi^*(\mathcal{D})(s) \oplus_\rho \Phi^*(\mathcal{D})(s') \\
& = & [\text{ind. hyp.}] & \mathcal{D}(s) \oplus_\rho \mathcal{D}(s') \\
& = & [\text{def. } \mathcal{D}] & \mathcal{D}(s \oplus_\rho s')
\end{array}$$

- When $s \overset{\rho}{\rightsquigarrow} s''$, and $s' \overset{\sigma}{\rightsquigarrow} s'''$, then repeatedly using rule (Choice 1) followed by repeatedly using rule (Choice 2) gives that $s \Box s' \overset{\rho\sigma}{\rightsquigarrow} s'' \Box s'''$. Inspection of $\mathcal{T}^{(2)}_{pnd}$ shows that all probabilistic alternatives of $s \Box s'$ are obtained in this way.

  For resolved statements $s''$ and $s'''$ we have that

$$\begin{array}{rcl}
\widehat{\Phi}^*(\mathcal{D})(s'' \Box s''') & = & \{\, \langle a, \mathcal{D}(r) \rangle \mid s'' \Box s''' \xrightarrow{a} r \,\} \\
& = & \{\, \langle a, \mathcal{D}(r) \rangle \mid s'' \xrightarrow{a} r \vee s'' \xrightarrow{a} r \,\} \\
& = & \{\, \langle a, \mathcal{D}(r) \rangle \mid s'' \xrightarrow{a} r \,\} \cup \{\, \langle a, \mathcal{D}(r) \rangle \mid s''' \xrightarrow{a} r \,\} \\
& = & \widehat{\Phi}^*(\mathcal{D})(s'') \cup \widehat{\Phi}^*(\mathcal{D})(s''')
\end{array}$$

Using these observations gives

$$\begin{array}{lll}
\Phi^*(\mathcal{D})(s \Box s') & & \\
\quad = [\text{def. } \Phi^*] & \{\!\!|\, \rho \cdot \widehat{\Phi}^*(\mathcal{D})(s'') \mid (s \Box s') \overset{\rho}{\rightsquigarrow} s'' \,|\!\!\} \\
\quad = [\text{def. } \overset{\rho}{\rightsquigarrow}] & \{\!\!|\, \rho\sigma \cdot \widehat{\Phi}^*(\mathcal{D})(s'' \Box s''') \mid s \overset{\rho}{\rightsquigarrow} s'', s' \overset{\sigma}{\rightsquigarrow} s''' \,|\!\!\} \\
\quad = & \{\!\!|\, \rho\sigma \cdot (\widehat{\Phi}^*(\mathcal{D})(s'') \cup \widehat{\Phi}^*(\mathcal{D})(s''')) \mid s \overset{\rho}{\rightsquigarrow} s'', s' \overset{\sigma}{\rightsquigarrow} s''' \,|\!\!\} \\
\quad = [\text{def. } \Box] & \{\!\!|\, \rho \cdot \widehat{\Phi}^*(\mathcal{D})(s'') \mid s \overset{\rho}{\rightsquigarrow} s'' \,|\!\!\} \Box \{\!\!|\, \sigma \cdot \widehat{\Phi}^*(\mathcal{D})(s''') \mid s' \overset{\sigma}{\rightsquigarrow} s''' \,|\!\!\} \\
\quad = [\text{def. } \Phi^*] & \Phi^*(\mathcal{D})(s) \Box \Phi^*(\mathcal{D})(s') \\
\quad = [\text{ind. hyp.}] & \mathcal{D}(s) \Box \mathcal{D}(s') \\
\quad = [\text{def. } \mathcal{D}] & \mathcal{D}(s \Box s')
\end{array}$$

$\Box$

The lemma above uses the usual fixed point approach to show that the models $\mathcal{O}^*$ and $\mathcal{D}$ are the same. The following abstraction functions $abs_l$ and $abs_g$ can be used to obtain the local and the global operational meaning from the processes given by $\mathcal{O}^*$. Depending on the interpretation of nondeterminism, the unmatched synchronization actions, present in a denotational process but not in an operational process, are replaced by deadlock or simply removed.

**Definition 4.3.32** *The abstraction functions* $abs_i : \mathbb{P}_d + \{\, p_\epsilon \,\} \rightarrow \mathbb{P}_o$ *and the auxiliary functions* $abs_i' : \mathbb{Q}_d \rightarrow \mathbb{Q}_o$ *are given by*

$$\begin{array}{rcl}
abs_i(p_\epsilon) & = & p_\epsilon \\
abs_i(p) & = & \{\!\!|\, \rho \cdot abs_i'(q) \mid \rho \cdot q \in p \,|\!\!\} \\
abs_l'(q) & = & \{\, \langle b, abs_l(\hat{p}) \rangle \mid \langle b, \hat{p} \rangle \in q \,\} \cup \{\, \delta \mid \langle c, \hat{p} \rangle \in q \,\} \\
abs_g'(q) & = & \left\{ \begin{array}{ll} \{\, \langle b, abs_g(\hat{p}) \rangle \mid \langle b, \hat{p} \rangle \in q \,\} & \text{if } \exists \langle b, \hat{p} \rangle \in q \\ \{\, \delta \,\} & \text{otherwise} \end{array} \right.
\end{array}$$

*where $i$ is $l$ for the local interpretation of nondeterminism or $g$ for the global interpretation of nondeterminism. Recall the convention that $\hat{p}$ ranges over $\mathbb{P}_d + \{\, p_\epsilon \,\}$, that $b$ ranges over observable actions only and that $\langle b, p_\epsilon \rangle$ is identified with $b$.*

The abstraction functions $abs_l$ and $abs_g$ from the denotational to the operational domain remove unmatched synchronization actions and introduce deadlock when needed. The abstraction for the local interpretation of nondeterminism replaces each unmatched synchronization action by deadlock. The abstraction for the global interpretation of nondeterminism removes each unmatched synchronization action and only introduces deadlock if no other options remain.

The following lemma relates the operational models $\mathcal{O}_l$ and $\mathcal{O}_g$ with the model $\mathcal{O}^*$. This lemma can be proven using the usual fixed point reasoning (see e.g. lemma 3.4.13).

**Lemma 4.3.33** *The operational models $\mathcal{O}_l$ and $\mathcal{O}_g$ are abstractions of the operational-like model $\mathcal{O}^*$*

$$\mathcal{O}_i = abs_i \circ \mathcal{O}^*$$

*where i is l for the local interpretation of nondeterminism or g for the global interpretation of nondeterminism.*

As the operational models $\mathcal{O}_l$ and $\mathcal{O}_g$ are abstractions from the operational-like model $\mathcal{O}^*$ they identify all the statements identified by $\mathcal{O}^*$. The reverse does not hold, $\mathcal{O}_l$ and $\mathcal{O}_g$ identify strictly more statements than $\mathcal{O}^*$.

**Example 4.3.34** *The programs $c_{bean}$ and $c_{leaf}$ are identified by $\mathcal{O}_l$ and $\mathcal{O}_g$ but not by $\mathcal{O}^*$*

$$\begin{array}{lclclcl} \mathcal{O}_i(c_{bean}) & = & \{\!|\, 1 \cdot \{\, \delta \,\} \,|\!\} & & = & \mathcal{O}_i(c_{leaf}) & \\ \mathcal{O}^*(c_{bean}) & = & \{\!|\, 1 \cdot \{\, c_{bean} \,\} \,|\!\} & \neq & \{\!|\, 1 \cdot \{\, c_{leaf} \,\} \,|\!\} & = & \mathcal{O}^*(c_{leaf}) \end{array}$$

Combining the results in this subsection gives that both operational models are abstractions of the denotational model.

**Theorem 4.3.35** *The operational semantics $\mathcal{O}_i[\![\bullet]\!]$ are abstractions of the denotational semantics $\mathcal{D}$, $\mathcal{O}_i[\![\bullet]\!] = abs_i \circ \mathcal{D}[\![\bullet]\!]$, where i is l for the local or g for the global interpretation of nondeterminism.*

**Proof**

$$\begin{array}{rcll} \mathcal{O}_i[\![s]\!] & = & \text{[lemma 4.3.18]} & \mathcal{O}(s) \\ & = & \text{[lemma 4.3.33]} & abs_i(\mathcal{O}^*(s)) \\ & = & \text{[lemma 4.3.31]} & abs_i(\mathcal{D}(s)) \\ & = & \text{[definition 4.3.27]} & abs_i(\mathcal{D}[\![s]\!]) \end{array}$$ □

The operational models $\mathcal{O}_l$ replaces unmatched synchronization actions by deadlock. By removing the deadlock from the processes given by $\mathcal{O}_l$ if other options exists, one obtains the processes given by $\mathcal{O}_g$. The reverse is not possible, $\mathcal{O}_g$ identifies strictly more statements than $\mathcal{O}_l$.

**Example 4.3.36** *The programs $c_{bean} \,\Box\, coin'$ and $coin'$ are identified by $\mathcal{O}_g$ but not by $\mathcal{O}_l$.*

$$\begin{array}{lclclcl} \mathcal{O}_l(c_{bean} \,\Box\, coin') & = & \{\!|\, 1 \cdot \{\, coin', \delta \,\} \,|\!\} & \neq & \{\!|\, 1 \cdot \{\, coin' \,\} \,|\!\} & = & \mathcal{O}_l(coin') \\ \mathcal{O}_g(c_{bean} \,\Box\, coin') & = & \{\!|\, 1 \cdot \{\, coin' \,\} \,|\!\} & = & \mathcal{O}_g(coin') & & \end{array}$$

The following graph summarizes the results of this subsection.

$$\begin{tikzcd}
\mathbb{P}_d + \{ p_\epsilon \} & & \mathbb{P}_o & \\
 & & \mathcal{O}_l \arrow[dll, shift left, "|" marking] \arrow[dd, shift right] & \\
\mathcal{D} = \mathcal{O}^* \arrow[urr, shift left, "abs_l"] \arrow[drr, shift right, "abs_g"'] & \text{ex. 4.3.34} & & \text{ex. 4.3.36} \\
 & & \mathcal{O}_g \arrow[ull, shift right, "|" marking] \arrow[uu, shift right, "|" marking] &
\end{tikzcd}$$

## 4.3.6 Linearizing the operational domain $\mathbb{P}_o$

In subsection 4.3.3 it was argued that a branching domain is more appropriate for the operational domain if it is visible which actions, i.e. which resources, are available at any point. If, however, the user cannot check the resources which are available at a given time, for example because the selection of the available resources is not entirely up to the user, a linear model is useful. In this subsection it is shown how a linear process can be obtained from the branching processes in $\mathbb{P}_o$. The domain of linear processes is called $\mathbb{P}_L$.

**Definition 4.3.37** *The linear operational domain $\mathbb{P}_L$ is given by*

$$\begin{array}{rcl}
\mathbb{P}_L & = & \mathit{Meas}(\mathbb{Q}_L) \\
\mathbb{Q}_L & = & \mathcal{P}_{nco}(\mathbb{R}_L) \\
\mathbb{R}_L & = & \mathit{OAct}_\delta^\infty
\end{array}$$

*with* $\mathit{OAct}_\delta^\infty = \mathit{OAct}^\star + \mathit{OAct}^\star \cdot \{ \delta \} + \mathit{OAct}^\omega$.

A sequence in $\mathbb{R}_L$ gives the actions produced during a single run of the system. The sequence can terminate normally (sequences in $\mathit{OAct}^\star$), deadlock after executing a number of actions (sequences in $\mathit{OAct}^\star \cdot \{ \delta \}$), or not terminate at all (sequences in $\mathit{OAct}^\omega$).

A linear subprocess in $\mathbb{Q}_L$ offers a choice between possible runs of the system in $\mathbb{R}_L$. Compared to processes $\mathbb{Q}_o$, all nondeterministic choices are combined into a single nondeterministic choice at the start of the process.

The linear operational domain $\mathbb{P}_L$ combines all probabilistic choices in a process into a single choice at the start of the process. As the number of options in the resulting choice may be infinite, this choice cannot be modeled by a finite multiset of labeled subprocesses. A measure over subprocesses is used instead. Recall that a measure gives the probability for all observable events. An observable event is a Borel set over $\mathbb{Q}_o$, i.e. a set of subprocesses.

**Example 4.3.38** *The sequences $b$, $bb'$, $b\delta$ and $b^\omega$ are all elements of $\mathbb{R}_L$. The first sequence corresponds to the execution of a single $b$ followed by normal termination. The second sequence corresponds to the execution of the action $b$ followed by the execution of the action $b'$ followed by normal termination. The sequence $b\delta$ corresponds to the execution* *of the action $b$ followed by deadlock of the system. The sequence $b^\omega$ corresponds to forever repeating the execution of the action $b$.*

*The sets $q_1 = \{ b \}$, $q_2 = \{ b, b' \}$ and $q_3 = \{ b, bb, bbb, \dots, b^\omega \}$ are elements of $\mathbb{Q}_L$. The subprocess $q_1$ offers only a single action $b$, $q_2$ offers both a single $b$ and a single $b'$ and $q_3$ offers the choice between any (positive) number of $b$'s, including infinitely many $b$'s. Note that the set $\{ b, bb, bbb, \dots \}$ without the possibility for infinitely many actions $b$ is not in $\mathbb{Q}_L$. For closedness and hence for compactness of the set, the possibility of infinitely many actions $b$ must be included.*

*The set $q_4 = \{ b, b' \}^\omega$ is also a subprocess in $\mathbb{Q}_L$. The subprocess $q_4$ offers the choice of any infinite sequence of actions $b$ and $b'$.*

*The measures $p_1 = \Delta_{q_1}$ and $p_2 = \frac{1}{4}\Delta_{\{ \delta \}} + \frac{3}{4}\Delta_{\{ b', b^\omega \}}$ are both processes in $\mathbb{P}_o$. The process $p_1$ behaves like $q_1$ with probability 1, i.e. the process always offers a single action $b$. The process $p_2$ deadlocks with probability $\frac{1}{4}$ and offers the choice between the single action $b'$ or infinitely many actions $b$ with probability $\frac{3}{4}$.*

*The measure $p_3$ with $p_3(\{ w \}\mathbb{Q}_L) = \frac{1}{2}^{\text{length of } w}$ for all $w \in \{ b, b' \}^*$, is also an element of $\mathbb{P}_L$. Note that $\{ w \}\mathbb{Q}_L$ is a collection of subprocesses, i.e. a collection of subsets of $\mathbb{R}_L$. A subprocess $q$ is in $\{ w \}\mathbb{Q}_L$ if all sequences in $q$ start with $w$. For example $\{ bb'b \} \in \{ bb' \}\mathbb{Q}_L$ and also $\{ bb'b, bb'bb, bb'bbb, \dots \} \in \{ bb' \}\mathbb{Q}_L$.*

*The support of measure $p_3$ consist of the subprocesses which contain a single infinite sequence $\mathrm{spt}(p_3) = \{ \{ w \} | \ w \in OAct^\omega \}$. For each of these subprocesses separately the probability that the subprocess is obtained is 0, in other words $p_3(\{ \{ w \} \}) = 0$ for each infinite sequence $w$. The probability for the set containing all those subprocesses that offer a sequence which starts with e.g. $bb'b$, is $\frac{1}{8}$ (as $p_3(\{ bb'b \}\mathbb{Q}_L) = \frac{1}{8}$).*

*That no finite sequence is offered by any subprocess in the support of $p_3$ can be derived as in example 4.2.18. That only a single sequence is offered and not, for example, the choice between two sequences, can be seen as follows: Take $S_n$ to be the collection of all subprocesses which contain two sequences with differ in the first $n$ places ($n \in \mathbb{N}$). Then $S_n \cap \{ w \}\mathbb{Q}_L = \emptyset$ holds for all words $w$ of length greater than $n$ implying that $p_3(S_n) = 0$. The collection $S$ of all subprocesses offering 2 or more sequences is equal to $\bigcup\{ S_n \mid n \in \mathbb{N} \}$. By countable additivity we obtain $p_3(S) = 0$.*

Having given the domain $\mathbb{P}_L$ the next step is to give an abstraction function that removes the branching information from processes in $\mathbb{P}_o$. In the previous section the moment of nondeterministic choice was abstracted away from by applying the abstraction function $abs_2$ introduced in definition 4.2.42. This abstraction function combines all nondeterministic choices into a single choice at the beginning.of the process. The fact that this may lead to an infinite number of options for the choice presents no problem, because the nondeterministic choices are modeled using the functor $\mathcal{P}_{nco}$ which allows infinite choices (of a restricted but sufficient form).

In the branching domain $\mathbb{P}_o$ the probabilistic choice is modeled by $\mathcal{MP}_f$ which allows for choices between finitely many options. The program $x$ with $D(x) = (a \oplus_{\frac{1}{2}} b); x$ (see example 3.3.12) shows that infinitely many options are also possible for the probabilistic choice and that multisets are insufficient to model them. In chapter 3 and in the previous section this was dealt with by using the functor $Meas$ to model nondeterminism instead.

A problem that arises in this section is that it is not clear how to combine measures over sets in the way needed for a fixed point definition of the abstraction from the branching to the linear domain: In chapter 3 and in the previous section the measures used were measures over action sequences. The observable events were sets of action sequences. The notion of prefixing with an action could be defined on measures by using the operation $\bullet/a$ on sets of sequences. In this section the measure is a measure over subprocesses. An observable event is, therefore, a set of subprocesses. Also, a step is not a single action, but a set of possible actions. In is not clear how to define a notion of prefixing with a set of actions on a measure over subprocesses. The solution to this problem is to only give the probability of special kinds of observable events: The open balls. To find the probability of other events, the properties of a measure are used. For the open balls only a finite part of the branching process is relevant. Restricting to a finite part of a process allows using multisets to model the probability. Unlike with measures, the composition of multisets of subprocesses is possible.

A process $p$ in $\mathbb{P}_o$ which describes the meaning of some program is finitely branching. By truncating the process at depth $n$, a finite tree which contains all information about the first $n$ actions of the process is obtained. In this finite process it is easy to linearize the probability. The process that is obtained this way called the *nth finite approximation* $app_n(p)$ of $p$. The finite approximations of processes are elements of the set $\mathbb{P}_f$. Example 4.3.40 below shows finite approximations of several processes.

**Definition 4.3.39** *The domain $\mathbb{P}_f$ is given by $\mathbb{P}_f = \cup_{n\in\mathbb{N}} \mathbb{P}_n$ with $\mathbb{P}_n$ given by*

$$\begin{array}{rcl} \mathbb{P}_n & = & \mathcal{MP}_f([0,1] \times \mathbb{Q}_n^*) \\ \mathbb{Q}_n^* & = & \mathcal{P}_f(OAct \times \mathbb{Q}_{n-1}^* \cup \{\, \delta \,\}) + \{\, p_\epsilon \,\} \\ \mathbb{Q}_0^* & = & \{\, p_\epsilon \,\} \end{array}$$

*The domain $\mathbb{P}_f$ is ranged over by $P$ and the domain $\mathbb{Q}_n^*$ ranged over by $Q$. The function $app_n : \mathbb{P}_o \to \mathbb{P}_n$ gives the n-th finite approximation of a process and $\widehat{app}_n : \mathbb{Q}_o \to \mathbb{P}_n$ gives the n-th finite approximation of a subprocess. These two functions are defined by*

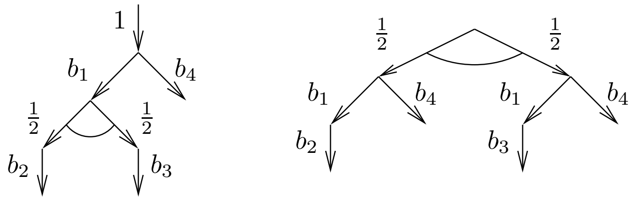
$$\begin{array}{rcl} app_0(p) & = & \{\!\mid 1 \cdot p_\epsilon \mid\!\} \\ app_n(p_\epsilon) & = & \{\!\mid 1 \cdot p_\epsilon \mid\!\} \\ app_n(p) & = & \{\!\mid \rho\sigma \cdot q' \mid \sigma \cdot q' \in \widehat{app}_n(q), \rho \cdot q \in p \mid\!\} \quad \textit{for } p \neq p_\epsilon \end{array}$$

$$\begin{array}{l} \widehat{app}_n(\{\, \langle b_1, p_1\rangle, \ldots, \langle b_m, p_m\rangle \,\}) \ = \\ \qquad \{\!\mid \left(\prod_{i=1}^m \rho_i\right) \cdot \{\, \langle b_1, Q_1\rangle, \ldots \langle b_m, Q_m\rangle \,\} \mid \rho_1 \cdot Q_1 \in app_{n-1}(p_1), \ldots, \\ \qquad\qquad \rho_m \cdot Q_m \in app_{n-1}(Q_m) \mid\!\} \\ \widehat{app}_n(\{\, \delta, \langle b_1, p_1\rangle, \ldots, \langle b_k, p_k\rangle \,\}) \ = \\ \qquad \{\!\mid \left(\prod_{i=1}^k \rho_i\right) \cdot \{\, \delta, \langle b_1, Q_1\rangle, \ldots \langle b_k, Q_k\rangle \,\} \mid \rho_1 \cdot Q_1 \in app_{n-1}(p_1), \ldots, \\ \qquad\qquad \rho_k \cdot Q_k \in app_{n-1}(p_k) \mid\!\} \end{array}$$

*where $m > 0$ and $k \geq 0$ and the empty product $\prod_{i=1}^0$ is 1 by definition.*

Note that the finite approximation functions are only well-defined for finitely branching processes. This is not a problem, as all processes of interest, i.e. processes corresponding to the meaning of some program, are finitely branching. To make the functions well-defined for all processes, an arbitrary element of $\mathbb{P}_f$ is assigned to processes that are not finitely branching.

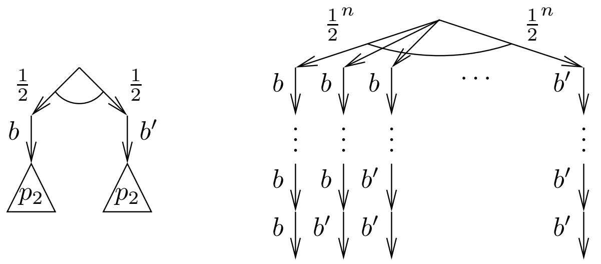
**Example 4.3.40** *The branching process $p_1$ given by $p_1 = \mathcal{O}_g((b_1;(b_2 \oplus_{\frac{1}{2}} b_3)) \,\square\, b_4) = \{\!| 1 \cdot \{ \langle b_1, \{\!| \frac{1}{2} \cdot \{ \langle b_2, p_\epsilon\rangle \}, \frac{1}{2} \cdot \{ \langle b_3, p_\epsilon \rangle \} |\!\}\rangle, \langle b_4, p_\epsilon\rangle \} |\!\}$ can be represented by the tree below. The nth approximation $\mathrm{app}_n(p_1)$ (for $n \geq 2$) of this process is also given.*



$p_1$ $\mathrm{app}_n(p_1)$

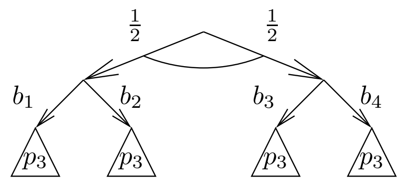
*In this picture the trailing $p_\epsilon$ at each leaf of the trees is omitted. The process $p_1$ contains a single subprocess. This subprocess offers two choices: Execute action $b_1$ followed by the process $\mathcal{O}_g(b_2 \oplus_\rho b_3)$ or to execute action $b_4$ followed by nothing. If action $b_1$ is selected this is followed by $b_2$ with probability $\rho$ and by $b_3$ with probability $1-\rho$. Both these possibilities are combined with $\{ \langle b_4, p_\epsilon \rangle \}$ which is obtained if action $b_4$ is chosen. This results in $\mathrm{app}_n(p_1) = \{\!| \rho \cdot \{ \langle b_1, \{ \langle b_2, p_\epsilon \}\rangle, \langle b_4, p_\epsilon\rangle \}, (1-\rho) \cdot \{ \langle b_1, \{ \langle b_3, p_\epsilon\rangle \}\rangle, \langle b_4, p_\epsilon \rangle \} |\!\}$ as the nth finite approximation for $p_1$ (for $n \geq 2$).*

*Let $D(x) = (b \oplus_{\frac{1}{2}} b'); x$. The nth approximation for $\mathcal{O}_g(x)$ contains $2^{-n} \cdot \{ w \}$ for every word $w$ in $\{ b, b' \}^n$.*

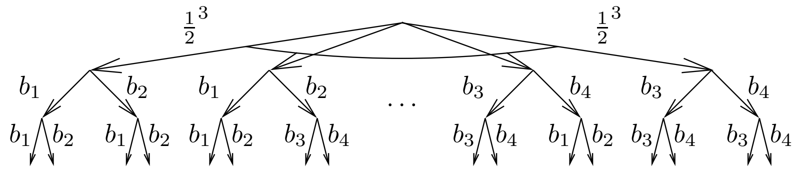


$p_2 = \mathcal{O}_g(x)$ $\mathrm{app}_n(\mathcal{O}_g(x))$

*Let $D(x') = (b_1 \,\square\, b_2) \oplus_{\frac{1}{2}} (b_3 \,\square\, b_4); x'$. The nth approximation for $\mathcal{O}_g(x')$ contains $\frac{1}{2}^{2^n-1} \cdot Q$ where $Q$ is a complete $n$-deep binary tree where at each node either $b_1$ and $b_2$ are offered or $b_3$ and $b_4$.*



$p_3$



$app_2(p_3)$

The abstraction function $abs_L$ from the branching domain to the operational domain is defined based on the finite approximations.

**Definition 4.3.41** *The abstraction function* $\mathrm{abs}_L : \mathbb{P}_o \to \mathbb{P}_L$ *is the unique function satisfying*

$$\mathrm{abs}_L(p)(O_n) \quad = \quad \sum \{\!|\, \rho\Delta_{\mathrm{abs}'_L(Q)}(O_n) \mid \rho \cdot Q \in \mathrm{app}_n(p) \,|\!\}$$

*where* $O_n$ *is an open ball with radius* $2^{-n}$ *and* $\mathrm{abs}'_L : (\cup_{n\in\mathbb{N}}\, \mathbb{Q}^*_n) \to \mathbb{Q}_o$ *is given by*

$$\begin{array}{rcl} \mathrm{abs}'_L(p_\epsilon) & = & \{\, \epsilon \,\} \\ \mathrm{abs}'_L(Q) & = & \cup\{\, b\, \mathrm{abs}'_L(Q') \mid \langle b, Q'\rangle \in t \,\} \cup \{\, \delta \mid \delta \in Q \,\} \end{array}$$

The probability of an open ball $O_n$ of radius of $2^{-n}$ is given by summing the probabilities of all alternatives in the $n$th finite approximation of the processes that end up in $O_n$. Recall from chapter 3 (lemma 3.3.15) that it is sufficient to give the probability of the open balls. By giving $abs_L(p)(O_n)$ for all open balls $O_n$ the measure $abs_L(p)$ is fully determined.

To see if an alternative will end up in the set $O_n$, the function $abs'_L$ is used. The function $abs'_L$ linearizes all nondeterminism to give a set of sequences in $\mathbb{R}_L$, i.e. an subprocess in $\mathbb{Q}_L$. The open ball with radius $2^{-n}$ around a subprocess $q$ contains all processes which behave the same as $q$ in the first $n$ steps. To check whether a subprocess will end up in the open ball it is sufficient to check the first $n$ steps as done in the $n$th approximation of a process, i.e. $app_n(p)$ can be used instead of $p$.

**Example 4.3.42** *Let the process* $p_1$ *be given by* $p_1 = \mathcal{O}_g((b_1; (b_2 \oplus_\rho b_3)) \Box b_4)$. *Using the finite abstraction for* $p_1$ *derived in example 4.3.40 gives*

$$\mathrm{abs}_L(p_1) \quad = \quad \rho\Delta_{\{\, b_1 b_2, b_4 \,\}} + (1-\rho)\Delta_{\{\, b_1 b_3, b_4 \,\}}$$

*Let* $D(x) = b; x$ *and put* $p = \mathcal{O}_g(x)$. *We will show that the process* $\mathrm{abs}_L(p)$ *is equal to* $\Delta_{\{\, b^\omega \,\}}$. *Using the definition of* $\mathcal{O}_g$ *gives that* $p = \{\!|\, 1 \cdot \langle b, p\rangle \,|\!\}$ *holds. For the* $n$*-th finite* *approximation this gives* $\mathrm{app}_n(p) = \{\!|\, 1 \cdot Q_n \,|\!\}$ *with* $Q_0 = p_\epsilon$ *and* $Q_{n+1} = \langle b, Q_n \rangle$ *for* $n \in \mathbb{N}$. *The function* $\mathrm{abs}'_L$ *linearizes* $Q_n$

$$\mathrm{abs}'_L(Q_n) \quad = \quad \{\, b^n \,\}$$

*As* $\{\, b^n \,\}\mathbb{Q}_L$ *is an open ball with radius* $2^{-n}$ *in* $\mathbb{Q}_L$ *the abstraction of* $p$ *satisfies*

$$\mathrm{abs}_L(p)(\{\, b^n \,\}\mathbb{Q}_L) \;=\; \Delta_{Q_n}(\{\, b^n \,\}\mathbb{Q}_L) \;=\; \Delta_{\{\, b^n \,\}}(\{\, b^n \,\}\mathbb{Q}_L) \;=\; 1$$

*Using similar reasoning as in example 4.3.38 gives that* $\mathrm{abs}_L(p)(S) = 0$ *where* $S$ *is the collection of all subprocesses in* $\mathbb{Q}_L$ *which contain more than one element. In other words any subprocess in the support of* $\mathrm{abs}_L(p)$ *offers a single sequence. For all* $n \in \mathbb{N}$ *we have that any sequence that does not start with* $b^n$ *is clearly not possible as* $\mathrm{abs}_L(p)(\{\, b^n \,\}\mathbb{Q}_L) = 1$. *The only sequence that starts with* $b^n$ *for all* $n$ *is* $b^\omega$. *With probability* 1 *the process* $\mathrm{abs}_L(p)$ *delivers the subprocess* $\{\, b^\omega \,\}$.

*Let* $D(x') = b \oplus_{\frac{1}{2}} b'$. *Using the finite approximation of* $\mathcal{O}_g(x)$ *given in example 4.3.40 it is easy to see that the process* $\mathrm{abs}_L(\mathcal{O}_g(x))$ *coincides with the process* $p_3$ *from example 4.3.38.*

The operational semantics can be linearized by applying the abstraction function $\mathit{abs}_L$.

**Definition 4.3.43** *The linear operational semantics* $\mathcal{O}_{iL}[\![\bullet]\!] : \mathcal{L}_{pnd} \to \mathbb{P}_L$ *is given by* $\mathcal{O}_{iL}[\![s]\!] = \mathrm{abs}_L(\mathcal{O}_i[\![s]\!])$ *where* $i$ *is* $l$ *for the local interpretation of nondeterminism or* $g$ *for the global interpretation of nondeterminism.*

As the linear operational models are abstractions from the branching operational models, they are also abstractions of the denotational semantics. The branching model for the global interpretation of nondeterminism can be obtained from the branching model for the local interpretation of nondeterminism. For the linear models this is not the case. The following example shows that $\mathcal{O}_{lL}[\![\bullet]\!]$ and $\mathcal{O}_{gL}[\![\bullet]\!]$ are incomparable in that there are programs identified by $\mathcal{O}_{lL}[\![\bullet]\!]$ but not by $\mathcal{O}_{gL}[\![\bullet]\!]$ and, conversely, programs that are identified by $\mathcal{O}_{gL}[\![\bullet]\!]$ but not by $\mathcal{O}_{lL}[\![\bullet]\!]$. The example also shows that the abstraction to the linear domain cannot be reversed, $\mathcal{O}_{iL}[\![\bullet]\!]$ identifies strictly more statements than $\mathcal{O}_i[\![\bullet]\!]$

**Example 4.3.44** *The programs* $s_1$ *and* $s_2$ *given by* $s_1 = \mathit{coin}; (\mathit{tea} \oplus_\rho \mathit{coffee})$ *and* $s_2 = (\mathit{coin}; \mathit{tea}) \oplus_\rho (\mathit{coin}; \mathit{coffee})$ *are identified by* $\mathcal{O}_{iL}[\![\bullet]\!]$ *but not by* $\mathcal{O}_i[\![\bullet]\!]$ *(cf. example 4.3.11)*

$$\begin{array}{rcl}
\mathcal{O}_{iL}[\![s_1]\!] & = & \rho\Delta_{\{\,\mathit{coin}\,\mathit{tea}\,\}} + (1-\rho)\Delta_{\{\,\mathit{coin}\,\mathit{tea}\,\}} \\
 & = & \mathcal{O}_{iL}[\![s_2]\!] \\
\mathcal{O}_i[\![s_1]\!] & = & \{\!|\, 1 \cdot \{\, \langle \mathit{coin}, \{\!|\, 1 \cdot \{\, \langle \mathit{tea}, p_\epsilon \rangle, \langle \mathit{coffee}, p_\epsilon \rangle \,\} \,|\!\} \rangle \,\} \,|\!\} \\
 & \neq & \{\!|\, 1 \cdot \{\, \langle \mathit{coin}, \{\!|\, 1 \cdot \{\, \langle \mathit{tea}, p_\epsilon \rangle \,\} \,|\!\} \rangle, \langle \mathit{coin}, \{\!|\, 1 \cdot \{\, \langle \mathit{coffee}, p_\epsilon \rangle \,\} \,|\!\} \rangle \,\} \,|\!\} \\
 & = & \mathcal{O}_i[\![s_2]\!]
\end{array}$$

*where* $i$ *is* $l$ *for the local interpretation of nondeterminism or* $g$ *for the global interpretation of nondeterminism.*

*The programs* $s_3 = c_{bean} \Box coin'$ *and* $s_4 = coin'$ *are identified by* $\mathcal{O}_{gL}[\![\bullet]\!]$ *but not by* $\mathcal{O}_{lL}[\![\bullet]\!]$

$$\begin{array}{rcl} \mathcal{O}_{lL}[\![s_3]\!] & = & \Delta_{\{\,\delta, coin'\,\}} \;\neq\; \Delta_{\{\,coin'\,\}} \;=\; \mathcal{O}_{lL}[\![s_4]\!] \\ \mathcal{O}_{gL}[\![s_3]\!] & = & \Delta_{\{\,coin'\,\}} \;=\; \mathcal{O}_{gL}[\![s_4]\!] \end{array}$$

*The programs* $s_5 = coin; (c_{bean} \Box coin')$ *and* $s_6 = (coin; c_{bean}) \Box (coin; coin')$ *are identified by* $\mathcal{O}_{lL}[\![\bullet]\!]$ *but not by* $\mathcal{O}_{gL}[\![\bullet]\!]$

$$\begin{array}{rclcl} \mathcal{O}_{lL}[\![s_5]\!] & = & \Delta_{\{\,coin\,coin', coin\,\delta\,\}} & = & \mathcal{O}_{lL}[\![s_6]\!] \\ \mathcal{O}_{gL}[\![s_5]\!] & = & \Delta_{\{\,coin\,coin'\,\}} & \neq & \Delta_{\{\,coin\,coin', coin\,\delta\,\}} \;=\; \mathcal{O}_{gL}[\![s_6]\!] \end{array}$$

The following combines the results of this subsection and that of the previous subsection

$$\begin{tikzcd}
\mathbb{P}_d + \{\,p_\epsilon\,\} & \mathbb{P}_o & \mathbb{P}_L \\
& \mathcal{O}_l \arrow[r, shift right, "abs_L"'] \arrow[dd, shift right] & \mathcal{O}_{lL}[\![\bullet]\!] \arrow[l, shift right, "/" marking, "\text{ex. } 4.3.44"'] \arrow[dd, leftrightarrow, "/" marking, "\text{ex. } 4.3.44"] \\
\mathcal{D} = \mathcal{O}^* \arrow[ur, "abs_l"] \arrow[dr, "abs_g"'] & & \\
& \mathcal{O}_g \arrow[r, shift left, "abs_L"] \arrow[uu, shift right, "/" marking, "\text{ex. } 4.3.36"'] & \mathcal{O}_{gL}[\![\bullet]\!] \arrow[l, shift left, "/" marking, "\text{ex. } 4.3.44"]
\end{tikzcd}$$

## 4.4 Combining probabilistic choice and nondeterministic choice without using priorities

Giving priority to either nondeterministic choice or probabilistic choice, as done in section 4.2 and section 4.3 respectively, solves the problem of interpreting programs which contain both of these operators. Being able to interpret these programs, the resulting models given in these sections are relatively simple extensions of the model for $\mathcal{L}_P$ given in chapter 3.

A disadvantage of granting priority to nondeterministic choice or probabilistic choice is that it fixes the interpretation of a nondeterministic choice. In either case the models are only suitable for specific types of nondeterminism. As the model with priority for the nondeterministic choice implies an opponent oriented view of nondeterminism, this model cannot be used to describe a selection of available resources. Similarly, as the model with priority for probabilistic choice implies a resource oriented view of nondeterminism, this model does not fit well with a game-like interpretation.

Another disadvantage of giving priority to one type of choice is that only the other type of choice can avoid deadlock: With priority for the nondeterministic choice, probabilistic choice can be conditional, however, nondeterministic choice must be local. With priority for the probabilistic choice, nondeterministic choice can be global, however, probabilistic choice must be unconditional. Global nondeterministic choice cannot be used in a setting with an 'opponent oriented' view of nondeterminism and can also not be combined with conditional probabilistic choice.

A more unified approach is developed in this section, as we want to be able to use the global and local interpretation of nondeterministic choice and the unconditional and conditional interpretation of probabilistic choice independently of what the nondeterminism is supposed to describe (like available resources or choices of a user). Instead of forcing all choices of one type to be made first, each choice is made as it is encountered. This means that a choice higher in the parse tree of a statement is made before any choice below it, independent of whether the choice is a nondeterministic or probabilistic choice. With this approach it is possible to choose for a local or global interpretation of nondeterministic choice and an unconditional or a conditional interpretation of probabilistic choice as desired, within a single framework. (It would even be possible to mix different interpretations in a single language by introducing different operators for e.g. local nondeterministic choice and global nondeterministic choice.) A single transition system gives the transitions possible for each program independent of the interpretations of nondeterministic choice and probabilistic choice. Based on this transition system the operational models for each combination of the interpretations are given.

In the previous two sections, the structure of the operational domain of processes which is used to give the operational meaning of a program was dictated by the type of nondeterministic choice. In this section the structure of the operational domain can still be freely selected. The domain chosen in this section is one which fits well within a verification setting: The structure of the domain is such that it can easily be used to check whether certain properties hold. The properties that one can check concern the probability of given events. Examples of such properties are property 1: "the chance that a sequence is produced that starts with $ab$ is not more than $\frac{1}{2}$" and property 2: "the chance that a sequence is produced that starts with $ab$ may be less than $\frac{1}{2}$". The first property is an example of a bound on the probability of starting with $ab$ that must be met, independent of the nondeterministic behavior. The second property is also a a bound on the probability of starting with $ab$ but this time this bound only needs to be met for one way of making the nondeterministic choices.

The operational domain used in this section will consist of sets of measures over sequences of actions as this domain is suitable for checking properties like property 1 and property 2 above: For a set of measures, one can check, for example, check property 2 by checking whether the property "the chance that a sequence is produced that starts with ab is less than $\frac{1}{2}$" holds for a measure in the set. For a measure over sequences of actions, $\mu \in Meas(Act^\infty)$, this property can easily be checked by looking at the probability $\mu(abAct^\infty)$ of the set of all sequences starting with $ab$.

In the remainder of this section we will concentrate on finding a model that allows checking probabilistic properties instead of on describing and checking these properties. In this section describing a probabilistic property is done by looking at probabilities of suitable observable events. No formal way of expressing properties like for example property 1 above is given. Instead property 1 is checked by looking at the probability of the event $abAct^\infty$. Describing and deriving probabilistic properties of programs is treated more extensively in a setting which also deals with data in chapter 6.

Having given the structure of the operational domain, the question is how the operational models for the different interpretations on nondeterministic and probabilistic choice are obtained from the transition system. In section 4.2 the transitions for each program follow the approach, first make some nondeterministic choices then make some probabilistic choices then take an action. With these transition systems it is easy to interpret the probabilistic choice as a conditional choice. The approach in section 4.3 reverses the order of the choices but is similar, first make some probabilistic choices then make some nondeterministic choices then take an action. With these transition systems it is easy to interpret the nondeterministic choice as a global choice. Transition systems that follow this approach of separating the nondeterministic and probabilistic choices are referred to as alternating models (see e.g. [103, 105]). Here the structure of the transition trees is more complicated. Nondeterministic choices and probabilistic choices are not separated but can be mixed arbitrarily. The more complex structure of the trees makes interpreting these trees and dealing with global nondeterministic choice and conditional probabilistic choice more difficult. Instead of simply removing alternatives that fail to produce actions a mechanism for backtracking through the transition tree is used. Within this setting parallel composition gives additional complications as described in subsection 4.4.7. To separate these two concerns a basic language $\mathcal{L}^{(3a)}_{pnd}$ without parallel composition is treated first, after which parallel composition is added to the language.

The constructs of the basic language are atomic actions, sequential composition, nondeterministic choice, probabilistic choice, recursion and the new construct *fail*, called failure, which describes failure to produce an action. The construct *fail* replaces failure by unsuccessful synchronization. The language allows studying the effects of combining the different interpretations of nondeterministic choice and probabilistic choice in a simple setting. In the language $\mathcal{L}^{(3b)}_{pnd}$ the notion of parallel composition is added. The addition of parallel composition does not affect the way the nondeterministic choice and probabilistic choice are dealt with.

The remainder of this section is organized as follows: In the next subsection the syntax of the basic language $\mathcal{L}^{(3a)}_{pnd}$ is given. In subsection 4.4.2 the transition system $\mathcal{T}^{(3a)}_{pnd}$ for $\mathcal{L}^{(3a)}_{pnd}$ is introduced. Some properties of this transition system are derived in subsection 4.4.3. The operational domain and the operational models which depend on the interpretation of nondeterministic choice and probabilistic choice are defined in subsection 4.4.4. In subsection 4.4.5 a notion of bisimulation is introduced. Why bisimulation is used instead of giving a denotational model is also explained there. In section 4.4.6 a conditional congruence result for the notion of bisimulation introduced in subsection 4.4.5 is derived. In subsection 4.4.7 the language $\mathcal{L}^{(3b)}_{pnd}$ which also contains parallel composition is given along with its operational semantics. Subsection 4.4.8 is devoted to extending the notion of bisimulation to $\mathcal{L}^{(3b)}_{pnd}$.

### 4.4.1 The language $\mathcal{L}^{(3a)}_{pnd}$: choice, chance and failure

As explained in the introduction, we first look at a basic language with failure, recursion, sequential composition, nondeterministic choice and probabilistic choice. As before atomic actions in the set $Act$ are used to describe the basic steps of the computation and procedure variables in $PVar$ are used for recursion. In the previous sections only binary probabilistic and nondeterministic choice were present in the language. This was sufficient as an $n$-ary probabilistic or nondeterministic choice could easily be modeled by using multiple binary choices. With the models in this section an $n$-ary choice is not always equivalent with multiple binary choices (cf. example 4.4.2). Therefore, $n$-ary probabilistic and nondeterministic choices are part of the syntax of the language $\mathcal{L}^{(3a)}_{pnd}$.

**Definition 4.4.1**

*(a) The set of statements Stat, ranged over by s, is given by*

$$s \quad ::= \quad a \mid \mathit{fail} \mid x \mid s \,;s \mid \Box_{i=1}^{n}\, s_i \mid \oplus_{i=1}^{n}\, \rho_i \cdot s_i$$

*where* $n \geq 2$, $0 < \rho_i < 1$, $(i = 1, \dots, n)$ *and* $\rho_1 + \cdots + \rho_n = 1$.

*(b) The set of guarded statements GStat, ranged over by g, is given by*

$$g \quad ::= \quad a \mid \mathit{fail} \mid g \,;s \mid \Box_{i=1}^{n}\, g_i \mid \oplus_{i=1}^{n}\, \rho_i \cdot g_i$$

*where* $n \geq 2$, $0 < \rho_i < 1$, $(i = 1, \dots, n)$ *and* $\rho_1 + \cdots + \rho_n = 1$.

*(c) The set of declarations Decl, ranged over by D, is given by*

$$\mathit{Decl} \quad = \quad \mathit{PVar} \to \mathit{GStat}$$

*(d) The language* $\mathcal{L}^{(3a)}_{pnd}$ *is given by*

$$\mathcal{L}^{(3a)}_{pnd} \quad = \quad \mathit{Decl} \times \mathit{Stat}$$

A program in $\mathcal{L}^{(3a)}_{pnd}$ is a declaration with a statement $s$. As before a fixed declaration $D$ is assumed and $D$ is dropped from the notation.

Actions $a$ in $\mathit{Act}$, procedure variables $x$ in $\mathit{PVar}$ and sequential compositions $s_1; s_2$ are as usual. The declaration $D$ gives the body $D(x)$ for each procedure variable $x$. The recursion is restricted to guarded recursion, i.e. the statement $D(x)$ must be a guarded statement. The fail-statement *fail* embodies abnormal inaction or failure. There will be no activity after its execution. The statement $\Box_{i=1}^{n}\, s_i$ denotes nondeterministic choice. The bound $n$ should be at least 2. The statement acts like one of the $s_i$, for any $i$ with $1 \leq i \leq n$. For a binary nondeterministic choice, i.e. when $n$ is 2, the notation $s_1 \Box s_2$ is employed.

The statement $\oplus_{i=1}^{n}\, \rho_i \cdot s_i$ denotes probabilistic choice or chance. Again the bound $n$ is at least 2. Each number $\rho_i$ is strictly between 0 and 1 and the sum of all the $\rho_i$'s equals 1. For the execution of the statement $\oplus_{i=1}^{n}\, \rho_i \cdot s_i$ a probabilistic choice is made: With probability $\rho_i$ the statement $s_i$ is selected and executed (with $1 \leq i \leq n$). For the binary probabilistic choice we write $s_1 \oplus_\rho s_2$ which executes $s_1$ with probability $\rho$ and $s_2$ with probability $1 - \rho$.

**Example 4.4.2** *The statement* $\Box_{i=1}^{3}\, s_i$ *with* $s_1 = a$, $s_2 = b$ *and* $s_3 = (c \oplus_\rho \mathit{fail})$ *describes a system with three options. The first two options consist of actions a and b respectively. The third option describes a probabilistic component that may produce an action or may fail. If this component fails, both a and b are available as nondeterministic alternatives.*

*The statement* $a \Box (b \Box (c \oplus_\rho \mathit{fail}))$ *describes a different system. This system has the same three options, however, if the third option fails, alternative b will always be used before trying the alternative a.*

**Example 4.4.3** *As an example of a situation that cannot be modeled using the models in the previous two sections consider a probabilistic strategy in a game against an opponent. Choices of the opponent are modeled by nondeterministic choices, the strategy of the player by probabilistic choices. At some point in the game some of the options may not be valid and the choice has to be made between the remaining options. For example in the very simple game described by the program* $(a \Box \mathit{fail}); (b \oplus_{\frac{1}{2}} \mathit{fail})$ *the opponent has to choose between* $a$ *or an invalid option after which the strategy of the player is to choose either option with equal probability. Clearly the only valid execution of this game results in the sequence* $ab$*. To obtain this sequence with probability* $1$*, the nondeterministic choice has to be global and at the same time the probabilistic choice has to be conditional.*

The example above shows a simple situation in which the combination of global nondeterministic choice and conditional probabilistic choice is needed. This combination was not possible in the models in section 4.2 and section 4.3 but a model for this combination will be developed this section.

## 4.4.2 The transition system $\mathcal{T}_{pnd}^{(3a)}$

In this subsection the transition system $\mathcal{T}_{pnd}^{(3a)}$ is given for the language $\mathcal{L}_{pnd}^{(3a)}$. Based on the transition system $\mathcal{T}_{pnd}^{(3a)}$ the operational model $\mathcal{O}$ will be defined in subsection 4.4.4. As in the previous two sections a single transition system is used to give the transitions of a program. The distinction between the different interpretations of nondeterminism and probabilistic choice is made when defining the operational model $\mathcal{O}$ (cf. definition 4.4.15).

A resumption, which describes the part of the program which remains to be executed, is a statement or the empty resumption E denoting a finished computation

$$r \quad ::= \quad s \mid \mathrm{E}$$

The configurations of the transition system are resumptions together with a declaration, $\mathit{Conf} = \mathit{Decl} \times \mathit{Res}$. As a fixed declaration is assumed, the declaration is omitted from the notation.

In section 4.2 the nondeterminism was resolved first and a nondeterministic choice had to be made between programs instead of just between actions. To be able to do this the nondeterminism was resolved explicitly, i.e. making a nondeterministic choice was represented by a transition with an auxiliary label $\nu$ in the transition system. Similarly in section 4.3 the probabilistic choice was resolved by an explicit transition in the transition system. In this section either choice may be made first.

**Example 4.4.4** *In* $a \Box (b \oplus_{\frac{1}{2}} c)$ *the nondeterministic choice occurs first and is therefore made first. One of the arguments of the nondeterministic choice is the probabilistic program* $(b \oplus_{\frac{1}{2}} c)$*. In* $(a \Box b) \oplus_{\frac{1}{2}} c$ *the probabilistic choice occurs first. (Recall that* $(a \Box b) \oplus_{\frac{1}{2}} c$ *is short for* $\oplus_{i=1}^{2} \rho_i \cdot s_i$ *with* $\rho_1 = \frac{1}{2}$*,* $s_1 = (a \Box b)$ *and* $\rho_2 = \frac{1}{2}$*,* $s_2 = c$*.) The probabilistic choice is therefore made first. One of the arguments of the probabilistic choice is the nondeterministic program* $b \Box c$*.*

To be able to deal with both nondeterministic choices between probabilistic programs and probabilistic choices between nondeterministic programs, both nondeterministic choices and probabilistic choices are resolved explicitly in the transition system $\mathcal{T}^{(3a)}_{pnd}$. As a result the labels used in $\mathcal{T}^{(3a)}_{pnd}$ consist of the atomic actions in $Act$, the auxiliary label $\nu$ for resolving nondeterministic choice and the auxiliary labels $\rho$ in $(0,1)$ for resolving probabilistic choice, thus

$$Lab \quad = \quad Act \cup \{\, \nu \,\} \cup (0,1)$$

Recall that $\theta$ is used to range over the set of labels. Additionally we use $\lambda$ to range over non action labels, i.e. $\lambda \in \{\, \nu \,\} \cup (0,1)$.

**Definition 4.4.5** *The transition system $\mathcal{T}^{(3a)}_{pnd}$ for $\mathcal{L}^{(3a)}_{pnd}$ is given by $\mathcal{T}^{(3a)}_{pnd} = (Conf, Lab, \rightarrow, Spec)$. The specification Spec contains the following axioms and rules*

$$a \xrightarrow{a} \mathrm{E} \qquad \textit{(Act)}$$

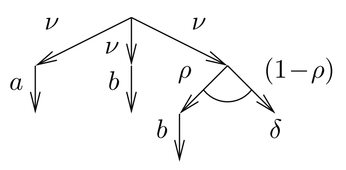
$$\frac{D(x) \xrightarrow{\theta} r}{x \xrightarrow{\theta} r} \quad \textit{(Rec)} \qquad\qquad \frac{s_1 \xrightarrow{\theta} r}{s_1; s_2 \xrightarrow{\theta} r; s_2} \quad \textit{(Seq)}$$

$$\Box^n_{i=1}\, s_i \xrightarrow{\nu} s_i \quad \textit{(Choice)} \qquad\qquad \oplus^n_{i=1}\, \rho_i \cdot s_i \xrightarrow{\rho_i} s_i \quad \textit{(Chance)}$$

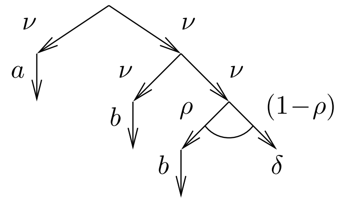
*where $r; s_2$ in rule (Seq) should be read as $s_2$ if $r = \mathrm{E}$.*

A statement consisting of a single action $a$ performs the action $a$ and then terminates. Recursion is handled by unfolding. The transitions for a procedure variable $x$ are precisely those of its body, i.e. the statement $D(x)$. If the first component $s_1$ of a sequential composition $s_1; s_2$ can do a $\theta$-transition, the statement $s_1; s_2$ itself can do a $\theta$-transition as well. After $s_1$ is finished (the case $r = \mathrm{E}$) execution continues with $s_2$. Selection of an alternative in a nondeterministic choice is made explicit by taking a $\nu$-transition. Likewise, the selection of a component of a probabilistic choice which has probability $\rho$ is signaled by a $\rho$-transition. Note that for the fail-statement there is no axiom or rule. As a consequence, *fail* has no transitions.

**Example 4.4.6** *The statement $s = \Box^3_{i=1}\, s_i$ with $s_1 = a$, $s_2 = b$ and $s_3 = (c \oplus_\rho \mathit{fail})$ has three transitions, $s \xrightarrow{\nu} a$, $s \xrightarrow{\nu} b$ and $s \xrightarrow{\nu} (c \oplus_\rho \mathit{fail})$ The statement $s' = a \,\Box\, (b \,\Box\, (c \oplus_\rho \mathit{fail}))$ has two transitions, $s \xrightarrow{\nu} a$ and $s \xrightarrow{\nu} (b \,\Box\, (c \oplus_\rho \mathit{fail}))$. The abstract transition trees for these two statements are*



$a \,\Box\, b \,\Box\, (c \oplus_\rho \mathit{fail})$



$a \,\Box\, (b \,\Box\, (c \oplus_\rho \mathit{fail}))$

*The statement fail; a has no transitions.*

*The statement $fail \Box a$ has two transitions, $fail \Box a \xrightarrow{\nu} fail$ and $fail \Box a \xrightarrow{\nu} a$.*

$$\begin{tikzcd}
& {} \arrow[dl, "\nu"'] \arrow[dr, "\nu"] & \\
\delta & & {} \arrow[d, "a"] \\
& & {}
\end{tikzcd}$$

$fail \Box a$

## 4.4.3 Properties of the transition system $\mathcal{T}_{pnd}^{(3a)}$

In the previous two sections the statements were split into nondeterministically resolved or probabilistically resolved statements and statements that can take a nondeterministic or probabilistic step. A similar distinction can be made here. There are four possible 'modes' for a statement.

**Definition 4.4.7**

(a) *The statement $s$ is said to be deterministic if $s$ has precisely one transition which is of the form $s \xrightarrow{a} r$*

(b) *The statement $s$ is said to be nondeterministic if $s$ has at least one transition and all transitions of $s$ are of the form $s \xrightarrow{\nu} s'$ for some statement $s'$. If $s$ is nondeterministic we write $s \Rightarrow_n S$ where $S$ is the multiset $\{\!\!\{ s' \mid s \xrightarrow{\nu} s' \}\!\!\}$.*

(c) *The statement $s$ is said to be probabilistic if $s$ has at least one transition and all transitions of $s$ are of the form $s \xrightarrow{\rho} s'$ for some probability $\rho$ and some statement $s'$. If $s$ is probabilistic we write $s \Rightarrow_p T$ where $T$ is the multiset $\{\!\!\{ \rho \cdot s' \mid s \xrightarrow{\rho} s' \}\!\!\}$.*

(d) *The statement $s$ is said to fail if $s$ has no transitions.*

For nondeterministic statements the nondeterminism has to be resolved and for probabilistic statements the probabilistic choices have to be resolved. The notation $s \Rightarrow_n S$ is used to collect all nondeterministic alternatives of a nondeterministic statement. Similarly the notation $s \Rightarrow_p T$ collects all probabilistic alternatives of a probabilistic statement. A deterministic statement corresponds to a statement that is both probabilistically resolved and nondeterministically resolved. A new type of statement, i.e. a failing statement is introduced by the explicit modeling of failure.

Each statement in $\mathcal{L}_{pnd}^{(3a)}$ is in exactly one of these modes, i.e. a statement is either deterministic, nondeterministic, probabilistic or failing. This can be shown using weight induction. The definition of the weight of a statement is similar to the definition given in 4.2.9. For statements which occur in both $\mathcal{L}_{pnd}^{(3a)}$ and $\mathcal{L}_{pnd}$ the weight functions coincide.

**Definition 4.4.8** *The function $wgt \colon \mathcal{L}_{pnd}^{(3a)} \to \mathbb{N}$ is given by*

$$\begin{array}{rcl@{\qquad}rcl}
wgt(a) & = & 1 & wgt(s_1; s_2) & = & wgt(s_1) + 1 \\
wgt(fail) & = & 1 & wgt(\Box_{i=1}^n s_i) & = & 1 + \sum_{i=1}^n wgt(s_i) \\
wgt(x) & = & wgt(D(x)) + 1 & wgt(\oplus_{i=1}^n \rho_i \cdot s_i) & = & 1 + \sum_{i=1}^n wgt(s_i)
\end{array}$$

One can show, using the guardedness of the procedure bodies $D(x)$, that the function *wgt* is well-defined. The following lemma can be proven straightforwardly by induction on the weight of a statement.

**Lemma 4.4.9** *For each statement $s$ in $\mathcal{L}^{(3a)}_{pnd}$ exactly one of the following holds: $s$ is deterministic, $s$ is nondeterministic, $s$ is probabilistic or $s$ fails.*

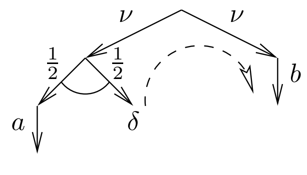
Based on the transition system, different operational models for the different interpretations of nondeterministic choice $\Box$ and probabilistic choice $\oplus$, are defined in the next subsection. The mode of a statement is used in the definition of the different operational models. For well-definedness of these operational models, the following properties are important. They can easily be shown to hold by weight induction.

**Lemma 4.4.10** *The transition system $\mathcal{T}^{(3a)}_{pnd}$ is finitely branching and has no internal divergence, i.e., there are no statements $s_0, s_1, s_2, \ldots$ and $\lambda_1, \lambda_2, \ldots$ in $\{\nu\} \cup (0,1)$ such that $s_0 \xrightarrow{\lambda_1} s_1 \xrightarrow{\lambda_2} s_2 \cdots$ is an infinite computation.*

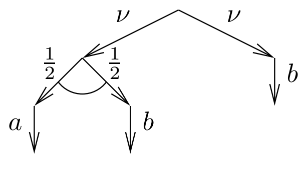
### 4.4.4 The operational semantics $\mathcal{O}$

The nondeterministic choice can be local or global. For the local case any component $s_i$ of a nondeterministic choice $\Box_{i=1}^n s_i$ can be selected. The other options, $s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n$, are dispensed with. For the global case also any component of a nondeterministic choice can be selected. However, if the computation for $s_i$ fails before executing any action, i.e. before performing an $a$-step, the execution will backtrack to the choice and select one of the other alternatives which then takes over control. This backtracking behavior is obtained by storing the other alternatives $s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n$, when $s_i$ is selected. In the previous sections, alternatives were not stored. Instead deadlock was removed by simply removing $\delta$ from the output if other options exist. This simple approach cannot be used here.

**Example 4.4.11** *In the program $(a \oplus_\rho \mathit{fail}) \Box b$ the option $a \oplus_\rho \mathit{fail}$ may fail but only with probability $1-\rho$. If nondeterminism is global, the option $b$ should not replace the whole option $a \oplus_\rho \mathit{fail}$ but only be used in case this option fails. For global nondeterministic choice and unconditional probabilistic choice, this statement behaves the same as $(a \oplus_\rho b) \Box b$.*



$(a \oplus_{\frac{1}{2}} \mathit{fail}) \Box b$



$(a \oplus_{\frac{1}{2}} b) \Box b$

*At the moment the nondeterministic choice has to be made in the program $(a \oplus_\rho \mathit{fail}) \Box b$ it is unknown whether the first component will fail or not. Therefore, selection of this alternatives is allowed, even for the global interpretation of choice. The global behavior of the choice is obtained by backtracking to alternative $b$ if the option $a \oplus_\rho \mathit{fail}$ fails.*

The probabilistic choice can either be unconditional or conditional. The unconditional probabilistic is similar to the local nondeterministic choice; a component is selected and other options are dispensed with. Conditional probability is similar to global nondeterminism. In the conditional interpretation of the probabilistic choice the other options are stored along with their probability when exploring a certain branch and reinvoked later when this exploration fails to perform any $a$-step.

Either interpretation of the nondeterministic choice can be combined with either interpretation for the probabilistic choice, which thus results in four different semantical models.

In order to be able to handle global nondeterministic choice and conditional probabilistic choice, the other alternatives should be stored when selecting an option in order to recover from a deadlock. To this end we introduce alternatives, which are essentially stacks of multisets of statements and of multisets of probability statement pairs.

**Definition 4.4.12** *The collection Alt of alternatives, with typical element $A$, is given by the clause $A ::= \bot \mid S \triangleright A \mid T \triangleright A$ where $\bot$ is a fresh symbol, $S$ is a nonempty multiset of statements $s$, and $T$ is a nonempty multiset of probability-statement pairs $\rho \cdot s$.*

The empty alternative is denoted by the new symbol $\bot$. The two other cases are $S \triangleright A$ and $T \triangleright A$. Here $S$ consists of statements which remain from the resolution of some nondeterministic choice $\Box_{i=1}^{n} s_i$. Likewise, $T$ consists of the alternatives still open for the resolution of a probabilistic choice $\oplus_{i=1}^{n} \rho_i \cdot s_i$. Since this also depends on the probability with which the branch is taken pairs $\rho \cdot s$ are used for this. Below we will employ the constructions $(S \backslash s) \triangleright A$ and $(T \backslash \rho \cdot s) \triangleright A$ of deletion of one element from the topmost multiset. The convention $\emptyset \triangleright A = A$ will be employed in case $S \backslash s$ or $T \backslash \rho \cdot s$ becomes the empty multiset, so that the resulting stacks are still of the proper format.

That the alternatives of a probabilistic choice need to be described by a multiset instead of a set was also seen in previous sections. Here the alternatives of a nondeterministic choice also need to be described by a multiset.

**Example 4.4.13** *Consider the program $(a \oplus_{\frac{1}{2}} fail) \Box (a \oplus_{\frac{1}{2}} fail)$ with the global interpretation of nondeterministic choice and the unconditional interpretation of probabilistic choice. If the first alternative $a \oplus_{\frac{1}{2}} fail$ is selected then this component will fail with probability $\frac{1}{2}$. However, if this happens, the second component will be tried instead. The second component will also fail with probability $\frac{1}{2}$ making the total probability of failure $\frac{1}{4}$.*

*The number of copies of the nondeterministic alternative $(a \oplus_{\frac{1}{2}} fail)$ is important as it influences the probability of failure. (See also example 4.4.16.)*

Given a program and an alternative the operational model $\mathcal{O}$ yields a process that describes the observable behavior of the program with this alternative. The domain of all processes is denoted by $\mathbb{P}_o$.

**Definition 4.4.14** *The operational domain $\mathbb{P}_o$, ranged over by $p$, is given by*

$$\begin{array}{rcl} \mathbb{P}_o & = & \mathcal{P}_{nco}(\mathbb{Q}_o) \\ \mathbb{Q}_o & = & Meas(\mathbb{R}_o) \\ \mathbb{R}_o & = & OAct_\delta^\infty \end{array}$$

*where* $OAct_\delta^\infty = OAct^\star \cup OAct^\star \cdot \{ \delta \} \cup OAct^\omega$.

To obtain the operational semantics for a statement any information in the transition tree of the program that is not considered to be observable behavior is removed and the alternatives are dealt with. For example, the auxiliary $\nu$ and $\rho$ steps do not correspond to actual observable behavior.

**Definition 4.4.15** *For $i = l$ for the local interpretation of nondeterminism or $i = g$ for the global interpretation of nondeterminism and $j = u$ for the unconditional interpretation of probabilistic choice or $j = c$ for the conditional interpretation of probabilistic choice the mapping $\mathcal{O}_{i,j}$: Res $\to$ Alt $\to \mathbb{P}_o$ is defined as follows:*

$$\begin{array}{rcll}
\mathcal{O}_{i,j}(\mathrm{E})[A] & = & \{ \Delta_\epsilon \} & \\[1ex]
\mathcal{O}_{i,j}(s_0)[A] & = & \mathcal{O}_{i,j}(r)[\bot]/a & s_0 \xrightarrow{a} r \\[1ex]
\mathcal{O}_{l,j}(s_0)[A] & = & \bigcup \{ \mathcal{O}_{l,j}(s)[A] \mid s_0 \xrightarrow{\nu} s \} & s_0 \Rightarrow_n S \\
\mathcal{O}_{g,j}(s_0)[A] & = & \bigcup \{ \mathcal{O}_{g,j}(s)[(S \backslash s) \triangleright A] \mid s_0 \xrightarrow{\nu} s \} & s_0 \Rightarrow_n S \\[1ex]
\mathcal{O}_{i,u}(s_0)[A] & = & \oplus \{\!\!\{ \rho \cdot \mathcal{O}_{i,u}(s)[A] \mid s_0 \xrightarrow{\rho} s \}\!\!\} & s_0 \Rightarrow_p T \\
\mathcal{O}_{i,c}(s_0)[A] & = & \oplus \{\!\!\{ \rho \cdot \mathcal{O}_{i,c}(s)[(T \setminus \rho \cdot s) \triangleright A] \mid s_0 \xrightarrow{\rho} s \}\!\!\} & s_0 \Rightarrow_p T \\[1ex]
\mathcal{O}_{i,j}(s_0)[\bot] & = & \{ \Delta_\delta \} & s_0 \not\to \\
\mathcal{O}_{i,j}(s_0)[S \triangleright A] & = & \bigcup \{ \mathcal{O}_{i,j}(s)[(S \setminus s) \triangleright A] \mid s \in S \} & s_0 \not\to \\
\mathcal{O}_{i,j}(s_0)[T \triangleright A] & = & \oplus \{\!\!\{ \frac{\rho}{R(T)} \cdot \mathcal{O}_{i,j}(s)[(T \setminus \rho \cdot s) \triangleright A] \mid \rho \cdot s \in T \}\!\!\} & s_0 \not\to
\end{array}$$

*with* $\oplus : \mathcal{MP}_f([0,1] \times \mathbb{P}_o) \to \mathbb{P}_o$ *given by*

$$\oplus \{\!\!\{ \rho_1 \cdot p_1, \ldots, \rho_n \cdot p_n \}\!\!\} \quad = \quad \{ \textstyle\sum_{k=1}^n \rho_k \mu_k \mid \mu_1 \in p_1, \ldots, \mu_n \in p_n \}$$

Recall that $\Delta_w$ denotes the Dirac measure on the sequence $w$ which assigns probability one to a given Borel set $B$ exactly when $w$ is an element of $B$. The operation $\oplus$ is introduced for ease of notation and to show the symmetry with the clauses for a nondeterministic statement. This operation combines several probabilistic options into a single process like the operation union combines several nondeterministic options: Given a multiset of probability process pairs, the operation $\oplus$ yields an operational process, i.e. a set of measures. The use of a multiset as the argument of $\oplus$ is needed to prevent the 'loss of probability' by multiple occurrences of the same (probability, process) pair.

The definition of the four operational models $\mathcal{O}_{i,j}$ above can be justified as usual by giving a contractive higher-order function $\Phi_{i,j}$ that has $\mathcal{O}_{i,j}$ as its unique fixed point (see for example definition 3.3.20 and lemma 3.3.21). The empty resumption E terminates immediately without producing any actions giving the empty sequence $\epsilon$ with probability 1 as the only possible behavior. For a deterministic statement $s$ which takes an $a$ step to a resumption $r$, the function 'prefix along $a$', denoted $\bullet/a$, is used to prefix the meaning of $r$ with the action $a$. Note that the process $\mathcal{O}_{i,j}(r)[\bot]$ is a set of measures and that $\{ \mu_1, \ldots, \mu_n \}/a = \{ \mu_1/a, \ldots, \mu_n/a \}$ with $\mu/a$ as given in definition 3.3.18. The meaning of a nondeterministic statement depends on the interpretation of nondeterministic choice. If the nondeterministic choice is local, one of the options of the nondeterministic choice is selected and the other options are discarded. If the nondeterministic choice is global, any option may be selected, however, the other options are remembered by adding them to the stack of alternatives. The distinction between unconditional and conditional probabilistic choice is made in a similar manner. For the unconditional choice the other alternatives of a choice are discarded. For a conditional choice the other alternatives together with their probabilities are added to the stack of alternatives.

The role of the alternatives becomes clear when looking at a failing statement. If no alternative exists, a failing statement will lead to deadlock, described by $\delta$. However, if there are still alternatives available from some global choice or some conditional choice, one of these alternatives is taken instead of producing deadlock. If the alternatives belong to some probabilistic choice then the associated probabilities are renormalized before selecting one of these alternatives.

**Example 4.4.16** *Using the global interpretation of nondeterminism and the unconditional interpretation of probabilistic choice, the program $s = \Box_{i=1}^{3} s_i$ with $s_1 = a$, $s_2 = b$ and $s_3 = (c \oplus_\rho \mathit{fail})$ and the program $s' = a \,\Box\, (b \,\Box\, (c \oplus_\rho \mathit{fail}))$ have a different operational meanings.*

*For the program $s$ with no additional alternatives we can find its operational meaning, $\mathcal{O}_{g,u}(s)[\bot]$, as follows:*

$$\begin{aligned}
\mathcal{O}_{g,u}(s)[\bot] &= \mathcal{O}_{g,u}(a)[\{\!|\, b, (c \oplus_\rho \mathit{fail}) \,|\!\} \triangleright \bot] \cup \mathcal{O}_{g,u}(b)[\{\!|\, a, (c \oplus_\rho \mathit{fail}) \,|\!\} \triangleright \bot] \cup \\
&\quad\ \mathcal{O}_{g,u}((c \oplus_\rho \mathit{fail}))[\{\!|\, a, b \,|\!\} \triangleright \bot] \\
&= \{\, \Delta_a \,\} \cup \{\, \Delta_b \,\} \cup \\
&\quad\ \rho \cdot \mathcal{O}_{g,u}(c)[\{\!|\, a, b \,|\!\} \triangleright \bot] \oplus (1-\rho) \cdot \mathcal{O}_{g,u}(\mathit{fail})[\{\!|\, a, b \,|\!\} \triangleright \bot] \\
&= \{\, \Delta_a, \Delta_b \,\} \cup \\
&\quad\ \rho \cdot \{\, \Delta_c \,\} \oplus (1-\rho) \cdot (\mathcal{O}_{g,u}(a)[\{\!|\, b \,|\!\} \triangleright \bot] \cup \mathcal{O}_{g,u}(b)[\{\!|\, a \,|\!\} \triangleright \bot]) \\
&= \{\, \Delta_a, \Delta_b \,\} \cup \rho \cdot \{\, \Delta_c \,\} \oplus (1-\rho) \cdot \{\, \Delta_a, \Delta_b \,\} \\
&= \{\, \Delta_a, \Delta_b, \rho\Delta_c + (1-\rho)\Delta_a, \rho\Delta_c + (1-\rho)\Delta_b \,\}
\end{aligned}$$

*For the program $s'$, also without any additional alternatives, we get*

$$\begin{aligned}
\mathcal{O}_{g,u}(s')[\bot] &= \mathcal{O}_{g,u}(a)[\{\!|\, b \,\Box\, (c \oplus_\rho \mathit{fail}) \,|\!\} \triangleright \bot] \cup \mathcal{O}_{g,u}(b \,\Box\, (c \oplus_\rho \mathit{fail}))[\{\!|\, a \,|\!\} \triangleright \bot] \\
&= \{\, \Delta_a \,\} \cup \mathcal{O}_{g,u}(b)[\{\!|\, (c \oplus_\rho \mathit{fail}) \,|\!\} \triangleright \{\!|\, a \,|\!\} \triangleright \bot] \cup \\
&\quad\ \mathcal{O}_{g,u}((c \oplus_\rho \mathit{fail}))[\{\!|\, b \,|\!\} \triangleright \{\!|\, a \,|\!\} \triangleright \bot] \\
&= \{\, \Delta_a, \Delta_b \,\} \cup \rho \cdot \{\, \Delta_c \,\} \oplus (1-\rho) \cdot (\mathcal{O}_{g,u}(\mathit{fail})[\{\!|\, b \,|\!\} \triangleright \{\!|\, a \,|\!\} \triangleright \bot] \\
&= \{\, \Delta_a, \Delta_b \,\} \cup \rho \cdot \{\, \Delta_c \,\} \oplus (1-\rho) \cdot (\mathcal{O}_{g,u}(b)[\{\!|\, a \,|\!\} \triangleright \bot] \\
&= \{\, \Delta_a, \Delta_b \,\} \cup \rho \cdot \{\, \Delta_c \,\} \oplus (1-\rho) \cdot (\{\, \Delta_a \,\} \\
&= \{\, \Delta_a, \Delta_b, \rho\Delta_c + (1-\rho)\Delta_b \,\}
\end{aligned}$$

*In the process obtained for $s'$, as compared to the process for $s$, the option $\rho\Delta_c + (1-\rho)\Delta_a$ is not present as failure is always replaced by $b$ before the alternative $a$ is considered.*

*Using the transitions derived in example 4.4.6 the operational meaning of the programs* $\mathit{fail}; a$ *and* $\mathit{fail} \Box a$ *is found as follows. The program* $\mathit{fail}; a$ *has no transitions, i.e. this statement is failing. Hence*

$$\mathcal{O}_{i,j}(\mathit{fail}; a)[\bot] \quad = \quad \{\, \Delta_\delta \,\}$$

*The program* $\mathit{fail} \Box a$ *has two nondeterministic transitions. The operational behavior depends on the interpretation of nondeterministic choice. We have*

$$\begin{array}{rcl} \mathcal{O}_{l,j}(\mathit{fail} \Box a)[\bot] &=& \mathcal{O}_{l,j}(\mathit{fail})[\bot] \cup \mathcal{O}_{l,j}(a)[\bot] \\ &=& \{\, \Delta_\delta, \Delta_a \,\} \\ \mathcal{O}_{g,j}(\mathit{fail} \Box a)[\bot] &=& \mathcal{O}_{g,j}(\mathit{fail})[\{\!|\, a \,|\!\} \rhd \bot] \cup \mathcal{O}_{g,j}(a)[\{\!|\, \mathit{fail} \,|\!\} \rhd \bot] \\ &=& \mathcal{O}_{g,j}(a)[\bot] \cup \{\, \Delta_a \,\} \\ &=& \{\, \Delta_a \,\} \end{array}$$

*That the multiplicity of an option in the alternative stack does matter is shown by the following two examples which use a global interpretation on nondeterministic choice and an unconditional interpretation of probabilistic choice. In the program* $\mathit{fail} \Box (a \oplus_{\frac{1}{2}} \mathit{fail})$ *the option* $\mathit{fail}$*, if selected, has one alternative* $(a \oplus_{\frac{1}{2}} \mathit{fail})$*. The total probability of failure is* $\frac{1}{2}$*. In the program* $\mathit{fail} \Box (a \oplus_{\frac{1}{2}} \mathit{fail}) \Box (a \oplus_{\frac{1}{2}} \mathit{fail})$ *there are two alternatives for* $\mathit{fail}$*, both equal to* $a \oplus_{\frac{1}{2}} \mathit{fail}$*. The total probability of failure is* $\frac{1}{4}$*.*

$$\begin{array}{rl} \multicolumn{2}{l}{\mathcal{O}_{g,u}(\mathit{fail} \Box (a \oplus_{\frac{1}{2}} \mathit{fail}))[\bot]} \\ = & \mathcal{O}_{g,u}(\mathit{fail})[\{\!|\, a \oplus_{\frac{1}{2}} \mathit{fail} \,|\!\} \rhd \bot] \cup \mathcal{O}_{g,u}(a \oplus_{\frac{1}{2}} \mathit{fail})[\{\!|\, \mathit{fail} \,|\!\} \rhd \bot] \\ = & \mathcal{O}_{g,u}(a \oplus_{\frac{1}{2}} \mathit{fail})[\bot] \cup (\frac{1}{2} \cdot \mathcal{O}_{g,u}(a)[\{\!|\, \mathit{fail} \,|\!\} \rhd \bot] \oplus \frac{1}{2} \cdot \mathcal{O}_{g,u}(\mathit{fail})[\{\!|\, \mathit{fail} \,|\!\} \rhd \bot]) \\ = & (\frac{1}{2} \cdot \mathcal{O}_{g,u}(a)[\bot] \oplus \frac{1}{2} \cdot \mathcal{O}_{g,u}(\mathit{fail})[\bot]) \cup (\frac{1}{2} \cdot \Delta_a \oplus \frac{1}{2} \cdot \mathcal{O}_{g,u}(\mathit{fail})[\bot]) \\ = & \{\, \frac{1}{2}\Delta_a + \frac{1}{2}\Delta_\delta \,\} \end{array}$$

*Using the global interpretation of nondeterminism and the unconditional interpretation of probabilistic choice, the probability that the program* $\mathit{fail} \Box (a \oplus_{\frac{1}{2}} \mathit{fail})$ *fails is* $\frac{1}{2}$*.*

$$\begin{array}{rcl} \multicolumn{3}{l}{\mathcal{O}_{g,u}(\mathit{fail} \Box (a \oplus_{\frac{1}{2}} \mathit{fail}) \Box (a \oplus_{\frac{1}{2}} \mathit{fail}))[\bot]} \\ &=& \mathcal{O}_{g,u}(\mathit{fail})[\{\!|\, (a \oplus_{\frac{1}{2}} \mathit{fail}), (a \oplus_{\frac{1}{2}} \mathit{fail}) \,|\!\} \rhd \bot] \cup \\ && \mathcal{O}_{g,u}((a \oplus_{\frac{1}{2}} \mathit{fail}))[\{\!|\, \mathit{fail}, (a \oplus_{\frac{1}{2}} \mathit{fail}) \,|\!\} \rhd \bot] \\ &=& \{\, \frac{3}{4}\Delta_a + \frac{1}{4}\Delta_\delta \,\} \end{array}$$

*because*

$$\begin{array}{rl} \multicolumn{2}{l}{\mathcal{O}_{g,u}(\mathit{fail})[\{\!|\, (a \oplus_{\frac{1}{2}} \mathit{fail}), (a \oplus_{\frac{1}{2}} \mathit{fail}) \,|\!\} \rhd \bot]} \\ = & \mathcal{O}_{g,u}((a \oplus_{\frac{1}{2}} \mathit{fail}))[\{\!|\, (a \oplus_{\frac{1}{2}} \mathit{fail}) \,|\!\} \rhd \bot] \\ = & \frac{1}{2} \cdot \mathcal{O}_{g,u}(a)[\{\!|\, (a \oplus_{\frac{1}{2}} \mathit{fail}) \,|\!\} \rhd \bot] \oplus \frac{1}{2} \cdot \mathcal{O}_{g,u}(\mathit{fail})[\{\!|\, (a \oplus_{\frac{1}{2}} \mathit{fail}) \,|\!\} \rhd \bot] \\ = & \frac{1}{2} \cdot \{\, \Delta_a \,\} \oplus \frac{1}{2} \cdot \mathcal{O}_{g,u}((a \oplus_{\frac{1}{2}} \mathit{fail}))[\bot] \\ = & \frac{1}{2} \cdot \{\, \Delta_a \,\} \oplus \frac{1}{2} \cdot \{\, \frac{1}{2}\Delta_a + \frac{1}{2}\Delta_\delta \,\} \\ = & \{\, \frac{3}{4}\Delta_a + \frac{1}{4}\Delta_\delta \,\} \end{array}$$

*And similarly* $\mathcal{O}_{g,u}((a \oplus_{\frac{1}{2}} \mathit{fail}))[\{\!|\, \mathit{fail}, (a \oplus_{\frac{1}{2}} \mathit{fail})\,|\!\} \rhd \bot] = \{\, \frac{3}{4}\Delta_a + \frac{1}{4}\Delta_\delta \,\}$. *Using the global interpretation of nondeterminism and the unconditional interpretation of probabilistic choice, the probability that the program* $\mathit{fail} \Box (a \oplus_{\frac{1}{2}} \mathit{fail}) \Box (a \oplus_{\frac{1}{2}} \mathit{fail})$ *fails is* $\frac{1}{4}$.

*Using the global interpretation of nondeterminism and the conditional interpretation of probabilistic choice, one available action is sufficient to avoid failure.*

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathcal{O}_{g,c}(a \Box (\mathit{fail} \oplus_{\frac{1}{2}} \mathit{fail}))[\bot]} \\
= & \mathcal{O}_{g,c}(a)[\{\!|\, \mathit{fail} \oplus_{\frac{1}{2}} \mathit{fail}\,|\!\} \rhd \bot] \cup \mathcal{O}_{g,c}(\mathit{fail} \oplus_{\frac{1}{2}} \mathit{fail})[\{\!|\, a\,|\!\} \rhd \bot] \\
= & \{\, \Delta_a \,\} \cup (\frac{1}{2} \cdot \mathcal{O}_{g,c}(\mathit{fail})[\{\!|\, \frac{1}{2} \cdot \mathit{fail}\,|\!\} \rhd \{\!|\, a\,|\!\} \rhd \bot] \oplus \frac{1}{2} \cdot \mathcal{O}_{g,c}(\mathit{fail})[\{\!|\, \frac{1}{2} \cdot \mathit{fail}\,|\!\} \rhd \{\!|\, a\,|\!\} \rhd \bot]) \\
= & \{\, \Delta_a \,\} \cup (\frac{1}{2} \cdot \mathcal{O}_{g,c}(\mathit{fail})[\{\!|\, a\,|\!\} \rhd \bot] \oplus \frac{1}{2} \cdot \mathcal{O}_{g,c}(\mathit{fail})[\{\!|\, a\,|\!\} \rhd \bot]) \\
= & \{\, \Delta_a \,\} \cup (\frac{1}{2} \cdot \mathcal{O}_{g,c}(a)[\bot] \oplus \frac{1}{2} \cdot \mathcal{O}_{g,c}(a)[\bot]) \\
= & \{\, \Delta_a \,\} \cup (\frac{1}{2} \cdot \{\, \Delta_a \,\} \oplus \frac{1}{2} \cdot \{\, \Delta_a \,\}) \\
= & \{\, \Delta_a \,\}
\end{array}$$

*If multiple alternatives are available, the "closest one" is always used.*

$$\begin{array}{ll}
\multicolumn{2}{l}{\mathcal{O}_{g,c}(a \Box (b \oplus_{\frac{1}{2}} \mathit{fail}))[\bot]} \\
= & \mathcal{O}_{g,c}(a)[\{\!|\, b \oplus_{\frac{1}{2}} \mathit{fail}\,|\!\} \rhd \bot] \cup \mathcal{O}_{g,c}(b \oplus_{\frac{1}{2}} \mathit{fail})[\{\!|\, a\,|\!\} \rhd \bot] \\
= & \{\, \Delta_a \,\} \cup (\frac{1}{2} \cdot \mathcal{O}_{g,c}(b)[\{\!|\, \frac{1}{2} \cdot \mathit{fail}\,|\!\} \rhd \{\!|\, a\,|\!\} \rhd \bot] \oplus \frac{1}{2} \cdot \mathcal{O}_{g,c}(\mathit{fail})[\{\!|\, \frac{1}{2} \cdot b\,|\!\} \rhd \{\!|\, a\,|\!\} \rhd \bot]) \\
= & \{\, \Delta_a \,\} \cup (\frac{1}{2} \cdot \{\, \Delta_b \,\} \oplus \frac{1}{2} \cdot \{\, \Delta_b \,\}) \\
= & \{\, \Delta_a, \Delta_b \,\}
\end{array}$$

*and similarly we have* $\mathcal{O}_{g,c}((a \Box \mathit{fail}) \oplus_\rho b)[\bot] = \{\, \frac{1}{2}\Delta_a + \frac{1}{2}\Delta_b \,\}$.

The operational semantics should give the meaning of programs. The model $\mathcal{O}_{i,j}$ gives the meaning of resumptions instead of programs and an extra argument containing alternatives is used. No alternatives are available when the execution of a program starts, so for a closed system the empty alternative $\bot$ should be used. The operational semantics $\mathcal{O}_{i,j}[\![\bullet]\!]$ restricts to programs and uses the empty alternative $\bot$ for the alternatives.

**Definition 4.4.17** *The operational semantics* $\mathcal{O}_{i,j}[\![\bullet]\!] \colon \mathcal{L}^{(3a)}_{pnd} \to \mathbb{P}_o$, *for* $i = l$ *for the local interpretation of nondeterminism or* $i = g$ *for the global interpretation of nondeterminism and* $j = u$ *for the unconditional interpretation of probabilistic choice or* $j = c$ *for the conditional interpretation of probabilistic choice is given by*

$$\mathcal{O}_{i,j}[\![s]\!] = \mathcal{O}_{i,j}(s)[\bot]$$

The various interpretations of nondeterministic and probabilistic choice give rise to four models with incomparable distinguishing power. Examples similar to those used in section 4.2 and section 4.3 can be used to show this.

**Example 4.4.18** *The operational semantics* $\mathcal{O}_{i,j}$ *with* $i = l$ *for the local interpretation of nondeterminism or* $i = g$ *for the global interpretation of nondeterminism and* $j = u$ *for the unconditional interpretation of probabilistic choice or* $j = c$ *for the conditional* *interpretation of probabilistic choice are four incomparable models for $\mathcal{L}^{(3a)}_{pnd}$, i.e. for each of pair of models there are programs identified by the first but not by the second model.*

*The programs $a$ and $a \,\Box\, fail$ are identified by the models with a global interpretation of nondeterministic choice but not by the models with a local interpretation.*

$$\mathcal{O}_{l,j}[\![a]\!] = \{\, \Delta_a \,\} \neq \{\, \Delta_a, \Delta_\delta \,\} = \mathcal{O}_{l,j}[\![a \,\Box\, \mathit{fail}]\!]$$

$$\mathcal{O}_{g,j}[\![a]\!] = \{\, \Delta_a \,\} = \mathcal{O}_{g,j}[\![a \,\Box\, \mathit{fail}]\!]$$

*The programs $a; (b \,\Box\, \mathit{fail})$ and $(a; b) \,\Box\, (a; \mathit{fail})$ are identified by the models with a local interpretation of nondeterministic choice but not by the models with a global interpretation.*

$$\mathcal{O}_{l,j}[\![a; (b \,\Box\, \mathit{fail})]\!] = \{\, \Delta_{ab}, \Delta_{a\delta} \,\} = \mathcal{O}_{l,j}[\![(a; b) \,\Box\, (a; \mathit{fail})]\!]$$

$$\mathcal{O}_{g,j}[\![a; (b \,\Box\, \mathit{fail})]\!] = \{\, \Delta_{ab} \,\} \neq \{\, \Delta_{ab}, \Delta_{a\delta} \,\} = \mathcal{O}_{g,j}[\![(a; b) \,\Box\, (a; \mathit{fail})]\!]$$

*The programs $a$ and $a \oplus_{\frac{1}{2}} \mathit{fail}$ are identified by the models with a conditional interpretation of probabilistic choice but not by the models with an unconditional interpretation.*

$$\mathcal{O}_{i,u}[\![a]\!] = \{\, \Delta_a \,\} \neq \{\, \tfrac{1}{2}\Delta_a + \tfrac{1}{2}\Delta_\delta \,\} = \mathcal{O}_{i,u}[\![a \oplus_{\frac{1}{2}} \mathit{fail}]\!]$$

$$\mathcal{O}_{i,c}[\![a]\!] = \{\, \Delta_a \,\} = \mathcal{O}_{i,c}[\![a \oplus_{\frac{1}{2}} \mathit{fail}]\!]$$

*The programs $a; (b \oplus_{\frac{1}{2}} \mathit{fail})$ and $(a; b) \oplus_{\frac{1}{2}} (a; \mathit{fail})$ are identified by the models with an unconditional interpretation of probabilistic choice but not by the models with a conditional interpretation.*

$$\mathcal{O}_{i,u}[\![a; (b \oplus_{\frac{1}{2}} \mathit{fail})]\!] = \{\, \tfrac{1}{2}\Delta_{ab} + \tfrac{1}{2}\Delta_{a\delta} \,\} = \mathcal{O}_{i,u}[\![(a; b) \oplus_{\frac{1}{2}} (a; \mathit{fail})]\!]$$

$$\mathcal{O}_{i,c}[\![a; (b \oplus_{\frac{1}{2}} \mathit{fail})]\!] = \{\, \Delta_{ab} \,\} \neq \{\, \tfrac{1}{2}\Delta_{ab} + \tfrac{1}{2}\Delta_{a\delta} \,\} = \mathcal{O}_{i,c}[\![(a; b) \oplus_{\frac{1}{2}} (a; \mathit{fail})]\!]$$

The backtracking within the transition trees that is caused by the global nondeterministic choice and the conditional probabilistic choice is modeled by alternatives. The alternatives are only used in giving the operational models $\mathcal{O}_{i,j}$. It is also possible to describe the backtracking behavior with a lifted transition relation on extended configurations which contain alternatives (see definition 4.4.19 below). As the way alternatives are used depends on the interpretation of nondeterministic choice and the interpretation of probabilistic choice, the extended transition relation will also depend on these interpretations.

In the next subsection, an extended transition relation which deals with the alternatives according to the interpretation that is being considered is exactly what is needed. One of the results obtained in the next subsection (cf. lemma 4.4.24) is that defining the operational semantics using this extended transition relation gives the same result as the direct definition used here.

### 4.4.5 Bisimulation for $\mathcal{L}_{pnd}^{(3a)}$: First step bisimulation

In chapter 3 and sections 4.2 and 4.3 a denotational model was given. As a denotational model is compositional, it provides an easier way of checking that programs are equivalent. Correctness of the denotational model then gives that denotationally equivalent programs, i.e. programs with the same denotational semantics, are also operationally equivalent, i.e. they have the same operational semantics. In both the sections 4.2 and 4.3 a single denotational model is used, independent of the interpretation of nondeterministic choice and probabilistic choice. A denotational model in this vain can also be given here. Such a model, however, identifies very few statements and as such is not very useful in showing equivalence of programs. Instead *bisimulation* is used here. Defining a notion of bisimulation is a commonly used way of obtaining equivalence of programs. The reason why bisimulation has not been used in this thesis up till now is that in the metric approach an equivalence relation obtained from a notion of bisimulation usually coincides with equality in a denotational model making also giving a notion of bisimulation redundant. (See [91, 38] for a result on strong bisimulation and [190] for a result on probabilistic bisimulation.) This is not the case here. A denotational model is always necessarily a congruence for all operators. The notion of bisimulation defined in this subsection will turn out to be only a conditional congruence for sequential composition (see subsection 4.4.6). As such no denotational model giving the same equivalence relation exists. Usually a notion of bisimulation also results in a full congruence. The result obtained at the end of subsection 4.4.7 (example 4.4.62 and theorem 4.4.63) explain why one does not obtain a congruence relation in this setting.

A first attempt in giving a notion of bisimulation for $\mathcal{L}_{pnd}^{(3a)}$ is simply ignoring the meaning of the auxiliary labels $\nu$ and $\rho$ dealing with resolution of nondeterminism and probabilistic choice. However, the multiset character of the transition relation should be taken into account: One would expect statements $s$ and $s'$ to be bisimilar precisely when there exists a one-one correspondence between the transitions $s \xrightarrow{\theta} r$ of $s$ and the transitions $s' \xrightarrow{\theta} r'$ of $s'$ where $r$ is bisimilar to $r'$. For example, the statements $s = \frac{1}{3}a \oplus \frac{1}{3}a \oplus \frac{1}{3}b$ versus $s' = \frac{1}{3}a \oplus \frac{1}{3}b \oplus \frac{1}{3}b$ are not identified by any operational model model so they should not be bisimilar either. As $s \xrightarrow{\frac{1}{3}} a$ twice and $s' \xrightarrow{\frac{1}{3}} a$ only once there can be no bijection between their transitions.

Dealing with multisets in this way one obtains an equivalence relation. It is also not difficult to check that the equivalence is a congruence for $\mathcal{L}_{pnd}^{(3a)}$. However, this equivalence turns out to be too fine to be interesting. In general, $s \Box s$ and $s$ should not be identified (and indeed $s \Box s$ and $s$ will turn out not to be bisimilar in general), but separation of, e.g., $a \Box a$ and $a$ seems unreasonable.

To find a more interesting notion of bisimulation we aim for a notion which extends the familiar notion of strong bisimulation for nondeterministic systems [171, 159] and the often used notion of Larsen-Skou bisimulation for probabilistic system [149]. In order to obtain this notion of bisimulation we abstract away from the auxiliary labels $\nu$ and $\rho$ which are, in the end, not real actions of the system. When doing so the possible interpretations of nondeterminism and probability, which are the parameters for the models $\mathcal{O}_{i,j}$, have to be dealt with. We aim, for example, at identification of $a \Box \mathit{fail}$ and $a$ for the global interpretation of nondeterministic choice, but not for the local one.

Our interpretations vary in the way alternatives for possible failure are handled. We now make this explicit by considering pairs of resumptions and alternatives as configurations in a 'lifted' transition system. Alternatives will be invoked upon failure (see clause (d)).

**Definition 4.4.19** *Put* $\mathit{Conf}' = \mathit{Decl} \times (\mathit{Res} \times \mathit{Alt})$, *and let* $t$ *range over* $\mathit{Conf}'$. *As usual the declaration part is suppressed. The notation* $r \times A$ *is used for a pair in* $\mathit{Res} \times \mathit{Alt}$. *The lifted transition relation* $\rightarrow$ *on* $\mathit{Conf}' \times \mathit{Lab} \times \mathit{Conf}'$ *is defined as follows (with* $A$ *any alternative in* $\mathit{Alt}$*):*

*(a) If* $s$ *is deterministic with* $s \xrightarrow{a} r$ *then*

$$s \times A \xrightarrow{a} r \times \bot$$

*(b) If* $s$ *is nondeterministic with* $s \Rightarrow_n S$ *then, for all statements* $s'$ *in* $S$

$$\begin{array}{ll} s \times A \xrightarrow{\nu} s' \times A & \textit{if the nondeterministic choice is local} \\ s \times A \xrightarrow{\nu} s' \times ((S \backslash s') \rhd A) & \textit{if the nondeterministic choice is global} \end{array}$$

*(c) If* $s$ *is probabilistic with* $s \Rightarrow_p T$ *then, for all* $\rho \cdot s' \in T$

$$\begin{array}{ll} s \times A \xrightarrow{\rho} s' \times A & \textit{if the probabilistic choice is unconditional} \\ s \times A \xrightarrow{\rho} s' \times ((T \backslash \rho \cdot s') \rhd A) & \textit{if the probabilistic choice is conditional} \end{array}$$

*(d) If* $s$ *is failing,* $s \not\rightarrow$, *then*

$$\begin{array}{ll} s \times (S \rhd A) \xrightarrow{\nu} s' \times ((S \backslash s') \rhd A) & \textit{for all } s' \in S \\ s \times (T \rhd A) \xrightarrow{\rho / R(T)} s' \times ((T \backslash \rho \cdot s') \rhd A) & \textit{for all } \rho \cdot s' \in T \end{array}$$

*where* $R(T) = \sum_{\rho \cdot s' \in T} \rho$.

The transition labels are as before. As for configurations (cf. definition 4.4.9 and lemma 4.4.9) the extended configurations of the form $s \times A$ can be classified as either deterministic, nondeterministic, probabilistic or failing. We write $t \Rightarrow_p T$ for a probabilistic extended configuration, where $T = \{\!|\, \rho \cdot t' \mid t \xrightarrow{\rho} t' \,|\!\}$. A similar notion $t \Rightarrow_n S$ for nondeterministic extended configuration is not needed. Note that definition 4.4.19 is parameterized by the interpretation of nondeterminacy and probabilistic choice. As a consequence, usage of $\rightarrow$ for extended configurations assumes a fixed interpretation of the two operators. Having this fixed interpretation in mind we will write $\mathcal{O}$, for short, instead of $\mathcal{O}_{i,j}$ (see definition 4.4.15).

The technical advantage of maintaining a stack of current alternatives for each model at the transition system level (replacing the different alternative-passing mechanisms in the definition of the $\mathcal{O}_{i,j}$'s) is that this information is made local. This enables a 'first step' analysis for extended configurations.

**Definition 4.4.20** *The set of first steps $\mathcal{R}$, ranged over by $R$, and the set of first step elements $\mathcal{E}$, ranged over by $e$, are given by*

$$\begin{array}{rcl} \mathcal{R} & = & \mathcal{MP}_f([0,1] \times \mathcal{E}) \\ \mathcal{E} & = & (Act \times Conf') \cup \{ \delta \} \end{array}$$

*The first step relation $\rightsquigarrow$ on $Conf' \times \mathcal{R}$, which gives the first steps possible for an extended configuration, is defined as follows:*

*(a) If $t \xrightarrow{a} t'$ then $t \rightsquigarrow \{\!|\, 1 \cdot \langle a, t' \rangle \,|\!\}$.*

*(b) If $t \xrightarrow{\nu} t'$ and $t' \rightsquigarrow R$ then $t \rightsquigarrow R$.*

*(c) If $t \Rightarrow_p \{\!|\, \rho_1 \cdot t_1, \ldots, \rho_n \cdot t_n \,|\!\}$ and $t_i \rightsquigarrow R_i$ for all $i \in \{ 1, \ldots, n \}$ then $t \rightsquigarrow \sqcup_{i=1}^{n} \rho_i\, R_i$ where $\rho\, R = \{\!|\, \rho\sigma \cdot e \mid \sigma \cdot e \in R \,|\!\}$.*

*(d) If $t \not\rightarrow$ then $t \rightsquigarrow \{\!|\, 1 \cdot \delta \,|\!\}$.*

For a deterministic extended configuration $t$ with $t \xrightarrow{a} t'$, only one possible first step exists: An $a$ step to $t'$ with probability 1. Any possible first step of a nondeterministic extended configuration $t$ is a first step of one of the nondeterministic alternatives for $t$. The first steps of a probabilistic extended configuration $t$ are found by combining the first steps of the probabilistic alternatives of $t$. For each alternative $t_i$ a first step $R_i$ is taken and multiplied with the probability of $t_i$, giving $\rho_i\, R_i$. In $\rho_i\, R_i$ the probability of each element of $R_i$ is multiplied by $\rho_i$. The union of the multisets obtained in this way gives a first step for $t$. For a failing extended configuration, there is only a single first step: Failure, described by the element $\delta$, is obtained with probability 1.

Note that for any first step element of the form $\langle a, t \rangle$ in a first step $R$ the alternative in $t$ is always $\perp$, as the alternatives are no longer available after an action is produced.

**Example 4.4.21** *Using the local interpretation of nondeterministic choice, the first steps of the extended configuration $((a \oplus_{\frac{1}{2}} b; s) \Box\, fail) \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \perp)$ are $\{\!|\, \frac{1}{2} \cdot \langle a, \mathrm{E} \times \perp \rangle, \frac{1}{2} \cdot \langle b, s \times \perp \rangle \,|\!\}$ and $\{\!|\, 1 \cdot \langle c, \mathrm{E} \times \perp \rangle \,|\!\}$. This can be derived as follows:*

*As $((a \oplus_{\frac{1}{2}} b; s) \Box\, fail) \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \perp) \xrightarrow{\nu} (a \oplus_{\frac{1}{2}} b; s) \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \perp)$ any first step of the second extended configuration is also a first step of the first extended configuration. The transitions of $(a \oplus_{\frac{1}{2}} b; s) \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \perp)$ are $(a \oplus_{\frac{1}{2}} b; s) \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \perp) \xrightarrow{\frac{1}{2}} a \times A$ and $(a \oplus_{\frac{1}{2}} b; s) \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \perp) \xrightarrow{\frac{1}{2}} (b; s) \times A'$ where the exact form of $A$ and $A'$ is irrelevant. The transitions for these extended configurations are $a \times A \xrightarrow{a} \mathrm{E} \times \perp$ and $b; s \times A \xrightarrow{b} s \times \perp$ Combining these transitions into a first step gives $\{\!|\, \frac{1}{2} \cdot \langle a, \mathrm{E} \times \perp \rangle, \frac{1}{2} \cdot \langle b, s \times \perp \rangle \,|\!\}$.*

*The second first step can be obtained from the transition sequence $((a \oplus_{\frac{1}{2}} b; s) \Box\, fail) \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \perp) \xrightarrow{\nu} fail \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \perp) \xrightarrow{1} c \times \perp \xrightarrow{c} \mathrm{E} \times \perp$. Note that the scaling of probabilities in the second clause of part (d) of definition 4.4.19 gives $fail \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \perp) \xrightarrow{1} c \times \perp$ rather than $fail \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \perp) \xrightarrow{\frac{1}{2}} c \times \perp$.*

The operational meaning of an extended configuration is given by the $\mathcal{O}'$. The definition of $\mathcal{O}'$ is based on the first step relation. The function $\mathcal{O}'$ is consistent with $\mathcal{O}$ of definition 4.4.15, that yields, given a statement and an alternative, a process in $\mathbb{P}_o$.

**Definition 4.4.22** *The mappings $\mathcal{O}'\colon Conf' \to \mathbb{P}_o$, $\overline{\mathcal{O}}'\colon \mathcal{R} \to \mathbb{P}_o$ and $\widehat{\mathcal{O}}'\colon \mathcal{E} \to \mathbb{P}_o$ are given by*

$$\begin{array}{rcl}
\mathcal{O}'(\mathrm{E} \times A) & = & \{\, \Delta_\epsilon \,\} \\
\mathcal{O}'(s \times A) & = & \bigcup\{\, \overline{\mathcal{O}}'(R) \mid s \times A \rightsquigarrow R \,\} \\
\multicolumn{3}{l}{\overline{\mathcal{O}}'(\{\!|\, \rho_1 \cdot e_1, \ldots, \rho_n \cdot e_n \,|\!\})} \\
 & = & \{\, \rho_1\, \mu_1 + \ldots + \rho_n\, \mu_n \mid \mu_1 \in \widehat{\mathcal{O}}'(e_1), \ldots, \mu_n \in \widehat{\mathcal{O}}'(e_n) \,\} \\
\widehat{\mathcal{O}}'(\langle a, t\rangle) & = & \mathcal{O}'(t)/a \\
\widehat{\mathcal{O}}'(\delta) & = & \{\, \Delta_\delta \,\}
\end{array}$$

Note that for any multiset $\{\!|\, \rho_1 \cdot e_1, \ldots, \rho_n \cdot e_n \,|\!\}$ which is actually the first step of some extended configuration, the sum of $\rho_1, \ldots, \rho_n$ is equal to 1. Therefore, the summation $\rho_1\, \widehat{\mathcal{O}}'(e_1) + \ldots \rho_n\, \widehat{\mathcal{O}}'(e_n)$ indeed yields a set of measures.

From the definition of the first step relation $\rightsquigarrow$ and the construction of $\mathcal{O}'$ it is clear that $\mathcal{O}$ and $\mathcal{O}'$ are closely related. To be able to show this, the complexity measure $wgt_c$ is introduced.

**Definition 4.4.23** *The function $wgt_c : Conf' \to \mathbb{N}$ is defined using the auxiliary function $wgt_{Alt} : Alt \to \mathbb{N}$ and the weight function wgt for statements introduced in definition 4.4.8.*

$$wgt_c(s \times A) \;=\; wgt(s) + wgt_{Alt}(A)$$

$$\begin{array}{rcl}
wgt_{Alt}(\bot) & = & 0 \\
wgt_{Alt}(\{\!|\, s_1, \ldots, s_n \,|\!\} \triangleright A) & = & wgt_{Alt}(A) + \sum_{i\in 1}^{n} wgt(s_i) \\
wgt_{Alt}(\{\!|\, \rho_1 \cdot s_1, \ldots, \rho_n \cdot s_n \,|\!\} \triangleright A) & = & wgt_{Alt}(A) + \sum_{i\in 1}^{n} wgt(s_i)
\end{array}$$

Note that the weight function $wgt_c$ satisfies the following property: If $t \xrightarrow{\lambda} t'$ then $wgt_c(t) > wgt_c(t')$ for any label $\lambda$ other than actions.

**Lemma 4.4.24** *For $s \in \mathcal{L}^{(3a)}_{pnd}$ and $A \in Alt$: $\mathcal{O}(s)[A] = \mathcal{O}'(s \times A)$.*

**Proof** The function $\lambda s.\lambda A.\mathcal{O}'(s \times A)$ is shown to be a fixed point of the higher-order operator $\Phi$ implicitly used to define $\mathcal{O}$. This is done by weight induction on $s \times A$ distinguishing the given interpretation of nondeterministic choice and probabilistic choice and the cases $s$ is deterministic, nondeterministic, probabilistic and failing. (The case for $E \times A$ is immediate.) Only a single case is given below

- Assume that the nondeterminism is global and $s$ is nondeterministic with $s \Rightarrow_n S$. Then $s \times A \xrightarrow{\nu} s' \times S \setminus s' \triangleright A$ for all $s' \in S$. As a result $\mathcal{O}'(s \times A) = \cup\{\, \overline{\mathcal{O}}'(R) \mid s \times A \rightsquigarrow R \,\} = \cup\{\, \overline{\mathcal{O}}'(R) \mid s' \times A \rightsquigarrow R, s' \in S \,\}$. As $\cup\{\, \overline{\mathcal{O}}'(R) \mid s' \times A \rightsquigarrow R \,\} = \mathcal{O}'(s' \times A)$

we have $\mathcal{O}'(s \times A) = \cup\{\, \mathcal{O}'(s' \times A) \mid s' \in S\,\}$. As the weight of $s' \times A$ is less than that of $s \times A$ for every $s'$ in $S$ this gives $\mathcal{O}'(s \times A) = \cup\{\, \Phi(\mathcal{O}')(s' \times A) \mid s' \in S\,\}$ but this exactly corresponds to $\Phi(\mathcal{O}')(s \times A)$ $\Box$

**Example 4.4.25** *Using the first steps obtained in example 4.4.21, the process* $\mathcal{O}'(((a \oplus_{\frac{1}{2}} b; s) \,\Box\, \mathit{fail}) \times (\{\!|\, \frac{1}{2} \cdot c \,|\!\} \rhd \bot))$ *can be found as follows:*

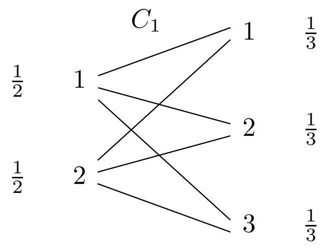
$$\begin{array}{ll} \multicolumn{2}{l}{\mathcal{O}'(((a \oplus_{\frac{1}{2}} b; s) \,\Box\, \mathit{fail}) \times (\{\!|\, \tfrac{1}{2} \cdot c \,|\!\} \rhd \bot))} \\ = & \overline{\mathcal{O}}'(\{\!|\, \tfrac{1}{2} \cdot \langle a, \mathrm{E} \times \bot\rangle, \tfrac{1}{2} \cdot \langle b, s \times \bot\rangle \,|\!\}) \cup \overline{\mathcal{O}}'(\{\!|\, 1 \cdot \langle c, \mathrm{E} \times \bot\rangle \,|\!\}) \\ = & \{\, \tfrac{1}{2}\mu_1 + \tfrac{1}{2}\mu_2 \mid \mu_1 \in \widehat{\mathcal{O}}'(\langle a, \mathrm{E} \times \bot\rangle), \mu_2 \in \widehat{\mathcal{O}}'(\langle b, s \times \bot\rangle)\,\} \\ & \cup\, \widehat{\mathcal{O}}'(\langle c, \mathrm{E} \times \bot\rangle) \\ = & \{\, \Delta_c, \tfrac{1}{2}\Delta_a + \tfrac{1}{2}(\mu/b) \mid \mu \in \mathcal{O}'(s \times \bot)\,\} \end{array}$$

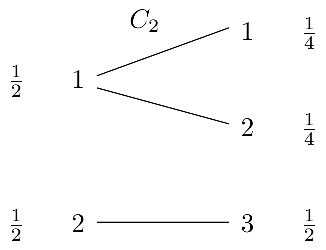
The definition of first steps paves the way for an adequate notion of bisimulation. Comparison of two configurations is based on comparison of their first steps. First a way of lifting a relation on configurations to a relation on first steps is required. The main idea is that a first step $R$ is related to a first step $R'$ if the elements of $R$ can be 'linked' to related elements of $R'$ such that the probabilities of linked elements sum up correctly. For example if the configuration $t_1$ is related to $t_1'$ and to $t_2'$ and $t_2$ is related to $t_3'$ then the lifted relation will relate the first step $\{\!|\, \frac{1}{2} \cdot \langle a, t_1\rangle, \frac{1}{2} \cdot \langle b, t_2\rangle \,|\!\}$ with the first step $\{\!|\, \frac{1}{4} \cdot \langle a, t_1'\rangle, \frac{1}{4} \cdot \langle a, t_2'\rangle, \frac{1}{2} \cdot \langle b, t_3'\rangle \,|\!\}$. The element $\frac{1}{2} \cdot \langle a, t_1\rangle$ can be linked to both $\frac{1}{4} \cdot \langle a, t_1'\rangle$ and $\frac{1}{4} \cdot \langle a, t_2'\rangle$ giving probability $\frac{1}{2}$ for an $a$ step to the equivalence class of $t_1$ in both cases. (See the linking $C_2$ in the example 4.4.27 below.) An extra complication is caused by the fact that there may be elements in $R$ which cannot be split into several parts. These elements may only be linked to a single element of $R'$. To formalize the idea of linking elements in a first step the notion of a *linking* is defined on finite sequences of probabilities.

**Definition 4.4.26** *A relation* $C \subseteq \{\,1, \ldots, n\,\} \times \{\,1, \ldots, m\,\}$ *is called a linking of the sequences of numbers* $\rho_1, \ldots \rho_n$ *and* $\sigma_1, \ldots \sigma_m$ *if for all* $i_0, j_0$ *there exist* $i, j$ *such that* $(i_0, j) \in C$ *and* $(i, j_0) \in C$ *and* $(i_0, j_0) \in C$ *implies that* $\sum_{(i,j_0)\in C} \rho_i = \sum_{(i_0,j)\in C} \sigma_j$. *The linking* $C$ *is said to split on the left at an index* $i$ *if* $(i, j) \in C$ *and* $(i, j') \in C$ *for different* $j$, $j'$. *Similarly,* $C$ *is said to split on the right at an index* $j$ *if* $(i, j) \in C$ *and* $(i', j) \in C$ *for different* $i$, $i'$.

Each element must be linked to some other element and the probabilities for linked elements must sum up correctly.

**Example 4.4.27** *The following pictures show two different linkings. The linking* $C_1$, *in the left part of the picture, splits at every index. The linking* $C_2$, *in the right part, splits only at index 1 on the left.*





When comparing first steps, instead of requiring a one on one correspondence between elements of the first steps, an single element in one first step may correspond to several elements in the other first step. Without this, the first steps of the programs $a$ and $a \oplus_{\frac{1}{2}} a$ can not be related and thus the programs would not be bisimilar. To divide a first step element with probability $\frac{1}{2}$ into two first step elements both with probability $\frac{1}{4}$, as needed for the linking $C_2$ in the example above, it must be possible to divide the probability of the first step element. For the first step element $\delta$ this is possible. For an element of the form $\langle a, t \rangle$ this is possible if the extended configuration $t$ is *splittable*. An extended configuration $t$ is called splittable if the process $\mathcal{O}'(t)$ is splittable. A process $p$ is called splittable if $p = p \oplus_\rho p$ for all probabilities $\rho$ in $(0, 1)$. A statement $s$ is called splittable when the extended configuration $s \times \bot$ is splittable. For a splittable extended configuration having one copy with probability $\frac{1}{2}$ is the same as having two copies both with probability $\frac{1}{4}$.

One would prefer a syntactical characterization of the notion of splittable instead of the semantical one employed here. It is easy to define suitable subclasses of splittable statements, e.g. the sublanguage $\mathcal{L}_p$ introduced in theorem 4.4.63. It is currently an open question if a full syntactical description of splittable statements and extended configurations is possible.

With the notions of linking and splittable extended configurations in place the first step lifting of a relation can be defined.

**Definition 4.4.28** *For a relation $\approx$ on extended configurations, the first step lifting $\approx_{\mathrm{fs}}$ of this relation is such that a first step $R_1 = \{\!|\, \rho_1 \cdot \langle a_1, t_1 \rangle, \ldots, \rho_n \cdot \langle a_n, t_n \rangle, \rho_{n+1} \cdot \delta, \ldots, \rho_{n+n'} \cdot \delta \,|\!\}$ is related to a first step $R_2 = \{\!|\, \sigma_1 \cdot \langle b_1, t'_1 \rangle, \ldots, \sigma_m \cdot \langle b_m, t'_m \rangle, \rho_{m+1} \cdot \delta, \ldots, \rho_{m+m'} \cdot \delta \,|\!\}$, denoted $R_1 \approx_{\mathrm{fs}} R_2$, exactly when there exists a linking $C$ between $\rho_1, \ldots, \rho_n$ and $\sigma_1, \ldots, \sigma_m$ satisfying: (i) if $(i, j) \in C$ then $a_i = b_j$ and $t_i \approx t'_j$, (ii) if $C$ splits on the left at $i$ then $t_i$ is splittable, and (iii) if $C$ splits on the right at $j$ then $t'_j$ is splittable.*

Two first step elements in $Act \times Conf'$ can only be linked if they start with the same action and the resulting extended configurations are related. The first step element $\delta$ can only be linked to $\delta$. If the linking splits at some index $i$ the corresponding extended configuration should be splittable. The linking $C$ can only exist if $\rho_1, \ldots, \rho_n$ and $\sigma_1, \ldots, \sigma_m$ sum up to the same value. This means that also $\rho_{n+1}, \ldots, \rho_{n+n'}$ and $\sigma_{m+1}, \ldots, \sigma_{m+m'}$ sum up to the same value.

**Definition 4.4.29** *The relation $\sim$ on $Conf'$, called first step bisimulation or fs-bisimulation is the greatest equivalence relation on $Conf'$ satisfying*

$$t_1 \sim t_2 \iff \textit{if } t_1 \rightsquigarrow R_1 \textit{ then } t_2 \rightsquigarrow R_2 \textit{ and } R_1 \sim_{\mathrm{fs}} R_2 \textit{ for some } R_2 \textit{ and}$$

*if $t_2 \rightsquigarrow R_2$ then $t_1 \rightsquigarrow R_1$ and $R_1 \sim_{fs} R_2$ for some $R_1$.*

Well-definedness of the above definition can by shown by the usual fixed point argument. The notion of fs-bisimulation can also be given via fs-bisimulation relations, i.e. equivalence relations on *Conf*$'$ satisfying the transfer properties of definition 4.4.29. Only transitivity of fs-bisimulation requires some care.

**Example 4.4.30** *The configurations $a \Box \mathit{fail}$ and $a$ are not bisimilar if nondeterminism is local, but they are bisimilar if nondeterminism is global.*

$$\begin{tikzcd}
 & (a \Box \mathit{fail}) \times \perp \arrow[dl, "\nu"'] \arrow[dr, "\nu"] & & & a \times \perp \arrow[d, "a"'] & & & (a \Box \mathit{fail}) \times \perp \arrow[dl, "\nu"'] \arrow[dr, very thick, "\nu"] & \\
a \times \{\!|\, \mathit{fail} \,|\!\} \rhd \perp \arrow[d, "a"'] & & \mathit{fail} \times \{\!|\, a \,|\!\} \rhd \perp \arrow[d, "\nu"] & \sim & \mathrm{E} & \not\sim & a \times \perp \arrow[d, "a"'] & & \mathit{fail} \times \perp \\
\mathrm{E} & & a \times \perp \arrow[d, "a"'] & & & & \mathrm{E} & & \\
 & & \mathrm{E} & & & & & &
\end{tikzcd}$$

global choice both local choice

*With local nondeterminism, the first steps of $a \Box \mathit{fail}$ are $\{\!|\, 1 \cdot a \,|\!\}$ and $\{\!|\, 1 \cdot \delta \,|\!\}$. The second first step, obtained by making the nondeterministic choice as indicated by the thick arrow in the figure above, is not related to any first step of $a$.*

Before going into the definition of fs-bisimulation for statements we first establish the correctness of fs-bisimulation on extended configurations. Using the correctness of fs-bisimulation, one can characterize $\sim_{fs}$ on first steps as follows. For non-splittable extended configurations, there has to be a one-one correspondence. For splittable extended configurations, the probabilities may be combined.

To be able to prove the last part of the lemma below, it is important to create a one-one correspondence for all extended configurations. For the non-splittable extended configurations this already exists. It is possible to create a one-one correspondence for the splittable extended configurations by splitting the appropriate probabilities.

**Lemma 4.4.31**

*(a) [Splitting Lemma] Let $C$ be a linking of $\rho_1, \dots \rho_n$ and $\sigma_1, \dots \sigma_m$. There exists real numbers $\kappa_{(1,1)} \dots \kappa_{(n,m)}$ such that if $\kappa_{(i,j)} > 0$ then $(i,j) \in C$ and $\sum_{j=1}^m \kappa_{(i_0,j)} = \rho_{i_0}$ and $\sum_{i=1}^n \kappa_{(i,j_0)} = \sigma_{j_0}$.*

*(b) $d(\sum_{k=1}^n \rho_k\, p_k, \sum_{k=1}^n \rho_k\, p'_k) \leq \max\{\, d(p_k, p'_k) \mid 1 \leq k \leq n \,\}$*

*(c) Let $C$ be a linking of $\rho_1, \dots, \rho_n$ and $\sigma_1 \dots \sigma_m$. Then it holds that*

$$d(\textstyle\sum_{i=1}^n \rho_i\, p_i, \sum_{j=1}^m \sigma_j\, p'_j) \leq \max\{\, d(p_i, p'_j) \mid (i,j) \in C \,\}$$

*provided that $p_i$ is splittable whenever $C$ splits at $i$ on the left, and $p'_j$ is splittable whenever $C$ splits at $j$ on the right.*

**Proof**

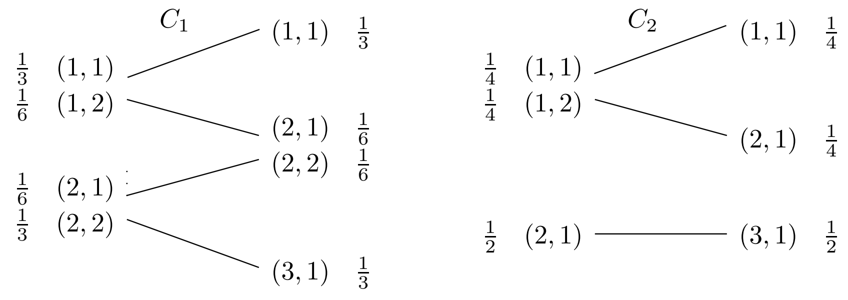
(a) This general graph theoretical result can be obtained using the max-flow min-cut theorem (see, e.g., [47, chapter 3, theorem 1]).

(b) Straightforward calculation of the distances (cf. definition 3.3.14).

(c) Direct consequence of part (a) and part (b). $\Box$

The last part of this lemma is used in the correctness proof of bisimulation for extended configuration (cf. lemma 4.4.33).

**Example 4.4.32** *The following pictures show how the probabilities in the linkings of example 4.4.27 can be split to obtain a one on one correspondence. For the linking $C_2$ there is only one option. For the linking $C_1$ the picture gives one of many possibilities.*



The splitting lemma is used to create, given a linking for two first steps $R$ and $R'$, a one on one correspondence between the extended configurations within the first steps. Using this one on one correspondence it is not difficult to see that the two first steps yield the same processes, i.e. $\mathcal{O}'(R) = \mathcal{O}'(R')$. From this correctness of fs-bisimulation follows directly.

**Lemma 4.4.33** *If $t \sim t'$ then $\mathcal{O}'(t) = \mathcal{O}'(t')$.*

**Proof** Define $\varepsilon \geq 0$ by $\varepsilon = \sup\{\, d(\mathcal{O}'(t), \mathcal{O}'(t')) \mid t \sim t' \,\}$. We show that $\varepsilon = 0$.

Suppose $R = \{\!\!\{\, \rho_1 \cdot \langle a_1, t_1 \rangle, \ldots, \rho_n \cdot \langle a_n, t_n \rangle, \rho_{n+1} \cdot \delta, \ldots, \rho_{n+n'} \cdot \delta \,\}\!\!\}$, $R' = \{\!\!\{\, \sigma_1 \cdot \langle b_1, t'_1 \rangle, \ldots, \sigma_m \cdot \langle b_m, t'_m \rangle, \rho_{m+1} \cdot \delta, \ldots, \rho_{m+m'} \cdot \delta \,\}\!\!\}$ and $R \sim_{\mathrm{fs}} R'$. Put $\rho_0 = \sum_{i=n+1}^{n+n'} \rho_i$ and $\sigma_0 = \sum_{i=m+1}^{m+m'} \sigma_i$. As $R \sim_{\mathrm{fs}} R'$ there exists a linking $C$ between $\rho_1, \ldots, \rho_n$ and $\sigma_1, \ldots, \sigma_m$ such that $(i,j) \in C \Rightarrow a_i = b_j \wedge t_i \sim t'_j$. Also $\rho_0$ must be equal to $\sigma_0$. We then have

$$\begin{aligned}
& d(\mathcal{O}'(R), \mathcal{O}'(R')) \\
& = \; d(\textstyle\sum_{i=1}^n \rho_i \, (\mathcal{O}'(t_i)/a_i) + \rho_0\{\, \Delta_\delta \,\}, \sum_{j=1}^m \sigma_j \, (\mathcal{O}'(t'_j)/b_j) + \sigma_0\{\, \Delta_\delta \,\}) \\
& \leq \; [\text{lemma 4.4.31(c)}] \;\; \max\{\, d(\mathcal{O}'(t_i)/a_i, \mathcal{O}'(t'_j)/b_j) \mid (i,j) \in C \,\} \\
& \leq \; [(i,j) \in C \Rightarrow a_i = b_j \wedge t_i \sim t'_j] \;\; \max\{\, \tfrac{1}{2} d(\mathcal{O}'(t_i), \mathcal{O}'(t'_j)) \mid t_i \sim t'_j \,\}
\end{aligned}$$

$$\le \text{[definition } \varepsilon] \ \tfrac{1}{2}\varepsilon$$

Using this fact we obtain, for any $t, t'$ with $t \sim t'$ that

$$\begin{aligned}
& d(\mathcal{O}'(t), \mathcal{O}'(t')) \\
& = d(\bigcup\{ \mathcal{O}'(R) \mid t \rightsquigarrow R \}, \bigcup\{ \mathcal{O}'(R') \mid t' \rightsquigarrow R' \}) \\
& \le [t \sim t'] \ \max\{ d(\mathcal{O}'(R), \mathcal{O}'(R')) \mid t \rightsquigarrow R, t' \rightsquigarrow R', R \sim_{\mathrm{fs}} R' \} \\
& \le \tfrac{1}{2}\varepsilon
\end{aligned}$$

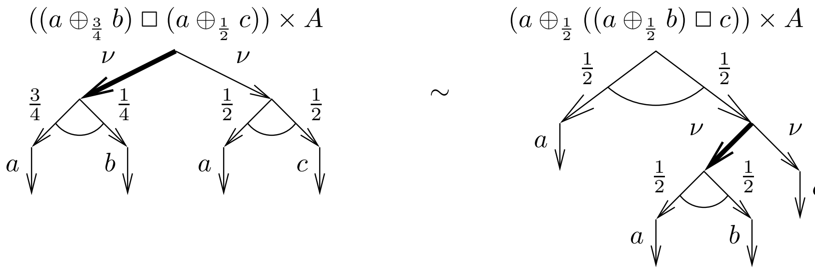
We therefore conclude that $\varepsilon = \sup\{ d(\mathcal{O}'(t), \mathcal{O}'(t')) \mid t \sim t' \} \le \frac{1}{2}\varepsilon$. Hence $\varepsilon = 0$, and $t \sim t'$ implies $d(\mathcal{O}'(t), \mathcal{O}'(t')) = 0$ or, equivalently, $t \sim t'$ implies $\mathcal{O}'(t) = \mathcal{O}'(t')$. $\square$

Finally we have arrived at the definition of fs-bisimulation on statements.

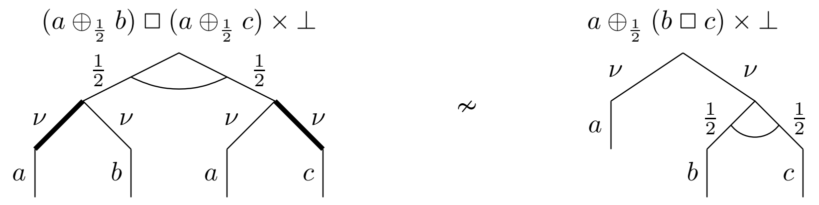
**Definition 4.4.34** *Two statements $s, s' \in \mathcal{L}^{(3a)}_{pnd}$ are called fs-bisimilar, denoted $s \sim s'$, if $s \times A \sim s' \times A$ for all $A \in Alt$.*

Note that for specific models this can be weakened by requiring only bisimilarity for alternatives $A$ of a special form and still obtain both correctness and the congruence results of the next subsection. For example, when using local nondeterminism, alternatives of the form $S \triangleright A$ will never be produced starting from an empty alternative. In this case one only needs to require that $s \times A \sim s' \times A$ for alternatives $A$ given by the clause $A ::= \perp \mid T \triangleright A$.

**Example 4.4.35** *In the transition trees given below the thick arrows indicate one way of resolving the nondeterminism to obtain a first step. The alternative $A$ can be any alternative in Alt.*



*The first step $\{\!|\ \frac{3}{4} \cdot \langle a, \mathrm{E}\rangle, \frac{1}{4} \cdot \langle b, \mathrm{E}\rangle\ |\!\}$ of the first extended configuration is related to the first step $\{\!|\ \frac{1}{2} \cdot \langle a, \mathrm{E}\rangle, \frac{1}{4} \cdot \langle a, \mathrm{E}\rangle, \frac{1}{4} \cdot \langle b, \mathrm{E}\rangle\ |\!\}$ of the second extended configuration and the first step $\{\!|\ \frac{1}{2} \cdot \langle a, \mathrm{E}\rangle, \frac{1}{2} \cdot \langle c, \mathrm{E}\rangle\ |\!\}$ of the first extended configuration is also a first step of the second extended configuration. As bisimilarity holds for every $A \in Alt$, we have, $(a \oplus_{\frac{3}{4}} b) \square (a \oplus_{\frac{1}{2}} c) \sim a \oplus_{\frac{1}{2}} ((a \oplus_{\frac{1}{2}} b) \square c)$.*


$(a \oplus_{\frac{1}{2}} b) \Box (a \oplus_{\frac{1}{2}} c) \times \bot$
$\frac{1}{2}$
$\frac{1}{2}$
$\nu$
$\nu$
$\nu$
$\nu$
$a$
$b$
$a$
$c$
$\not\sim$
$a \oplus_{\frac{1}{2}} (b \Box c) \times \bot$
$\nu$
$\nu$
$a$
$\frac{1}{2}$
$\frac{1}{2}$
$b$
$c$

*The first step* $\{\!|\, \frac{1}{2} \cdot \langle a, \mathrm{E} \rangle, \frac{1}{2} \cdot \langle c, \mathrm{E} \rangle \,|\!\}$ *of* $(a \Box b) \oplus (a \Box c) \times \bot$ *obtained by making the nondeterministic choices as indicated by the thick lines cannot be matched by a similar first step from* $a \Box (b \oplus c) \times \bot$.

As a consequence of the definitions and our earlier results, correctness of fs-bisimulation at the statement level is easily obtained.

**Theorem 4.4.36** *The notion of fs-bisimulation is correct with respect to the operational semantics, that is if two programs* $s$ *and* $s'$ *in* $\mathcal{L}^{(3a)}_{pnd}$ *are bisimilar* $s \sim s'$ *then they have the same meaning* $\mathcal{O}[\![s]\!] = \mathcal{O}[\![s']\!]$.

**Proof** Pick $s, s' \in \mathcal{L}^{(3a)}_{pnd}$ with $s \sim s'$. From $s \sim s'$ we get $s \times \bot \sim s' \times \bot$ by definition of $\sim$ on $\mathcal{L}^{(3a)}_{pnd}$. Thus $\mathcal{O}'(s \times \bot) = \mathcal{O}'(s' \times \bot)$ by lemma 4.4.33 and $\mathcal{O}(s)[\bot] = \mathcal{O}(s')[\bot]$ by lemma 4.4.24, i.e., $\mathcal{O}[\![s]\!] = \mathcal{O}[\![s']\!]$. $\Box$

In the next subsection a conditional congruence result is presented. We will use the name prefixing for placement into a context of the form '$s; \bullet$' and postfixing when the context has the form '$\bullet\,; s$'. The condition required for $\sim$ to be a congruence is that postfixing is limited to postfixing with a splittable statement $s$.

### 4.4.6 Congruence results for first step bisimulation

First step bisimulation is a *conditional congruence*: First step bisimulation is a congruence for probabilistic and nondeterministic choice and for prefixing. First step bisimulation is not generally a congruence for postfixing as the following example show. It is easy to check that $a \sim (a \oplus_\rho a)$, however, $a; (b \Box c) \not\sim (a \oplus_\rho a); (b \Box c)$, as these statements are not operationally equal. Postfixing with the statement $b \Box c$ makes the resulting configuration after the $a$ step non-splittable. This means that, in the context of postfixing $(b \Box c)$, the element $1 \cdot \langle a, t \rangle$ can no longer be split into $\rho \cdot \langle a, t \rangle$ and $(1 - \rho) \cdot \langle a, t \rangle$. When adding the condition that only splittable statements may be postfixed, one does obtain a congruence.

First we will show the congruence result for nondeterministic and probabilistic choice, then the conditional postfixing congruence result is presented. Finally these results are combined with the fact that first step bisimulation is a congruence for prefixing to obtain the conditional congruence result.

Extra alternatives $A'$ can be added to a stack of alternatives $A$ by replacing the final $\bot$ in $A$ by $A'$.

**Definition 4.4.37** *The operation of adding additional alternatives from $A'$ to $A$, denoted $A \triangleright A'$, is given by*

$$\begin{array}{rcl} \bot \triangleright A' & = & A' \\ (S \triangleright A) \triangleright A' & = & S \triangleright (A \triangleright A') \\ (T \triangleright A) \triangleright A' & = & T \triangleright (A \triangleright A') \end{array}$$

In order to obtain a congruence result for fs-bisimulation we need the following technical properties.

**Lemma 4.4.38** *If $s \sim s'$ for $s, s' \in \mathcal{L}^{(3a)}_{pnd}$ then, for any $s'' \in \mathcal{L}^{(3a)}_{pnd}$ and $A, A' \in Alt$,*

$$\begin{array}{rcl} s'' \times (A' \triangleright (\{\!\!\{ s \}\!\!\} \sqcup S) \triangleright A) & \sim & s'' \times (A' \triangleright (\{\!\!\{ s' \}\!\!\} \sqcup S) \triangleright A) \\ s'' \times (A' \triangleright (\{\!\!\{ \rho \cdot s \}\!\!\} \sqcup T) \triangleright A) & \sim & s'' \times (A' \triangleright (\{\!\!\{ \rho \cdot s' \}\!\!\} \sqcup T) \triangleright A) \end{array}$$

**Proof** The relation $\sim$ with these pairs added is shown to be a bisimulation relation by showing by induction on the weight of the lefthand side that for each first step of the extended configuration on the lefthand side a related first step of the righthand side exists. The reverse holds by symmetry.

The proof is straightforward but requires many case distinctions. Only the cases for nondeterministic extended configurations are given.

Put $t = s'' \times (A' \triangleright (\{\!\!\{ s \}\!\!\} \sqcup S) \triangleright A)$ and $t' = s'' \times (A' \triangleright (\{\!\!\{ s' \}\!\!\} \sqcup S) \triangleright A)$ and assume that $t$ is nondeterministic. There are three possibilities: Either $s''$ is nondeterministic with $s'' \Rightarrow_n S''$ or $s''$ is failing and $A' = S' \triangleright A''$ for some $S'$ and $A''$, or $s''$ is failing and $A' = \bot$.

For the first case we have $s'' \Rightarrow_n S''$ and $t \rightsquigarrow R$ exactly when for some $s_0 \in S''$ we have that $t_0 \rightsquigarrow R$ where $t_0$ is defined by $t_0 = s_0 \times (S'' \setminus s_0) \triangleright A' \triangleright (\{\!\!\{ s \}\!\!\} \sqcup S) \triangleright A$. Let $t'_0$ be given by $t'_0 = s_0 \times (S'' \setminus s_0) \triangleright A' \triangleright (\{\!\!\{ s' \}\!\!\} \sqcup S) \triangleright A$. As $wgt_c(t_0) < wgt_c(t)$ induction gives that $t'_0 \rightsquigarrow R'$ for some $R'$ related with $R$. But $t'_0$ is one of the nondeterministic alternatives of $t'$ so also $t' \rightsquigarrow R'$.

The second case is similar. For the third case, $t = s'' \times ((\{\!\!\{ s \}\!\!\} \sqcup S) \triangleright A)$ with $s''$ failing, note that the nondeterministic alternatives of $t$ are $s_0 \times A$ for $s_0 \in S$ and $s \times A$. A first step of $t$ must come from one of these alternatives. The nondeterministic alternatives of $t'$ are the same except that $s' \times A$ replaces $s \times A$. As $s$ is bisimilar with $s'$ a related first step of $s' \times A$ exists for every first step of $s \times A$. $\Box$

Using lemma 4.4.38, the proof of the following lemma is straightforward.

**Lemma 4.4.39** *First step bisimulation is a congruence for nondeterministic choice and probabilistic choice.*

Showing that bisimulation is a conditional congruence for postfixing requires some more work. First we extend the notion of postfixing to extended configurations and first steps. Next we introduce a different formulation of bisimulation which is then used to obtain the conditional congruence result.

**Definition 4.4.40** *For ease of notation we introduce postfixing on alternatives* $;\, : Alt \times Stat \to Alt$, *extended configurations* $;\, : Conf' \times Stat \to Conf'$, *and first steps* $;\, : \mathcal{R} \times Stat \to \mathcal{R}$.

$$\begin{aligned}
\bot; s &= \bot \\
(S \rhd A); s &= \{\!|\, s'; s \mid s' \in S \,|\!\} \rhd (A; s) \\
(T \rhd A); s &= \{\!|\, \rho \cdot (s'; s) \mid \rho \cdot s' \in T \,|\!\} \rhd (A; s) \\
\\
(s' \times A); s &= (s'; s) \times (A; s) \\
\\
R; s &= \{\!|\, (\rho, a, r; s) \mid (\rho, a, r) \in R \,|\!\} \sqcup \{\!|\, (\rho, \delta) \mid (\rho, \delta) \in R \,|\!\}
\end{aligned}$$

The first steps of an extended configuration after postfixing can be found from the first steps prior to postfixing, in a simple manner. By analyzing the transition rules it is easy to check that $t; s \rightsquigarrow R$ exactly when $R = R'; s$ and $t \rightsquigarrow R'$. A formal proof can be given by induction on the weight of $t$.

If $R \sim_{\text{fs}} R'$ and $s''$ is splittable then $R; s'' \sim_{\text{fs}} R'; s''$. This already implies that if $s \sim s'$ then $s; s'' \times A; s'' \sim s'; s'' \times A; s''$. The problem is accounting for alternatives which are not of the form form $A; s''$. To deal with this problem we now introduce an equivalent formulation of bisimilarity.

Consider the adapted first step lifting of a relation obtained by considering $\delta$ to be non-splittable. We use $\approx^{+}_{\mathit{fs}}$ to denote this adjusted first step lifting of $\approx$. Define bisimulation $\sim^{+}$ on extended configuration as before, but now using this new first step lifting. An equivalent formulation of first step bisimulation on statements is obtained by taking $s \sim s'$ exactly when $s \times \bot \sim^{+} s' \times \bot$.

**Lemma 4.4.41** *Two statements $s$ and $s'$ are fs-bisimilar exactly when $s \times \bot \sim^{+} s \times \bot$.*

This formulation has the technical advantage that we need only consider the empty alternative to show bisimilarity of statements. Lemma 4.4.41 can be established by examining the first steps using lemma 4.4.43 below.

The first steps of an extended configuration $s \times A$ can be found from the first steps of $s \times \bot$ by *adding the additional alternatives* in $A$. The operations of adding alternatives to a configuration, a first step element and a first step is defined based on the same notion for alternatives given in definition 4.4.37.

**Definition 4.4.42** *Define the operations of adding additional alternatives* $\bullet \rhd \bullet : (Conf' \times Alt) \to Conf'$, $\bullet \rhd \bullet : (\mathcal{E} \times Alt) \to \mathcal{P}(\mathcal{R})$ *and* $\bullet \rhd \bullet : (\mathcal{R} \times Alt) \to \mathcal{P}(\mathcal{R})$ *by*

$$\begin{aligned}
(s \times A') \rhd A &= s \times (A' \rhd A) \\
\\
\langle a, t \rangle \rhd A &= \{\, \{\!|\, 1 \cdot \langle a, t \rangle \,|\!\} \,\} \\
\delta \rhd A &= \{\, R \mid \mathit{fail} \times A \rightsquigarrow R \,\}
\end{aligned}$$

$$\begin{aligned}
\{\!|\, \rho_1 \cdot e_1, \ldots \rho_n \cdot e_n \,|\!\} \rhd A &= \\
&\{\, \rho_1\, R_1 \sqcup \ldots \sqcup \rho_n\, R_n \mid R_1 \in e_1 \rhd A, \ldots, R_n \in e_n \rhd A \,\}
\end{aligned}$$

To add alternatives to an extended configuration $s \times A$, the alternatives are added to $A$. A first step element $\langle a, t\rangle$ always takes an $a$ step. Any additional alternatives, therefore, do not play a role. Failure, i.e. $\delta$, is replaced by a first step of $\mathit{fail} \times A$, i.e. by failure with alternatives remaining. Since $\mathit{fail} \times A$ may have more than one possible first step, $\delta \triangleright A$ is not a single first step but a set of first steps. To add alternatives to a first step $R$, the alternatives are added to each element of the $R$.

First finding the first steps of an extended configuration $t$ and then adding the alternative $A$ is the same as first adding the alternative $A$ to $t$ and then finding the first steps. A proof can be given by induction on $wgt(t)$.

**Lemma 4.4.43** *If $t \rightsquigarrow R$ then $t \triangleright A \rightsquigarrow R'$ for $R' \in R \triangleright A$ and if $t \triangleright A \rightsquigarrow R'$, $R' \in R \triangleright A$ then $t \rightsquigarrow R'$.*

The first steps of $t \triangleright A$ can be found from the first steps of $t$ by replacing any occurrence of $\delta$ in a first step of $t$ by a first step of $\mathit{fail} \times A$ as described in definition 4.4.42.

**Example 4.4.44** *The first steps*

$$\{\!|\, \tfrac{1}{2} \cdot \langle a, \mathrm{E} \times \bot\rangle, \tfrac{1}{2} \cdot \langle b, \mathrm{E} \times \bot\rangle \,|\!\} \quad \textit{and} \quad \{\!|\, \tfrac{1}{2} \cdot \langle a, \mathrm{E} \times \bot\rangle, \tfrac{1}{2} \cdot \langle c, \mathrm{E} \times \bot\rangle \,|\!\}$$

*of the extended configuration $(a \oplus_{\frac{1}{2}} \mathit{fail}) \times (\{\!|\, b, c \,|\!\} \triangleright \bot)$ can be obtained from the first step $\{\!|\, \frac{1}{2} \cdot \langle a, \mathrm{E} \times \bot\rangle, \frac{1}{2} \cdot \delta \,|\!\}$ of $(a \oplus_{\frac{1}{2}} \mathit{fail}) \times \bot$ by replacing $\frac{1}{2} \cdot \delta$ by $\frac{1}{2} \cdot \langle b, \mathrm{E} \times \bot\rangle$ and $\frac{1}{2} \cdot \langle c, \mathrm{E} \times \bot\rangle$ respectively.*

*The first step $\{\!|\, \frac{1}{2} \cdot \langle a, \mathrm{E} \times \bot\rangle, \frac{1}{4} \cdot \langle b, \mathrm{E} \times \bot\rangle, \frac{1}{4} \cdot \langle c, \mathrm{E} \times \bot\rangle \,|\!\}$ of the extended configuration $(a \oplus_{\frac{1}{2}} \mathit{fail}) \times (\{\!|\, \frac{1}{2} \cdot b, \frac{1}{2} \cdot c \,|\!\} \triangleright \bot)$ can be obtained from the first step $\{\!|\, \frac{1}{2} \cdot \langle a, \mathrm{E} \times \bot\rangle, \frac{1}{2} \cdot \delta \,|\!\}$ of $(a \oplus_{\frac{1}{2}} \mathit{fail}) \times \bot$ by replacing $\frac{1}{2} \cdot \delta$ by the two elements $\frac{1}{4} \cdot \langle b, \mathrm{E} \times \bot\rangle$ and $\frac{1}{4} \cdot \langle c, \mathrm{E} \times \bot\rangle$.*

Using the equivalent formulation of bisimulation we now obtain a conditional congruence result for postfixing. We show that a relation $\mathcal{S}$ which contains $(s; s'', s'; s'')$ is a bisimulation.

**Lemma 4.4.45** *The relation $\mathcal{S} = \{\, (s; s'', s'; s'') \mid s \sim s', s'' \mathit{splittable} \,\} \cup \sim$ is a first step bisimulation.*

**Proof** Assume that $s \sim s'$ and that $s''$ is a splittable statement. If $s; s'' \times \bot \rightsquigarrow R_0$ then $s \times \bot \rightsquigarrow R$ and $R_0 = R; s$ (for some $R$). Using the second formulation of bisimilarity (lemma 4.4.41) gives that there exists a first step $R'$ with $s' \times \bot \rightsquigarrow R'$ and $R \sim^{+} R'$. But then $s'; s'' \times \bot \rightsquigarrow R_1 = R; s''$. It is easy to see that if $R \sim^{+}_{\mathit{fs}} R'$ then $R; s'' \; \mathcal{S}^{+}_{\mathit{fs}} \; R'; s''$ which gives that $R_0 \; \mathcal{S}^{+}_{\mathit{fs}} \; R_1$ holds. For every first step of $s; s''$ a related first step of $s'; s''$ can be found. The opposite direction holds by symmetry. $\Box$

Now we have all the ingredients for the announced conditional congruence result.

**Theorem 4.4.46**

*(a) First step bisimulation is a congruence for prefixing, nondeterministic choice and probabilistic choice.*

*(b) First step bisimulation is a congruence for postfixing with splittable statements.*

**Proof** Lemma 4.4.39 gives the congruence result for nondeterministic choice and probabilistic choice. Lemma 4.4.45 gives a conditional postfixing result. If $s \sim s'$ and $s''$ is splittable then $s; s'' \sim s'; s''$.

Clearly, $\sim$ is a congruence for action prefixing, since $s \sim s'$ implies $\{\!|\, 1 \cdot \langle a, s \times \bot \rangle \,|\!\}\sim_{fs}$ $\{\!|\, 1 \cdot \langle a, s' \times \bot \rangle \,|\!\}$ and these are exactly the first steps of $a; s \times A$ and $a; s' \times A$. The proof for general prefixing proceeds by induction on the weight of the prefixed statement. $\Box$

As we have already seen, the requirement that a postfixed statement is splittable cannot be dropped.

In the next subsections the results obtained for $\mathcal{L}^{(3a)}_{pnd}$ are extended to a language $\mathcal{L}^{(3b)}_{pnd}$ which additionally contains parallel composition.

### 4.4.7 The language $\mathcal{L}^{(3b)}_{pnd}$: Adding parallel composition

In this subsection the language $\mathcal{L}^{(3b)}_{pnd}$ is obtained by adding parallel composition to the language $\mathcal{L}^{(3a)}_{pnd}$ introduced in subsection 4.4.1. As before the operator $\|$ called merge is used to introduce parallelism into the language. The statement $s \| s'$ describes the parallel execution of the statements $s$ and $s'$. A distributed interpretation of parallelism is adopted in the sense that there will be progress if either statement is able to make progress. Failure of one statement does not directly cause failure of the system as a whole. Unlike in the previous sections, synchronization of parallel components will not be allowed. The operation merge with no synchronization is also referred to as free merge in literature. The complications that synchronization between components causes in this setting are discussed in section 4.5.

The syntax of $\mathcal{L}^{(3b)}_{pnd}$ is obtained by adding the operator $\|$, called merge, to the syntax of $\mathcal{L}^{(3a)}_{pnd}$. As no synchronization is present, the set of atomic action $Act$ does not contain any synchronization actions but only observable actions.

**Definition 4.4.47**

*(a) The set of statements Stat, ranged over by s, is given by*

$$s ::= a \mid \mathit{fail} \mid x \mid s\,;s \mid \Box_{i=1}^{n}\, s_i \mid \oplus_{i=1}^{n}\, \rho_i \cdot s_i \mid s \| s$$

*where $n \geq 2$, $0 < \rho_i < 1$, $(i = 1, \ldots, n)$ and $\rho_1 + \cdots + \rho_n = 1$.*

*(b) The set of guarded statements GStat, ranged over by g, is given by*

$$g \quad ::= \quad a \mid \mathit{fail} \mid g\,;s \mid \Box_{i=1}^{n}\, g_i \mid \oplus_{i=1}^{n}\, \rho_i \cdot g_i \mid g \| g$$

*where $n \geq 2$, $0 < \rho_i < 1$, $(i = 1, \ldots, n)$ and $\rho_1 + \cdots + \rho_n = 1$.*

*(c) The set of declarations Decl, ranged over by D, is given by*

$$\mathit{Decl} \quad = \quad \mathit{PVar} \to \mathit{GStat}$$

*(d) The language $\mathcal{L}^{(3b)}_{pnd}$ is given by*

$$\mathcal{L}^{(3b)}_{pnd} \quad = \quad \mathit{Decl} \times \mathit{Stat}$$

As usual a fixed declaration $D$ is assumed, and the declaration is dropped from the notation. The interpretation of the atomic actions, the statement *fail*, procedure variables and the operators for sequential composition nondeterministic choice and probabilistic choice is as in the previous subsections. The statement $s \| s'$ denotes the parallel composition of the statements $s$ and $s'$. To be able to deal with parallel composition in the transition system the auxiliary operator leftmerge $\lfloor\!\lfloor$ is used as before. Note that, as no communication is allowed between the parallel components the auxiliary operator $|$, used in section 4.2, is not needed here. The extended set of statements obtained by adding the operator $\lfloor\!\lfloor$ is denoted by $\mathit{Stat}^+$. The set $\mathit{Stat}^+$ is also ranged over by $s$.

$$s ::= \ldots \mid s \lfloor\!\lfloor s$$

In $s \| s'$ either $s$ or $s'$ can produce the first action. The statement $s \lfloor\!\lfloor s'$ denotes a parallel composition where the first action will come from $s$, provided that $s$ can indeed produce an action. In this statement, the component $s$ is called the *primary component.* A new auxiliary label $\pi$ is used to signal selection of the primary component in the transition system. Why the label $\nu$ cannot be used for this is explained after example 4.4.54.

A distributed view of parallelism is used: If the primary component in a parallel composition cannot produce an action, this is assumed not to affect the other components. As a result, if the primary component fails, another component that at some point produces an action takes over the computation. In the transition system this event will be signaled by the new auxiliary label $\mathit{tt}$, called take-over. Determining whether a component can produce an action and if so executing this action is assumed to be done instantly. As a result, a take-over event can only occur if the primary component fails.

As before the atomic actions in $\mathit{Act}$, the auxiliary label $\nu$ and the probabilities in $(0,1)$ may also be used as transition labels. The set of labels $\mathit{Lab}$ for the transition system $\mathcal{T}^{(3b)}_{pnd}$ is thus given by

$$\mathit{Lab} \quad = \quad \mathit{Act} \cup \{\, \nu, \pi, \mathit{tt} \,\} \cup (0,1)$$

As a take-over transition is only possible for statements for which the primary component fails, the rule for take-over in the transition system uses the condition that the primary component fails. In the previous subsections a statement was defined to be failing if it has no transitions. To avoid the use of negative premises in the transition system a syntactical characterization of failing statements is given instead of using this definition. After giving the transition system it is easy to check that the failing statements as defined here are exactly the statements which have no transitions.

**Definition 4.4.48** *The statement fail fails. The statement $x$ fails whenever $D(x)$ fails. The statement $s; s'$ fails for all failing statements $s$. All other statements are not failing.*

Well-definedness of the set of failing statements can be shown by weight induction. To this end the weight function *wgt* introduced in definition 4.4.8 is extended as follows.

**Definition 4.4.49** . *The function wgt : Stat$^+ \to \mathbb{N}$ is defined by adding the following clauses to the clauses given in definition 4.4.8.*

$$\begin{aligned} wgt(s \| s') &= wgt(s \mathbin{|\!\lfloor} s') + wgt(s' \mathbin{|\!\lfloor} s) + 1 \\ wgt(s \mathbin{|\!\lfloor} s') &= wgt(s) + wgt(s') + 1 \end{aligned}$$

Well-definedness of weight is clear by structural induction, first on guarded statements and then on all statements.

The transition system $\mathcal{T}^{(3b)}_{pnd}$ is obtained from $\mathcal{T}^{(3a)}_{pnd}$ by adding the axioms and rules for merge, leftmerge and take-over.

**Definition 4.4.50** *The transition system $\mathcal{T}^{(3b)}_{pnd}$ for $\mathcal{L}^{(3b)}_{pnd}$ is given by $\mathcal{T}^{(3b)}_{pnd}$ = (Res, Lab, $\to$, Spec) where Spec contains the axioms and rules of $\mathcal{T}^{(3a)}_{pnd}$ given in definition 4.4.5 and additionally*

$$\begin{array}{ll} s_1 \| s_2 \xrightarrow{\pi} s_1 \mathbin{|\!\lfloor} s_2 \\ s_1 \| s_2 \xrightarrow{\pi} s_2 \mathbin{|\!\lfloor} s_1 \end{array} \quad \textit{(Merge)} \qquad \frac{s_1 \xrightarrow{a} r}{s_1 \mathbin{|\!\lfloor} s_2 \xrightarrow{a} r \| s_2} \quad \textit{(Leftmerge 1)}$$

$$\frac{s_1 \xrightarrow{\lambda} s}{s_1 \mathbin{|\!\lfloor} s_2 \xrightarrow{\lambda} s \mathbin{|\!\lfloor} s_2} \quad \textit{(Leftmerge 2)} \qquad \frac{s_1 \ \textit{fails}}{s_1 \mathbin{|\!\lfloor} s_2 \xrightarrow{tt} s_2; \textit{fail}} \quad \textit{(Take Over)}$$

*with $\lambda \in \{\nu, \pi, tt\} \cup (0,1)$. In rule (Leftmerge 1) $r \| s_2$ should be read as $s_2$ if $r = \mathrm{E}$.*

The rules from $\mathcal{T}^{(3a)}_{pnd}$ are included unchanged, however, the statements within these rules now refer to statements in $\mathcal{L}^{(3b)}_{pnd}$. The axioms (Merge) express that either component of the parallel composition is allowed to begin first. By selection of a primary component the $\|$ is resolved into $\mathbin{|\!\lfloor}$. The statement $s_1 \mathbin{|\!\lfloor} s_2$ will behave like $s_1$ until $s_1$ has taken an action or fails. The rule (Leftmerge 1) describes the situation in which the primary component $s_1$ takes an action. Rule (Leftmerge 2) describes the situation in which primary component $s_1$ has to resolve choice, chance or parallelism or do a take-over. The primary component still has to do an action so the $\mathbin{|\!\lfloor}$ remains on the right-hand side of the conclusion in rule (Leftmerge 2).

If, in $s_1 \mathbin{|\!\lfloor} s_2$, the primary component $s_1$ cannot take any steps, i.e. fails, the component $s_2$ will take over. This event is made explicit in the transition system by the use of a $tt$-transition as described by rule (Take Over). Of course, if a component fails, the parallel composition cannot terminate normally. After all other components terminate or fail, the parallel system as a whole will fail.

Clearly the new axioms and rules do not play a role for programs without parallel composition, i.e. for programs from $\mathcal{L}^{(3a)}_{pnd}$, so for these programs the transition trees remain the same.

Using the transition system different operational semantics can be defined depending on the interpretation of the nondeterministic choice and the probabilistic choice. The definitions are mostly extensions of a corresponding definition in the previous section.

In $\mathcal{L}^{(3a)}_{pnd}$ a statement was either failing, deterministic, non-deterministic or probabilistic. In $\mathcal{L}^{(3b)}_{pnd}$ two new types of statements are present. A statement $s$ is said to be *parallel* if it has at least one transition and all transitions are of the form $s \xrightarrow{\pi} s'$ for some statement $s'$. A statement $s$ is called *take-over* if it has at least one transition and all transitions are of the form $s \xrightarrow{tt} s'$. It is easy to check that again each statement is of exactly one of these types.

Alternatives for choices that are made have to be remembered in the global and conditional models. In $\mathcal{L}^{(3b)}_{pnd}$ there is an extra complication compared to $\mathcal{L}^{(3a)}_{pnd}$. An alternative from within the component will prevent the component from failing and should therefore take precedence over a take-over action. An alternative from outside the parallel composition should only be used if the whole parallel composition fails. As the parallel composition will only fail if all components fail, the take-over action should be tried before reverting to any top level alternative.

**Example 4.4.51** *For the statement $(a \oplus_{\frac{1}{2}} fail) \| b$ either component can become the primary component as described by the two transitions, $(a \oplus_{\frac{1}{2}} fail) \| b \xrightarrow{\pi} (a \oplus_{\frac{1}{2}} fail) \mathbin{\|\!\lfloor} b$ and $(a \oplus_{\frac{1}{2}} fail) \| b \xrightarrow{\pi} b \mathbin{\|\!\lfloor} (a \oplus_{\frac{1}{2}} fail)$, of this statement. If $(a \oplus_{\frac{1}{2}} fail)$ becomes the primary component, then the fail is selected with probability $\frac{1}{2}$. With a conditional interpretation of probabilistic choice, the alternative $a$ is available to prevent failure. The alternative $a$ is an alternative within the component. The statement $(a \oplus_{\frac{1}{2}} fail) \| b$ is equivalent with $a \| b$.*

*In the statement $a \oplus_{\frac{1}{2}} (fail \| b)$ the option $fail \| b$ will never fail in the first step; a $b$-step (from the second component) is always possible. The action $a$ is, therefore, not available to replace failure. The alternative $a$ is an alternative from outside the current component. The program $a \oplus_{\frac{1}{2}} (fail \| b)$ is equivalent with the program $a \oplus_{\frac{1}{2}} (b; fail)$.*

In order to be able to make the distinction between 'inside' and 'outside' alternatives the notion of an alternative is extended. The use of the extended alternatives for the statements in the example above is illustrated in example 4.4.54.

**Definition 4.4.52** *The collection Alt of alternatives, ranged over by $A$, is given by $A ::= \perp \mid S \rhd A \mid T \rhd A \mid *A$, where $\perp$, $S$, $T$ are as in definition 4.4.12.*

Compared to the definition of alternatives for $\mathcal{L}^{(3a)}_{pnd}$ the only addition is $*A$. The symbol $*$ indicates that the alternatives in $A$ are disabled because they are from outside the current parallel composition. A take-over action will be tried before options from $*A$.

**Definition 4.4.53** *For $i = l$ for the local interpretation of nondeterminism or $i = g$ for the global interpretation of nondeterminism and $j = u$ for the unconditional interpretation of probabilistic choice or $j = c$ for the conditional interpretation of probabilistic choice the mapping $\mathcal{O}_{i,j}\colon Res \to Alt \to \mathbb{P}_o$ is given as before (cf. definition 4.4.15 for* E *and for $s_0$*

*deterministic, nondeterministic or probabilistic. For $s_0$ failing, parallel or take-over it is given by*

$$\begin{array}{rcll}
\mathcal{O}_{i,j}(s_0)[\bot] & = & \{\, \Delta_\delta \,\} & s_0 \not\to \\
\mathcal{O}_{i,j}(s_0)[S \rhd A] & = & \bigcup\{\, \mathcal{O}_{i,j}(s)[(S \setminus s) \rhd A] \mid s \in S \,\} & s_0 \not\to \\
\mathcal{O}_{i,j}(s_0)[T \rhd A] & = & \oplus\{\!|\, \frac{\rho}{R(T)} \cdot \mathcal{O}_{i,j}(s)[(T \setminus \rho \cdot s) \rhd A] \mid \rho \cdot s \in T \,|\!\} & s_0 \not\to \\
\mathcal{O}_{i,j}(s_0)[*A] & = & \{\, \Delta_\delta \,\} & s_0 \not\to \\
 & & & \\
\mathcal{O}_{i,j}(s_0)[A] & = & \bigcup\{\, \mathcal{O}_{i,j}(s)[*A] \mid s_0 \xrightarrow{\pi} s \,\} & s_0 \xrightarrow{\pi} \\
 & & & \\
\mathcal{O}_{i,j}(s_0)[\bot] & = & \mathcal{O}_{i,j}(s_1)[\bot] & s_0 \xrightarrow{tt} s_1 \\
\mathcal{O}_{i,j}(s_0)[S \rhd A] & = & \bigcup\{\, \mathcal{O}_{i,j}(s)[(S \setminus s) \rhd A] \mid s \in S \,\} & s_0 \xrightarrow{tt} s_1 \\
\mathcal{O}_{i,j}(s_0)[T \rhd A] & = & \oplus\{\!|\, \frac{\rho}{R(T)} \cdot \mathcal{O}_{i,j}(s)[(T \setminus \rho \cdot s) \rhd A] \mid \rho \cdot s \in T \,|\!\} & s_0 \xrightarrow{tt} s_1 \\
\mathcal{O}_{i,j}(s_0)[*A] & = & \mathcal{O}_{i,j}(s_1)[A] & s_0 \xrightarrow{tt} s_1
\end{array}$$

*with $R(T) = \sum_{\rho \cdot s \in T} \rho$ and $\oplus : \mathcal{MP}_f([0,1] \times \mathbb{P}_o) \to \mathbb{P}_o$ given by*

$$\oplus\{\!|\, \rho_1 \cdot p_1, \ldots, \rho_n \cdot p_n \,|\!\} \quad = \quad \{\, \textstyle\sum_{k=1}^{n} \rho_k \mu_k \mid \mu_1 \in p_1, \ldots, \mu_n \in p_n \,\}$$

The well-definedness of $\mathcal{O}_{i,j}$ can be shown using the usual fixed point reasoning (cf definition 3.3.20, lemma 3.3.21). Compared with definition 4.4.15 the clauses for parallel statements and take-over statements are new and an extra case is added for the failing statements.

The parallel statement allows selection of any of the parallel alternatives. This corresponds to taking one of the parallel components as the primary component. Any alternatives that exist are disabled as they are from outside the parallel composition. These alternatives should only be used if all components of the parallel composition fail.

A take-over statement is similar to a failing statement because a take-over action is caused by failure of the primary component of the take-over statement. The definition of $\mathcal{O}_{i,j}$ is the same for a take-over statement as for a failing statement when there are alternatives available, i.e. for $S \rhd A$ and $T \rhd A$. The alternatives in $S$ or $T$ are from within the current component and can be used to avoid failure of the component. The other component does not need to take over the computation. If there are no alternatives available, i.e. for $\bot$ and $*A$, the behavior of a take-over statement is different from a failing statement. A failing statement will produce $\Delta_\delta$, i.e. failure with probability 1. A take-over statement will instead allow the other component to take over the computation. If there are disabled alternatives, i.e. $*A$, these alternatives will no longer be disabled after the take-over action. If failure occurs after the take-over action this means that both alternatives of the parallel composition fail and therefore the parallel composition as a whole may be replaced by an alternative from outside the parallel composition.

**Example 4.4.54** *The processes given by the operational model for the statements $(a \oplus_{\frac{1}{2}}$ fail$) \| b$ and $a \oplus_{\frac{1}{2}} ($fail$\| b)$ (see also example 4.4.51) can be found as follows:*

$$\begin{array}{l}
\mathcal{O}_{i,c}((a \oplus_{\frac{1}{2}} \mathit{fail}) \| b)[\bot] \\
\quad = \quad \mathcal{O}_{i,c}((a \oplus_{\frac{1}{2}} \mathit{fail}) \mathbin{|\!\!\lfloor} b)[*\bot] \cup \mathcal{O}_{i,c}(b \mathbin{|\!\!\lfloor} (a \oplus_{\frac{1}{2}} \mathit{fail}))[*\bot]
\end{array}$$

$$\begin{aligned}
&= (\tfrac{1}{2}\cdot\mathcal{O}_{i,c}(a \mathbin{\lfloor\!\lfloor} b)[\{\!|\,\tfrac{1}{2}\cdot(\mathit{fail} \mathbin{\lfloor\!\lfloor} b)\,|\!\}\rhd * \perp] \oplus \tfrac{1}{2}\cdot\mathcal{O}_{i,c}(\mathit{fail} \mathbin{\lfloor\!\lfloor} b)[\{\!|\,\tfrac{1}{2}\cdot(a \mathbin{\lfloor\!\lfloor} b)\,|\!\}\rhd * \perp]) \\
&\quad \cup \mathcal{O}_{i,c}(a \oplus_{\frac{1}{2}} \mathit{fail})[\perp]/b \\
&= (\tfrac{1}{2}\cdot\mathcal{O}_{i,c}(b)[\perp]/a \oplus \tfrac{1}{2}\cdot\mathcal{O}_{i,c}(a \mathbin{\lfloor\!\lfloor} b)[*\perp]) \cup \{\,\Delta_a/b\,\} \\
&= (\tfrac{1}{2}\cdot\{\,\Delta_{ab}\,\} \oplus \tfrac{1}{2}\cdot\{\,\Delta_{ab}\,\}) \cup \{\,\Delta_{ba}\,\} \\
&= \{\,\Delta_{ab}, \Delta_{ba}\,\}
\end{aligned}$$

$$\begin{aligned}
&\mathcal{O}_{i,c}(a \oplus_{\frac{1}{2}} (\mathit{fail}\|b))[\perp] \\
&\quad= \tfrac{1}{2}\cdot\mathcal{O}_{i,c}(a)[\{\!|\,\tfrac{1}{2}\cdot(\mathit{fail}\|b)\,|\!\}\rhd\perp] \oplus \tfrac{1}{2}\cdot\mathcal{O}_{i,c}(\mathit{fail}\|b)[\{\!|\,\tfrac{1}{2}\cdot a\,|\!\}\rhd\perp] \\
&\quad= \tfrac{1}{2}\cdot\{\,\Delta_a\,\} \oplus \tfrac{1}{2}\cdot(\mathcal{O}_{i,c}(\mathit{fail} \mathbin{\lfloor\!\lfloor} b)[*\{\!|\,\tfrac{1}{2}\cdot a\,|\!\}\rhd\perp] \cup \mathcal{O}_{i,c}(b \mathbin{\lfloor\!\lfloor} \mathit{fail})[*\{\!|\,\tfrac{1}{2}\cdot a\,|\!\}\rhd\perp]) \\
&\quad= \tfrac{1}{2}\cdot\{\,\Delta_a\,\} \oplus \tfrac{1}{2}\cdot(\mathcal{O}_{i,c}(b;\mathit{fail})[\{\!|\,\tfrac{1}{2}\cdot a\,|\!\}\rhd\perp] \cup \mathcal{O}_{i,c}(\mathit{fail})[\perp]/b) \\
&\quad= \tfrac{1}{2}\cdot\{\,\Delta_a\,\} \oplus \tfrac{1}{2}\cdot(\mathcal{O}_{i,c}(\mathit{fail})[\perp]/b \cup \{\,\Delta_{b\delta}\,\}) \\
&\quad= \{\,\tfrac{1}{2}\Delta_a + \tfrac{1}{2}\Delta_{b\delta}\,\}
\end{aligned}$$

*In the second program, the action $a$ is not available as an alternative for failure. If, however, the whole parallel composition fails, an alternative from outside the parallel composition can be used.*

$$\begin{aligned}
&\mathcal{O}_{i,c}(a \oplus_{\frac{1}{2}} (\mathit{fail}\|\mathit{fail}))[\perp] \\
&\quad= \tfrac{1}{2}\cdot\mathcal{O}_{i,c}(a)[\{\!|\,\tfrac{1}{2}\cdot(\mathit{fail}\|\mathit{fail})\,|\!\}\rhd\perp] \oplus \tfrac{1}{2}\cdot\mathcal{O}_{i,c}(\mathit{fail}\|\mathit{fail})[\{\!|\,\tfrac{1}{2}\cdot a\,|\!\}\rhd\perp] \\
&\quad= \tfrac{1}{2}\cdot\{\,\Delta_a\,\} \oplus \tfrac{1}{2}\cdot\mathcal{O}_{i,c}(\mathit{fail} \mathbin{\lfloor\!\lfloor} \mathit{fail})[*\{\!|\,\tfrac{1}{2}\cdot a\,|\!\}\rhd\perp] \\
&\quad= \tfrac{1}{2}\cdot\{\,\Delta_a\,\} \oplus \tfrac{1}{2}\cdot\mathcal{O}_{i,c}(\mathit{fail};\mathit{fail})[\{\!|\,\tfrac{1}{2}\cdot a\,|\!\}\rhd\perp] \\
&\quad= \tfrac{1}{2}\cdot\{\,\Delta_a\,\} \oplus \tfrac{1}{2}\cdot\mathcal{O}_{i,c}(a)[\perp] \\
&\quad= \{\,\Delta_a\,\}
\end{aligned}$$

*By a similar calculation one can find $\mathcal{O}_{g,u}((a \oplus_{\frac{1}{2}} \mathit{fail})\|\mathit{fail})[\perp] = \{\,\frac{1}{2}\Delta_{a\delta} + \frac{1}{2}\Delta_\delta\,\}$ while $\mathcal{O}_{g,u}(((a \oplus_{\frac{1}{2}} \mathit{fail}) \mathbin{\lfloor\!\lfloor} \mathit{fail}) \Box (\mathit{fail} \mathbin{\lfloor\!\lfloor} (a \oplus_{\frac{1}{2}} \mathit{fail})))[\perp] = \{\,\frac{3}{4}\Delta_{a\delta} + \frac{1}{4}\Delta_\delta\,\}$. The two alternatives $(a \oplus_{\frac{1}{2}} \mathit{fail}) \mathbin{\lfloor\!\lfloor} \mathit{fail}$ and $\mathit{fail} \mathbin{\lfloor\!\lfloor} (a \oplus_{\frac{1}{2}} \mathit{fail})$ both fail in the first step with probability $\frac{1}{2}$. Due to the global interpretation of the nondeterministic choice, the other option will be tried if this happens. As a result the probability of failure in the first step is $\frac{1}{2}\cdot\frac{1}{2}$. (See also example 4.4.16.)*

The last two statements in the example above show that the selection of the primary component in a parallel composition is a form of choice that cannot be described using a nondeterministic choice. In the statement $s' \Box s$ (see also example 4.4.16) the statements $s$ and $s'$ can be seen as 'separate resources' in that failure of $s$ will not affect $s'$. If nondeterministic choice is global, and both $s$ and $s'$ fail with probability $\frac{1}{2}$ then $s \Box s'$ only fails with probability $\frac{1}{4}$. The statement $s\|s'$ is a choice between $s \mathbin{\lfloor\!\lfloor} s'$ and $s' \mathbin{\lfloor\!\lfloor} s$. However, if the component $s$ fails, this not only affects $s \mathbin{\lfloor\!\lfloor} s'$ but also $s' \mathbin{\lfloor\!\lfloor} s$. If the component $s = a \oplus_{\frac{1}{2}} \mathit{fail}$ fails then even if another component $s'$ takes over, the probabilistic choice between $a$ and *fail* is already made. The alternative for $s \mathbin{\|\!\_} s'$ is not $s' \mathbin{\|\!\_} s$, as it is in $(s \mathbin{\|\!\_} s') \Box (s' \mathbin{\|\!\_} s)$, but $s' \mathbin{\|\!\_}$ *fail*. This difference shows the need for distinguishing between making a nondeterministic choice from choosing a primary component as is done by using separate labels, $\nu$ and $\pi$, in the transition system.

The operational semantics should give the meaning of programs. The model $\mathcal{O}_{i,j}$ gives the meaning of resumptions instead of programs and uses an extra argument containing alternatives. The operational semantics $\mathcal{O}_{i,j}[\![\bullet]\!]$ is obtained by restricting to programs and using the empty alternative $\perp$ for the extra argument.

**Definition 4.4.55** *The operational semantics $\mathcal{O}_{i,j}[\![\bullet]\!]\colon \mathcal{L}^{(3b)}_{pnd} \to \mathbb{P}_o$, for $i = l$ for the local interpretation of nondeterminism or $i = g$ for the global interpretation of nondeterminism and $j = u$ for the unconditional interpretation of probabilistic choice or $j = c$ for the conditional interpretation of probabilistic choice is given by*

$$\mathcal{O}_{i,j}[\![s]\!] = \mathcal{O}_{i,j}(s)[\perp]$$

For a statement from $\mathcal{L}^{(3a)}_{pnd}$ the transition tree given by $\mathcal{T}^{(3a)}_{pnd}$ is the same as that given by $\mathcal{T}^{(3b)}_{pnd}$ and the adjustments made to the definition of the operational models $\mathcal{O}_{i,j}$ are all caused by take-over and parallel steps. For a statement that does not contain any parallelism, and, therefore, no take-over, the definition is the same as definition 4.4.15. The operational models, and therefore also the operational semantics, of a statement from $\mathcal{L}^{(3a)}_{pnd}$ remains the same. In the theorem below the operational semantics for $\mathcal{L}^{(3a)}_{pnd}$ given in subsection 4.4.4 is denoted by $\mathcal{O}^{(fail)}_{i,j}[\![\bullet]\!]$ and the operational semantics from this subsection by $\mathcal{O}^{(par)}_{i,j}[\![\bullet]\!]$.

**Theorem 4.4.56** *The operational semantics of $\mathcal{L}^{(3b)}_{pnd}$ is a consistent extension of the operational semantics of $\mathcal{L}^{(3a)}_{pnd}$, i.e. $\mathcal{O}^{(par)}_{i,j}[\![s]\!] = \mathcal{O}^{(fail)}_{i,j}[\![s]\!]$ for all $s \in \mathcal{L}^{(3a)}_{pnd}$.*

A direct consequence of theorem 4.4.56 is that the results from example 4.4.18 also hold for $\mathcal{L}^{(3b)}_{pnd}$.

**Corollary 4.4.57** *The four operational semantics $\mathcal{O}_{i,j}[\![\bullet]\!]$ with $i = l$ for the local interpretation of nondeterminism or $i = g$ for the global interpretation of nondeterminism and $j = u$ for the unconditional interpretation of probabilistic choice or $j = c$ for the conditional interpretation of probabilistic choice each have different distinguishing power.*

### 4.4.8 Bisimulation for $\mathcal{L}^{(3b)}_{pnd}$

As for $\mathcal{L}^{(3a)}_{pnd}$, a denotational model can be given but is quite uninformative. A notion of bisimulation is given instead. As in subsection 4.4.5 we aim for a notion of bisimulation that extends the familiar notions of strong bisimulation [171, 159] and probabilistic bisimulation [149]. The notion of bisimulation depends on the interpretation of the nondeterministic and probabilistic choice. In this way we can have that, for example, $a \Box$ *fail* is bisimilar with $a$ for the global models but not for the local models. The approach of subsection 4.4.5 is extended by interpreting the statements as statements of $\mathcal{L}^{(3b)}_{pnd}$ instead of $\mathcal{L}^{(3a)}_{pnd}$ and possibly by adding new cases which are similar to the existing cases.

As in subsection 4.4.5 the transition relation has to be lifted to configurations in $Res \times Alt$ to be able to define the first step relation.

**Definition 4.4.58** *The lifted transition relation $\rightarrow\ \subseteq Conf \times Lab \times Conf$ is given by (with $A$ any alternative in Alt):*

*(a) As before (definition 4.4.19) if $s$ is deterministic, nondeterministic, probabilistic or failing.*

*(b) If $s$ is parallel then for all $s'$ with $s \xrightarrow{\pi} s'$ we have*

$$s \times A \xrightarrow{\nu} s' \times (*A)$$

*(c) If $s$ is overtaking, $s \xrightarrow{tt} s'$, then*

$$\begin{array}{ll} s \times \bot \xrightarrow{\theta} t & \textit{if } s' \times \bot \xrightarrow{\theta} t \\ s \times (S \triangleright A) \xrightarrow{\nu} s'' \times ((S \backslash s'') \triangleright A & \textit{for all } s'' \in S \\ s \times (T \triangleright A) \xrightarrow{\rho/R(T)} s'' \times ((T \backslash \rho \cdot s'') \triangleright A) & \textit{for all } \rho \cdot s'' \in T \\ s \times (*A) \xrightarrow{\theta} t & \textit{if } s' \times A \xrightarrow{\theta} t \end{array}$$

*where $R(T) = \sum_{\rho \cdot s_0 \in T} \rho$. Note that $\theta$ ranges over all labels, but only labels in $Act \cup \{\nu\} \cup (0,1)$ can actually occur.*

The lifted transition relation $\rightarrow$ assumes a fixed interpretation of nondeterministic choice and probabilistic choice. With this fixed interpretation in mind we write $\mathcal{O}$ for the operational model instead of $\mathcal{O}_{i,j}$.

Unlike the transition relation given in $\mathcal{T}^{(3b)}_{pnd}$, the lifted transition relation does not need to distinguish between nondeterminism caused by nondeterministic choice and nondeterminism caused by parallelism. Resolving either type of nondeterminism is modeled by a $\nu$ step. The fact that the options from outside the parallel composition need to be disabled can be incorporated in the configuration and no longer need to be signaled by a special label $\pi$.

The effect of a take-over is also dealt with in the lifted transition system. The take-over step $tt$ is no longer needed: If there are alternatives available, these will be taken. If there are no alternatives available as in $s \times \bot$ or $s \times *A$ the steps of the statement after take-over $s' \times \bot$ or $s' \times A$ can be taken directly.

Showing well-definedness of the lifted transition relation is straightforward by using weight induction with the weight function $wgt_c$ given below.

**Definition 4.4.59** *The function $wgt_c : Stat \times Alt \rightarrow \mathbb{N}$ is defined using the function $wgt : Stat \rightarrow \mathbb{N}$ given in definition 4.4.49 and the auxiliary function $wgt_{Alt} : Alt \rightarrow \mathbb{N}$. The auxiliary function $wgt_{Alt}$ extends $wgt_{Alt}$ as given in definition 4.4.23.*

$$\begin{array}{rcl} wgt_c(s \times A) & = & wgt(s) + wgt_{Alt}(A) \\ wgt_{Alt}(*A) & = & wgt_{Alt}(A) \end{array}$$

When $s \xrightarrow{tt} s'$ then the weight $wgt(s)$ of $s$ is greater than the weight of $s'$. As a result the inequalities

$$\begin{array}{rcl} wgt_c(s \times \bot) & > & wgt_c(s' \times \bot) \\ wgt_c(s \times *A) & > & wgt_c(s' \times A) \end{array}$$

hold, justifying the first and last clause of part (c) of definition 4.4.58 above.

The first step relation for configurations based on $\mathcal{L}^{(3b)}_{pnd}$ is defined as the first step relation on $\mathcal{L}^{(3a)}_{pnd}$ (see definition 4.4.20) except that the extended configurations in $\mathit{Conf}'$ now refer to extended configurations for $\mathcal{L}^{(3b)}_{pnd}$ instead of for $\mathcal{L}^{(3a)}_{pnd}$.

The semantics of an extended configuration is also defined literally as for $\mathcal{L}^{(3a)}_{pnd}$ in definition 4.4.22, but now with statements, alternatives and first steps as in this section.

The relationship between the semantics of configurations $s \times A$ and the operational semantics of $s$ given $A$ is again clear by weight induction.

**Lemma 4.4.60** *For $s \in \mathcal{L}^{(3b)}_{pnd}$ and $A \in \mathit{Alt}$: $\mathcal{O}(s)[A] = \bigcup\{\, \overline{\mathcal{O}}'(R) \mid s \times A \rightsquigarrow R\,\}$.*

With the first step relation in place, bisimulation (on configurations and statements) can be defined as expected.

**Definition 4.4.61** *Bisimulation on Conf is the greatest equivalence relation on Conf satisfying*

$$\begin{array}{rcl} t_1 \sim t_2 & \iff & \text{if } t_1 \rightsquigarrow R_1 \text{ then } t_2 \rightsquigarrow R_2 \text{ and } R_1 \sim_{fs} R_2 \text{ for some } R_2 \text{ and} \\ & & \text{if } t_2 \rightsquigarrow R_2 \text{ then } t_1 \rightsquigarrow R_1 \text{ and } R_1 \sim_{fs} R_2 \text{ for some } R_1. \end{array}$$

*Two statements $s, s' \in \mathcal{L}^{(3b)}_{pnd}$ are bisimilar if $s \times A \sim s' \times A$ for each alternative $A \in \mathit{Alt}$.*

The above definition can be justified as usual. Correctness of the bisimulation relation $\sim$ on $\mathcal{L}^{(3b)}_{pnd}$ with respect to $\mathcal{O}$ can be obtained from the correctness of the first step relation along the same lines as in subsection 4.4.5.

The congruence results obtained for the operators of $\mathcal{L}^{(3a)}_{pnd}$ carry over to the present setting of $\mathcal{L}^{(3b)}_{pnd}$. For sequential composition we considered in subsection 4.4.5 the derived notion of prefixing and of conditional postfixing. (These restrictions where triggered by the interplay between nondeterminacy on the one hand and probability on the other.) However, for the merge operator neither component is a prefix. In a parallel construct both components can become primary, and, consequently, both components may initiate the computation. Therefore a full congruence result cannot be expected. The following example shows that the parallel operator is 'too nondeterministic' to obtain even a conditional congruence result similar to that for sequential composition. Adding a parallel component $s'$ to a statement $s$ has the effect, among others, that the first step of $s$ can postfixed with a nondeterministic combination of the remaining steps of $s$ and the steps of $s'$.

**Example 4.4.62** *Consider the statements $(a;b)\|c$ and $((a \oplus_{\frac{1}{2}} a);b)\|c$. In this example we choose for unconditional probabilistic choice and omit the alternative from the notation. (The alternative is always empty.)*

*It is easy to check that $\mathcal{O}[\![(a;b)\|c]\!] = \{\, \Delta_{abc}, \Delta_{acb}, \Delta_{cab} \,\}$. However,*

$$\begin{aligned}
\mathcal{O}[\![((a \oplus_{\frac{1}{2}} a);b)\|c]\!] &= \mathcal{O}(((a \oplus_{\frac{1}{2}} a);b) \mathbin{\|\!\!\lfloor} c) \cup \mathcal{O}(c \mathbin{\|\!\!\lfloor} ((a \oplus_{\frac{1}{2}} a);b)) \\
&= \tfrac{1}{2} \cdot \mathcal{O}((a;b) \mathbin{\|\!\!\lfloor} c) \oplus \tfrac{1}{2} \cdot \mathcal{O}((a;b) \mathbin{\|\!\!\lfloor} c) \cup \mathcal{O}((a \oplus_{\frac{1}{2}} a);b)/c \\
&= \tfrac{1}{2} \cdot \{\, \Delta_{abc}, \Delta_{acb} \,\} \oplus \tfrac{1}{2} \cdot \{\, \Delta_{abc}, \Delta_{acb} \,\} \cup \{\, \Delta_{cab} \,\} \\
&= \{\, \Delta_{abc}, \Delta_{acb}, \Delta_{cab}, \tfrac{1}{2}\Delta_{abc} + \tfrac{1}{2}\Delta_{acb} \,\}
\end{aligned}$$

The statements $a;b$ and $(a \oplus_{\frac{1}{2}} a);b$ are bisimilar but $(a;b)\|c$ and $((a \oplus_{\frac{1}{2}} a);b)\|c$ are not, because their semantics differ. This show that even parallel composition with the simple statement $c$ can destroy the bisimilarity. The point is that parallel composition introduces non-determinism not only in the first step, but at every step while both its components are running. In contrast, the nondeterministic choice operator induces nondeterminism at the first step only.

In the key paper [149] a notion of probabilistic bisimulation is introduced for a probabilistic process language including synchronous product as parallel operator. Larsen-Skou bisimulation relates two statements exactly when the probability to take a step and to end up in any set of statements closed under probabilistic bisimulation, is equal for both statements.

**Theorem 4.4.63** *Let the sublanguages $\mathcal{L}_{nd}$ and $\mathcal{L}_p$ of $\mathcal{L}^{(3b)}_{pnd}$ be given by $s ::= a \mid x \mid s;s \mid \square_{i=1}^{n} s_i \mid s\|s$ and $s ::= a \mid x \mid s;s \mid \oplus_{i=1}^{n} \rho_i \cdot s_i$, respectively. Two statements $s, s' \in \mathcal{L}_{nd}$ are strongly bisimilar iff $s \sim_{\mathrm{fs}} s'$. Two statements $s, s' \in \mathcal{L}_p$ are probabilistically bisimilar in the sense of [149] iff $s \sim_{\mathrm{fs}} s'$.*

On the probabilistic part $\mathcal{L}_p$ our notion of first step bisimulation specializes to probabilistic bisimulation. This is clear from the definition of fs-bisimulation when one considers that all statements in $\mathcal{L}_p$ are splittable and have exactly one first step. Note that example 4.4.62 above shows that it is not possible to obtain a notion of bisimulation on $\mathcal{L}^{(3b)}_{pnd}$ that is correct with respect to $\mathcal{O}$, a congruence for $\|$ and which extends Larsen-Skou bisimulation.

Focusing on the non-probabilistic part $\mathcal{L}_{nd}$ we recover from fs-bisimulation the familiar notion of strong bisimulation (cf. [171, 159]). This is because for statements in $\mathcal{L}_{nd}$ the first steps always consist of exactly one element, as can be established straightforwardly by weight-induction. A further analysis of the first step $s \rightsquigarrow \{\!|\, 1 \cdot \langle a, r \rangle \,|\!\}$, for $s \in \mathcal{L}_{nd}$, shows that the first step relation $\rightsquigarrow$ coincides with the standard transition system, i.e. $s \rightsquigarrow \{\!|\, 1 \cdot \langle a, r \rangle \,|\!\}$ precisely when $s \xrightarrow{a} r$. Equality of fs-bisimulation and strong bisimulation on the nondeterministic fragment of $\mathcal{L}^{(3b)}_{pnd}$ then follows immediately.

## 4.5 Conclusions and bibliographical remarks

In this chapter three different settings for the combination of nondeterminism and probability have been studied. All have their advantages and disadvantages. The main conclusion that can be drawn is that the issue of combining nondeterminism and probability is complex and that there does not exist one model for all cases. Instead the choice of a model has to be made depending on the interpretation of the nondeterminism.

The models in section 4.2 and section 4.3 gave priority to nondeterminism respectively probability. In section 4.4 nondeterminism and probability were treated 'on an equal level'. The notion of first step bisimulation introduced in section 4.4 extends the well known notions of strong bisimulation for nondeterministic processes and probabilistic (Larsen-Skou) bisimulation for probabilistic processes. It was also shown that no relation on programs which is a congruence relation can be correct with respect to the operational semantics and extend both strong and probabilistic bisimulation. Indeed, the notion of first step bisimulation is only a conditional congruence for sequential composition.

The operational model of section 4.4 distinguishes many statements, to allow as many interpretations of the nondeterministic choice as possible. For example the statements $(a \oplus_{\frac{1}{2}} b) \Box (a \Box b)$ and $a \Box b$ are not identified. If the nondeterministic choice, for example, is known to be consistent given some additional information then these statements can indeed be distinguished. This can be the case if nondeterministic choice is used for underspecification. For the first statement, it would still be possible for different outcomes to occur in multiple runs, while in the second statement this is not possible. If, however, the additional assumption is made that one can only see the worst and best case probabilities for events then it is possible to identify these statements. Making this assumption imposes a restriction on the type of nondeterministic phenomena that can be modeled with the nondeterministic choice, but does have the advantage that by using the extra identification allowed by this assumption, the notion of first step bisimulation becomes a full congruence (as all statements become splittable). More work, not reported in this thesis, needs to be done on the instantiation of the setting of this chapter (especially section 4.4) with specific types of nondeterminism for which such extra assumptions can be made.

The approach of [181] is similar: Two different probabilistic automata can be given describing $(a \oplus_{\frac{1}{2}} b) \Box (a \Box b)$ and $a \Box b$ respectively. However, in the verification of properties only worst and best case probabilities are considered and the two automata will satisfy exactly the same properties. Note that for the approach of section 4.3 and other alternating approaches which first resolve the probability (see e.g. [8]) the programs $a \Box b$ and $(a \oplus_{\frac{1}{2}} b) \Box (a \Box b)$ are always the same. The first program offers the resources $a$ and $b$. Additionally offering either $a$ or $b$ with probability $\frac{1}{2}$ each does not change anything as both are already available.

More further work consists of the addition of synchronization to the parallel composition of language $\mathcal{L}_{pnd}^{(3b)}$ of section 4.4. In this setting many different possible assumptions on how deadlock is avoided are possible and give different results. For example the strategy 'directly use an alternative if synchronization is not possible' leads to different probabilities than the strategy 'use an alternative only when the system is deadlocked.' Practical examples should be studied to see which assumptions are reasonable in which settings. Another problem is the question of the status of choices after backtracking: If a choice has been made but needs to be retracted because of a failure to synchronize then what should happen with the choice if another component selects an alternative. Should the choice use the same outcome or can it be made again? The safest approach seems to be to allow as much freedom as possible in the resolving of nondeterministic choices, i.e. allow each choice to be remade. In this way the most nondeterministic options are generated. Correctness in this model thus guarantees correctness for cases where there is less freedom for the nondeterministic choice. If additional assumptions can be made, an identification of processes as indicated above can again be used to obtain additional identifications.

The work in this chapter extends the work done in chapter 3 by adding the construct of nondeterminism. Quite some work on the combination of nondeterminism and probability exists. Most of this work, however, concentrates on a single class or type of nondeterminism and does not provide the extensive overview of different modelings for different types of nondeterminism presented here.

As explained above three main approaches are considered. The first approach, given in section 4.2 is suited for interpretation of nondeterminism as a choice for an opponent or adversary. This approach is based on the report [107].

The probabilistic automata treated by Segala in his thesis [181] also combine probability and nondeterminism. For the verification of properties of probabilistic automata the nondeterminism present in the automata is assumed to be resolved by some malicious adversary, which chooses the worst possible option for the given property. Restricted adversaries that have only partial information or which must satisfy some constraints such as e.g. fairness conditions are also considered. The verification then becomes a matter of checking the probabilities for the desired executions in a completely probabilistic system.

The probabilistic automata used by Segala fit in the interpretation of nondeterminism given in section 4.2. Other work that deals with the combination of nondeterminism and probability that falls into this category can be found in [142, 154]. The model in [142] uses a reactive form of probabilistic choice, the probabilistic choice is between end states after the execution of an action. Nondeterminism is present in the selection of which action to execute. Each nondeterminism option, however, must have a different starting action. In [154] Lowe treats reactive probabilistic and nondeterministic processes. Although the transition systems obtained are closer to those given in section 4.3, the trace semantics given fits within the setting of section 4.2. Besides this trace semantics several other semantical models are also given in [154].

The second approach, considered in section 4.3, is suited for an available resources interpretation of nondeterministic choice. This approach is also based on [107]. Work done by Andova on the probabilistic extension of process algebra in [10, 8, 9] fits in this category. This work studies process algebras for languages with both probabilistic choice and nondeterminism. The operational meaning of programs, used to justify the axioms of the process algebra, is captured by giving transition systems using the same strategy as in section 4.3: All probabilistic choices are resolved to find which nondeterministic alternatives are available. A difference with the approach of section 4.3 is that the probabilities are not included in the transition system but calculated separately. The paper [19] deals with process algebras for different interpretations of nondeterministic choice in a setting without probability, showing that different interpretations of nondeterministic choice are already present in this setting.

In [200] may and must testing, i.e. worst and best case behavior, is studied for processes in an extension of CCS which contains probabilistic choice as well as nondeterminism in the form of nondeterministic choice and parallel composition. In the nondeterministic or parallel composition of two programs, the probability is synchronously resolved until one of the processes can produce an action. This action is then taken. This is similar to the approach of section 4.3 except that in section 4.3 the probability in both programs is first resolved completely before deciding an action.

The final approach, considered in section 4.4, does not enforce a specific interpretation of nondeterminism. However, due to the form of the processes obtained, it is closer to the first approach than to the second. This approach is based on [109, 110].

The transitions systems used in sections 4.2 and 4.3 can be characterized as alternating models [103, 105]. Nondeterminism and probability are resolved in two distinct stages after which an action is produced. The transition system in section 4.4 is not alternating; an arbitrary sequence of nondeterministic and probabilistic choices can be taken before an action is produced.

In [23] Baier treats fully probabilistic systems as well as nondeterministic and probabilistic systems, which are referred to as concurrent probabilistic systems. The process language PCCS with reactive probabilistic choice, nondeterministic choice and interleaving parallel composition, the language PSCCS with generative probabilistic choice and synchronize product and the language PSLCCS with a less restrictive form of synchronize product are studied. Transition systems are given which use finite probability distributions to describe the probabilistic steps and an alternating approach when nondeterminism is also present.

The work of Mislove [161] also deals with the combination of probabilistic choice and nondeterministic choice. In this work a different point of view is taken: A set of equational laws which should hold is taken as the starting point. A domain theoretic approach is then used to find models that satisfy these laws.

To deal with nondeterministic and probabilistic systems Jonsson and Larsen [135] introduce transition systems where sets of probabilities are used as labels to indicate the possible probabilities for taking this transition. The work of Seidel in [183] exploits the measure theoretical apparatus of stochastic kernels, referred to as conditional probability measures in [183], for the modeling of CSP-style parallelism and synchronization. A model using measures instead of kernels is developed for a purely probabilistic sublanguage. An axiom system is also given and a self stabilizing token ring is treated as an example.

In section 4.4, a notion of bisimulation is defined rather than giving a denotational model. For nondeterministic systems strong or Park-Milner bisimulation [171, 159] is a standard notion. The most standard definition for bisimulation on probabilistic systems is due to Larsen and Skou [149].

The papers [190] exploits the general coalgebraic approach of [179] to show that probabilistic bisimulation of Larsen and Skou coincides with equality in a denotational domain. In [23] denotational domains specified by domain equations, using either a metric or a complete partial order approach, are investigated and also shown to capture bisimulation.

Algorithms and tools have been developed to actually calculate probabilistic bisimulation equivalence classes [24, 122, 185, 26]. Probabilistic process algebra [8, 21, 9] also uses probabilistic bisimulation as the notion of equivalence of processes. The process algebra provides a way to use equational reasoning to obtain equivalence of probabilistic processes, thus finding equivalences without having to find the semantics of the processes or calculating the actual bisimulation relation.

Extensions of probabilistic bisimulation, to deal with language constructs not considered in [149], have also been studied [33, 182, 191]. In [76] an extension of probabilistic bisimulation to continuous probabilities is reported. In [190] another definition of bisimulation is given that extends probabilistic bisimulation and can deal with non-discrete probabilistic choice.

In [26, 23] a notion of weak bisimulation for purely probabilistic processes is treated. Another notion of weak bisimulation for probabilistic processes, which is designed to be extendable with nondeterminism, is given in [11]. The paper [172] introduced a notion of weak bisimulation for probabilistic and nondeterministic systems by using schedulers to resolve the nondeterminism. In this thesis no results on weak bisimulation are reported.

# Chapter 5

# Action Refinement

## 5.1 Introduction

In this chapter we study the combination of action refinement and probabilistic choice. A language $\mathcal{L}_{pr}$ is introduced which contains both these constructs. First, however, action refinement is treated in a setting without probability by studying the language $\mathcal{L}_{ref}$. The language $\mathcal{L}_{ref}$ contains, besides action refinement, the basic constructs recursion, sequential composition, nondeterministic choice and parallel composition. The language $\mathcal{L}_{pr}$ includes probabilistic choice instead of the nondeterministic constructs nondeterministic choice and parallel composition. In the treatment of action refinement the metric techniques are exploited both in the definition of the operational semantics and the denotational semantics as well as in the comparison of these models. The strong similarity between the treatment of $\mathcal{L}_{ref}$ and $\mathcal{L}_{pr}$ shows that probability and action refinement can be treated orthogonally in the metric setting. Also the applicability of compact support measures to this setting is shown.

In section 5.2 both an operational and a denotational model for the language $\mathcal{L}_{ref}$ are given in the *interleaving* framework. The interpretation of action refinement is non-atomic. It is shown that the denotational semantics is fully abstract with respect to the operational one. It follows that the notion of action refinement in itself, as was not perceived so far, does not enforce a truly concurrent interpretation.

Previously, it has been argued by several authors that interleaving semantics is not the appropriate model when dealing with non-atomic action refinement. For example, in [56] it is noted that in an interleaving interpretation the two statements $a \| b$ and $a; b \,\square\, b; a$ have the same meaning, viz. $\{\, ab, ba \,\}$, whereas, when the action $a$ is refined to the sequential composition $a_1; a_2$, the two respective refinements $(a \| b)\langle a \rightsquigarrow a_1; a_2 \rangle$ and $(a; b \,\square\, b; a)\langle a \rightsquigarrow a_1; a_2 \rangle$ have different meanings, viz. $\{\, a_1 a_2 b, a_1 b a_2, b a_1 a_2 \,\}$ and $\{\, a_1 a_2 b, b a_1 a_2 \,\}$, respectively. The point is that for $(a \| b)\langle a \rightsquigarrow a_1; a_2 \rangle$ it is allowed for $b$ to be scheduled in between $a_1$ and $a_2$; for $(a; b \,\square\, b; a)\langle a \rightsquigarrow a_1; a_2 \rangle$ the action $b$ will be taken either before or after both of $a_1$ and $a_2$. Consequently, Castellano et al. conclude that no compositional interleaving model exists for a language combining the operators of sequential, alternative and parallel composition used in the statements above with the notion of action refinement. Section 5.2 argues that this point of view might be reconsidered: We shall propose to consider two programs semantically equivalent whenever their meanings coincide *under all refinements*, be it syntactic refinements for the operational semantics, or semantic refinements for the denotational semantics. We feel that, in the framework of schematic (or process-description) languages customarily employed to study concurrency semantics, the fact that the elementary actions are uninterpreted induces as a natural counterpart that the semantic equivalence of two programs requires equality of their associated meanings under all interpretations of the elementary actions, or equivalently, under all their possible refinements.

The operational semantics of $\mathcal{L}_{ref}$ yields sets of sequences of actions and is based on a transition system. In the configurations of the transition system a stack of refinements is maintained. This component plays the role of a *syntactic* refinement sequence. The denotational model for $\mathcal{L}_{ref}$ employs the notion of a *semantic* refinement. An auxiliary argument of the semantic function mirrors the syntactic refinement sequence used for the operational semantics. The denotational semantics also yields sets of sequences of actions, but with a semantical refinement as an additional argument. Two statements are identified by this compositional model exactly when their meanings coincide for all semantic refinements. The denotational model distinguishes more programs than the operational model. However, the denotational model is fully abstract with respect to the operational one: The denotational model only distinguishes statements that need to be distinguished to be able to have a compositional model that is correct with respect to the operational model.

The parallel composition in $\mathcal{L}_{ref}$ does not allow communication between parallel components. In [112] an extension of $\mathcal{L}_{ref}$ is treated that does allow for this communication. The communication takes the form of synchronization of parallel components by the simultaneous execution of special, synchronization actions. The same communication method was used in sections 4.2 and 4.3. As done here for the language $\mathcal{L}_{ref}$, an operational semantics and a denotational semantics are given in [112] based on sequences of actions. The denotational model is built on failure sets (see also [178, 38, 50]) and is shown to be fully abstract with respect to the operational model. Due to the need for the use of failure sets in the denotational model, the definition of the denotational model and the full abstractness proof are much more technically involved than the corresponding items for the language $\mathcal{L}_{ref}$ presented here.

In section 5.3 the results of section 5.2 are extended to the language $\mathcal{L}_{pr}$. An operational and a denotational model are given for $\mathcal{L}_{pr}$ and the denotational model is again shown to be fully abstract with respect to the operational one. Unlike the denotational domains used in chapter 3 and chapter 4 the domain used by the denotational model for $\mathcal{L}_{pr}$ is a linear domain. To allow the definition of the denotational model on the linear domain the sequential composition of measures on sequences of actions is defined. This composition is defined specifically for compact support measures over action sequences. A more general form of composition of measures is treated in chapter 7.

The similarity of the definitions and the proofs used in sections 5.2 and 5.3 shows that the metric machinery is unaffected by the addition of probabilistic choice. The metric approach makes the problem of adding action refinement and obtaining full abstractness results orthogonal to other issues like adding probabilistic choice to the language.

The language $\mathcal{L}_{pr}$ does not contain the constructs of nondeterministic choice and parallel composition. There is no intrinsic need to remove the nondeterministic constructs from the language $\mathcal{L}_{ref}$ when adding probabilistic choice. However, combining probabilistic choice and action refinement, as discussed in section 5.3, on the one hand and combining probabilistic choice and nondeterminism, as discussed in chapter 4, on the other hand seem to be orthogonal issues which can be dealt with separately. Note that the denotational models given in chapter 4 are not fully abstract with respect to the corresponding operational models. An extension with action refinement similar to the work done in section 5.3 is unlikely to change this. For a full abstractness result like theorem 5.3.27 one should start with a different denotational model.

Some related investigations dealing with full abstractness for action refinement are given below. In a chapter of the handbook of process algebra [97] devoted to action refinement, Gorrieri and Rensink provide an extensive discussion of action refinement and the work done on this subject. A comprehensive overview of further related work may be found there. An early result concerning full abstraction and action refinement is the work of Nielsen, Engberg and Larsen [165]. Their work is based on a special kind of series-parallel pomsets and falls in the linear-time true concurrency framework, since operationally at a given moment in time several actions may be observed simultaneously. In the dissertational work of Engberg [80] additionally adequate logics and axiomatizations are discussed in this context. A minor difference, related to the other aspects stressed there, is the usage of set-based pre-orders and the role played by termination.

In [192, 193] a full abstractness result is obtained for safe Petri nets. In this work Vogler takes language equivalence as a starting point and uses a notion of interval semiwords, ordered structures in which 'timing information' of actions can be encoded. The resulting notion of equivalence is finer than step semantics and yields the coarsest congruence contained in failure semantics. Also this research falls in the true concurrency framework. In [194] the results are extended to event structures with silent moves, in particular for history-preserving bisimulation. Related work in the setting of Petri-nets is the full abstractness result of Jategaonkar and Meyer [132] with respect to testing-equivalence.

Other semantical investigations for action refinement, driven by concrete process languages are, e.g., that of Aceto and Hennessy (cf. [1, 2]) and of Gorrieri et al. [95, 74]. The former is based on a syntactic interpretation of action refinement, a point of view not adopted here. In the approach considered here the syntactic interpretation applies to the operational semantics only, whereas in the denotational semantics action refinement is handled using so-called semantic refinements. (For a discussion on the relationship between syntactic and semantic action refinement we refer to [94] where also sufficient conditions are proposed for the coincidence of these notions.) The work of Gorrieri considers atomized action refinement in an interleaving setting on the one hand (an approach followed in [37] as well, see also [106]) and, on the other hand, arbitrary action refinement exploiting the causal trees of Darondeau and Degano [72, 73], thus taking place in the true concurrency framework. Other key differences between [37] and the work presented in section 5.2, both following the interleaving approach, are

– [37] is not based on the 'equivalence under all refinements' approach; moreover, both the transition system for $\mathcal{O}$, and the definitions of the semantic operators for $\mathcal{D}$, are radically different

– [37] uses a branching (or 'bisimulation') model for the denotational semantics; below, linear denotational models are used

– [37] has a considerably more complicated proof of the correctness of $\mathcal{D}$ versus $\mathcal{O}$ for the linear case

– [37] does not feature a full abstractness result.

Further work on the semantics of action refinement in a true concurrency setting includes [63] in the context of LOTOS, and [176] based on Rensink's PhD thesis [175]. Further applications of metric techniques for true concurrency semantics, though not dealing with action refinement, can be found in [40, 152, 30].

Another body of the extensive literature related to action refinement deals with language independent semantical equivalences. A comprehensive taxonomy of semantical equivalences, referred to as the linear-time branching-time spectrum is proposed by Van Glabbeek et al. (cf. [88, 89, 90]). In the context of this chapter the linear-time notions of $split_n$ and $ST$-bisimulation are relevant. Split semantics goes back at least to [115]; two statements or structures are identified if they cannot be distinguished by splitting up their actions in sequences of actions up to length $n$. In $ST$-bisimulation, as introduced in [93], the current state of a process distinguishes between the actions that have been completed and the actions that have been started but not completed as yet. (See [193, 90] for a further discussion of these notions.) In the denotational model we propose here we quantify over all semantic refinements. Hence the resulting semantical equivalence is at least as fine as $split_n$-semantics. In [96] and [195] the limit of $split_n$-equivalence is studied in a branching time and language-equivalence setting, respectively. Vogler conjectures in [195] that 'on the level of failure semantics full abstractness for splitting is different from full abstractness for general action refinement'. The results presented in [112] (see discussion above) support this conjecture. For the language with synchronization general refinements are needed for the construction of the critical context which is used to obtain the full abstractness result.

## 5.2 An interleaving model for action refinement

This section provides an introduction to action refinement by treating the language $\mathcal{L}_{ref}$. As mentioned in the introduction to this chapter, the language $\mathcal{L}_{ref}$ does not contain probabilistic constructs but only action refinement and the basic constructs of recursion, sequential composition, nondeterministic choice and parallel composition. This section also illustrates the advantage of exploiting the metric machinery underlying the semantical models to obtain a full abstractness result. The main idea is the following: the denotational semantics $\mathcal{D}$ has functionality $\mathcal{D}\colon Stat \to SemRef \to \mathbb{P}_d$, where $Stat$ is the syntactic class of statements, $SemRef$ is the collection of semantical refinements given by $SemRef = Act \to \mathbb{P}_d$, i.e. the mappings from actions to processes, and $\mathbb{P}_d$ is a complete metric space of sets of sequences of actions. Now, if two statements $s_1$ and $s_2$ differ according to the denotational semantics, then —since denotations are elements of a metric space— the distance $d(\mathcal{D}(s_1)(\eta), \mathcal{D}(s_2)(\eta))$ differs from 0, for some particular semantical refinement $\eta$. We then can construct from $\eta$, using metric considerations, a *finitary* semantical refinement $\eta'$ such that $d(\mathcal{D}(s_1)(\eta'), \mathcal{D}(s_2)(\eta')) \neq 0$. This function $\eta'$ is finitary in the sense that $\eta'$ is nontrivial for finitely many actions $a$ only and for these actions, $\eta'(a)$ is a finite set of finite sequences. The special property of such a finitary $\eta'$ is that it corresponds to a finite sequence of syntactical refinements $\langle a_i \rightsquigarrow s'_i \rangle_{i=1}^k$. By the correctness of the denotational model $\mathcal{D}$ with respect to the operational model $\mathcal{O}$, we then obtain that $d(\mathcal{O}[\![ s_1 \langle a_i \rightsquigarrow s'_i \rangle_{i=1}^k ]\!], \mathcal{O}[\![ s_2 \langle a_i \rightsquigarrow s'_i \rangle_{i=1}^k ]\!]) \neq 0$, or, equivalently $\mathcal{O}[\![ s_1 \langle a_i \rightsquigarrow s'_i \rangle_{i=1}^k ]\!] \neq \mathcal{O}[\![ s_2 \langle a_i \rightsquigarrow s'_i \rangle_{i=1}^k ]\!]$. Hence the context $(\cdot)\langle a_i \rightsquigarrow s'_i \rangle_{i=1}^k$ is a context in which the statements $s_1$ and $s_2$ yield different observational behavior.

The layout of the remainder of this section is as follows: Subsection 5.2.1 introduces the language $\mathcal{L}_{ref}$ and its transition system and related operational model. In subsection 5.2.2 the denotational semantics for $\mathcal{L}_{ref}$ is constructed employing the notion of a semantic refinement. Subsection 5.2.3 treats the comparison of the operational and denotational model and presents the full abstractness result.

### 5.2.1 Syntax and operational semantics

In this subsection the syntax for the language $\mathcal{L}_{ref}$ with action refinement is given. Using configurations which can store so called refinement sequences, the notion of action refinement can be captured intuitively by a transition system. The operational semantics is based on the transition system. As usual a set of atomic actions *Act* ranged over by $a$ is used to describe the basic steps of the computation. Action refinement may be used to further specify the behavior of an action but otherwise the actions remain without any further interpretation. The set *Act* is assumed to be infinite. (The full abstractness result presented in subsection 5.2.3 requires that we can always pick a fresh action outside some given finite set of actions.) A set of procedure variable *PVar* ranged over by $x$ is used for recursion.

**Definition 5.2.1**

*(a) The set of statements Stat, ranged over by $s$, is given by*

$$s \quad ::= \quad a \mid x \mid s\,;s \mid s \,\Box\, s \mid s \| s \mid s\langle a \rightsquigarrow s \rangle$$

*(b) The set of guarded statements GStat, ranged over by $g$, is given by*

$$g \quad ::= \quad a \mid g\,;s \mid g \,\Box\, g \mid g \| g \mid g\langle a \rightsquigarrow g \rangle$$

*(c) The set of declarations Decl, ranged over by $D$, is given by*

$$Decl \quad = \quad PVar \to GStat$$

*(d) The language $\mathcal{L}_{ref}$ is given by*

$$\mathcal{L}_{ref} \quad = \quad Decl \times Stat$$

The actions in *Act* represent the basic steps of the computation and are left uninterpreted except that the computation that an action $a$ describes may be further refined by use of the refinement construct as in $s_1\langle a \rightsquigarrow s_2 \rangle$. The statement $s_1\langle a \rightsquigarrow s_2 \rangle$ is read as "$s_1$ where

$a$ is refined by $s_2$". In $s_1\langle a \rightsquigarrow s_2\rangle$ the interpretation of any action $a$ in $s_1$ is no longer simply the action $a$ itself but is instead given by the statement $s_2$.

The procedure variables and the constructs sequential composition, nondeterministic choice and parallel composition are as usual. Again a fixed declaration $D$ is assumed and the declaration is suppressed in the notation.

The operational semantics for $\mathcal{L}_{ref}$ is based on the transition system $\mathcal{T}_{\mathrm{ref}}$. As usual resumptions are used in the transition system to describe the part of a program that still remains to be executed. To be able to deal with refinements, the resumptions are more complicated then just statements and the empty resumption. Instead there are three types of resumptions.

- The empty resumption E, denoting a finished computation.
- Resumptions of the format $s : \langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle$ where $n \geq 0$, $s, s_1, s_2, \ldots, s_n$ are statements, and $a_1, a_2, \ldots, a_n$ are, not necessarily distinct, actions. Such a resumption describes a statement $s$ in which first $a_1$ has to be refined by $s_1$, then $a_2$ has to be refined by $s_2$, and so on. We call $\langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle$, also denoted as $\langle a_i \rightsquigarrow s_i\rangle_{i=1}^{n}$, a *refinement sequence.*
- The third type of resumption consists of a combination of resumptions involving the binary operators ;, $\Box$, and $\|$, that are available in $\mathcal{L}_{ref}$. We thus have resumptions of the form $r_1; r_2$, $r_1 \Box r_2$, and $r_1 \| r_2$, respectively.

To describe the refinements that apply for a statement $s$ a refinement sequence $\langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle$ is added to the statement. A single refinement sequence is insufficient because different refinements may apply to different part of the statement as in e.g. $s_1\langle a \rightsquigarrow s\rangle \| s_2$ where the refinement $\langle a \rightsquigarrow s\rangle$ should be used in $s_1$ but not in $s_2$. Instead, a separate stack is needed for each part of a statement. To be able to describe a situation with different refinement stacks for different components, the resumptions, which contain refinement stacks, can be combined with the operators of the languages. In this way one can form e.g. $s_1 : \langle a \rightsquigarrow s\rangle\langle b \rightsquigarrow s'\rangle \| s_2 : \langle b \rightsquigarrow s'\rangle$ where $\langle a \rightsquigarrow s\rangle$ only applies to the first component of the parallel composition.

**Definition 5.2.2**

*(a) The class RefSeq of refinement sequences, ranged over by R, is given by*

$$R \ ::= \ \epsilon \mid \langle a \rightsquigarrow s\rangle \, R$$

*(b) The class Res of resumptions, ranged over by r, is given by*

$$r \ ::= \ \mathrm{E} \mid s : R \mid r \; op \; r$$

*where the operator op is either* ;, $\Box$, *or* $\|$.

A refinement sequence $R$ is either the empty sequence $\epsilon$ or $R = \langle a \rightsquigarrow s\rangle \, R'$ for some $a \in \mathit{Act}$, $s \in \mathit{Stat}$ and $R' \in \mathit{RefSeq}$. Alternatively, an element $R$ of *RefSeq* can be written as

$$R \ = \ \langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle$$

for suitable $n \geq 0$, $a_1, a_2, \ldots, a_n \in \mathit{Act}$, $s_1, s_2, \ldots, s_n \in \mathit{Stat}$. In case $n = 0$ we have $R = \epsilon$.

A configuration in the transition system is a resumption together with a declaration. As a fixed declaration $D$ is assumed, the declaration is suppressed in the notation.

$$\mathit{Conf} \;=\; \mathit{Decl} \times \mathit{Res}$$

The labels used in the transition systems $\mathcal{T}_{\mathrm{ref}}$ are only the atomic actions.

$$\mathit{Lab} \;=\; \mathit{Act}$$

The following form of rule, which describes that the resumption $r_1$ has the same steps as the resumption $r_2$, is used a lot in the transition system.

$$\frac{r_2 \xrightarrow{a} r_3}{r_1 \xrightarrow{a} r_3}$$

The notation $r_1 \rightarrow_0 r_2$, called a zero-step, is therefore used as a shorthand for this rule.

**Definition 5.2.3** *The transition system $\mathcal{T}_{ref}$ is given by $\mathcal{T}_{ref} = (\mathit{Conf}, \mathit{Lab}, \rightarrow, \mathit{Spec})$ where the specification Spec consists of the following axioms and rules*

- $$a : \epsilon \xrightarrow{a} \mathrm{E} \qquad \textit{(Act 1)}$$
  $$a : \langle a' \rightsquigarrow s' \rangle\, R \rightarrow_0 s' : R \qquad \textit{if } a = a' \qquad \textit{(Act 2)}$$
  $$a : \langle a' \rightsquigarrow s' \rangle\, R \rightarrow_0 a : R \qquad \textit{if } a \neq a' \qquad \textit{(Act 3)}$$
- $$x : R \rightarrow_0 D(x) : R \qquad \textit{(Rec)}$$
- $$(s_1\; op\; s_2) : R \rightarrow_0 (s_1 : R)\; op\; (s_2 : R) \qquad \textit{for } op \in \{\, ;\, , \Box, \parallel \,\} \qquad \textit{(Op)}$$
- $$s\langle a' \rightsquigarrow s' \rangle : R \rightarrow_0 s : \langle a' \rightsquigarrow s' \rangle\, R \qquad \textit{(Ref)}$$
- $$\frac{r_1 \xrightarrow{a} r_1'}{r_1 ; r_2 \xrightarrow{a} r_1' ; r_2} \qquad \textit{(Seq)}$$
  *where $r_1' ; r_2$ should be read as $r_2$ is $r_1' = \mathrm{E}$.*
- $$r_1 \Box r_2 \rightarrow_0 r_1 \qquad r_1 \Box r_2 \rightarrow_0 r_2 \qquad \textit{(Choice 1,2)}$$
- $$\frac{r_1 \xrightarrow{a} r_1'}{r_1 \| r_2 \xrightarrow{a} r_1' \| r_2} \qquad \frac{r_2 \xrightarrow{a} r_2'}{r_1 \| r_2 \xrightarrow{a} r_1 \| r_2'} \qquad \textit{(Par 1,2)}$$
  *where $r_1' \| r_2$ in rule (Par 1) should be read as $r_2$ when $r_1' = \mathrm{E}$ and $r_1 \| r_2'$ in rule (Par 2) should be read as $r_1$ when $r_2' = \mathrm{E}$.*

Axiom (Act 1) states that a single action onto which no refinement needs to be applied further, just executes the action and then terminates. The rules (Act 2) and (Act 3) handle the nonempty sequences of refinements from left to right. For all other resumptions of the format $s : R$ exactly one of the zero-step rules (Rec), (Op) or (Ref) applies. (Rec) is the usual rule formalizing body replacement. The (Op)-rule distributes the operator (either $;$, $\Box$, or $\|$) hence reducing the complexity of the statement component. Refinements are dealt with by adding the particular refinement in front of the current refinement sequence. The five remaining rules are the familiar structural rules, in our setting triggered by the earlier rule (Op).

**Example 5.2.4**

(a) *The resumption* $(a \Box (b;c)) : \langle a \leadsto d\rangle\langle b \leadsto e\rangle$ *has two transitions which are derived as follows.*

*Taking the first option for the nondeterministic choice gives*

$$\begin{array}{lll} (a \Box (b;c))\langle a \leadsto d\rangle : \langle b \leadsto e\rangle & & \\ \quad \to_0 & (a \Box (b;c)) : \langle a \leadsto d\rangle\langle b \leadsto e\rangle & \textit{(Ref)} \\ \quad \to_0 & (a : \langle a \leadsto d\rangle\langle b \leadsto e\rangle) \Box ((b;c) : \langle a \leadsto d\rangle\langle b \leadsto e\rangle) & \textit{(Op)} \\ \quad \to_0 & a : \langle a \leadsto d\rangle\langle b \leadsto e\rangle & \textit{(Choice 1)} \\ \quad \to_0 & d : \langle b \leadsto e\rangle & \textit{(Act 2)} \\ \quad \to_0 & d : \epsilon & \textit{(Act 3)} \end{array}$$

*and* $d : \epsilon \xrightarrow{d} \mathrm{E}$ *by (Act 1). Therefore*

$$(a \Box (b;c))\langle a \leadsto d\rangle : \langle b \leadsto e\rangle \xrightarrow{d} \mathrm{E}$$

*Taking the other choice option,*

$$\begin{array}{lll} (a \Box (b;c))\langle a \leadsto d\rangle : \langle b \leadsto e\rangle & & \\ \quad \to_0 & (a \Box (b;c)) : \langle a \leadsto d\rangle\langle b \leadsto e\rangle & \textit{(Ref)} \\ \quad \to_0 & (a : \langle a \leadsto d\rangle\langle b \leadsto e\rangle) \Box ((b;c) : \langle a \leadsto d\rangle\langle b \leadsto e\rangle) & \textit{(Op)} \\ \quad \to_0 & (b;c) : \langle a \leadsto d\rangle\langle b \leadsto e\rangle & \textit{(Choice 2)} \\ \quad \to_0 & (b : \langle a \leadsto d\rangle\langle b \leadsto e\rangle);(c : \langle a \leadsto d\rangle\langle b \leadsto e\rangle) & \textit{(Op)} \end{array}$$

*Using axioms (Act 3) and (Act 2) gives*

$$b : \langle a \leadsto d\rangle\langle b \leadsto e\rangle \to_0 b : \langle b \leadsto e\rangle \to_0 e : \epsilon$$

*and* $e : \epsilon \xrightarrow{e} \mathrm{E}$ *follows by axiom (Act 1). Thus,* $b : \langle a \leadsto d\rangle\langle b \leadsto e\rangle \xrightarrow{e} \mathrm{E}$ *and by application of (Seq)*

$$(b : \langle a \leadsto d\rangle\langle b \leadsto e\rangle);(c : \langle a \leadsto d\rangle\langle b \leadsto e\rangle) \xrightarrow{e} c : \langle a \leadsto d\rangle\langle b \leadsto e\rangle$$

*and therefore*

$$(a \Box (b;c))\langle a \leadsto d\rangle : \langle b \leadsto e\rangle \xrightarrow{e} c : \langle a \leadsto d\rangle\langle b \leadsto e\rangle$$

*(b) As another example consider*

$$\begin{aligned}
&((a;b)\langle a \rightsquigarrow b;c\rangle)\langle b \rightsquigarrow d;e\rangle : \langle b \rightsquigarrow f\rangle \\
&\quad \rightarrow_0 \quad (a;b)\langle a \rightsquigarrow b;c\rangle : \langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle \qquad \textit{(Ref)} \\
&\quad \rightarrow_0 \quad (a;b) : \langle a \rightsquigarrow b;c\rangle\langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle \qquad \textit{(Ref)} \\
&\quad \rightarrow_0 \quad (a : \langle a \rightsquigarrow b;c\rangle\langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle); \\
&\qquad\qquad (b : \langle a \rightsquigarrow b;c\rangle\langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle) \qquad \textit{(Op)}
\end{aligned}$$

*For use with the (Seq)-rule we observe the following*

$$\begin{aligned}
&a : \langle a \rightsquigarrow b;c\rangle\langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle \\
&\quad \rightarrow_0 \quad (b;c) : \langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle \qquad \textit{(Act 2)} \\
&\quad \rightarrow_0 \quad (b : \langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle); (c : \langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle) \qquad \textit{(Op)}
\end{aligned}$$

*Since*

$$\begin{aligned}
&b : \langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle \\
&\quad \rightarrow_0 \quad (d;e) : \langle b \rightsquigarrow f\rangle \qquad \textit{(Act 2)} \\
&\quad \rightarrow_0 \quad (d : \langle b \rightsquigarrow f\rangle); (e : \langle b \rightsquigarrow f\rangle) \qquad \textit{(Act 3)}
\end{aligned}$$

*and* $d : \langle b \rightsquigarrow f\rangle \rightarrow_0 d : \epsilon \xrightarrow{d} \mathrm{E}$ *we obtain, using rule (Seq) gives*

$$b : \langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle \xrightarrow{d} e : \langle b \rightsquigarrow f\rangle$$

*So* $a : \langle a \rightsquigarrow b;c\rangle\langle b \rightsquigarrow d;e\rangle : \langle b \rightsquigarrow f\rangle \xrightarrow{d} (e : \langle b \rightsquigarrow f\rangle); (c : \langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle)$ *and*

$$\begin{aligned}
&((a;b)\langle a \rightsquigarrow b;c\rangle)\langle b \rightsquigarrow d;e\rangle : \langle b \rightsquigarrow f\rangle \\
&\quad \xrightarrow{d} \quad ((e : \langle b \rightsquigarrow f\rangle); (c : \langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle)); \\
&\qquad\qquad (b : \langle a \rightsquigarrow b;c\rangle\langle b \rightsquigarrow d;e\rangle\langle b \rightsquigarrow f\rangle)
\end{aligned}$$

The example above illustrates, among others, how the rules (Ref), (Act 2) and (Act 3) deal with the addition and removal of refinements from the refinement sequence within a configuration. To show several properties of the transition system, such as the fact that the addition and removal of refinements to the refinement sequence does not create a loop in the transition system, weight induction is used. To this end the complexity function *wgt* is defined, first for programs and then for configurations.

**Definition 5.2.5**

*(a) The function wgt*: $\mathcal{L}_{ref} \to \mathbb{N}$ *is given by*

$$\begin{array}{rcl} wgt(a) & = & 1 \\ wgt(x) & = & wgt(D(x)) + 1 \\ wgt(s_1; s_2) & = & wgt(s_1) + 1 \\ wgt(s_1 \, op \, s_2) & = & wgt(s_1) + wgt(s_2) + 1 \;\; \textit{for } op \in \{ \,\square, \| \,\} \\ wgt(s_1 \langle a \rightsquigarrow s_2 \rangle) & = & wgt(s_1) + wgt(s_2) + 1. \end{array}$$

*(b) The function wgt*: *Conf* $\to \mathbb{N}$ *is given by*

$$\begin{array}{rcl} wgt(\mathrm{E}) & = & 0 \\ \multicolumn{3}{l}{wgt(s : \langle a_1 \rightsquigarrow s_1 \rangle \langle a_2 \rightsquigarrow s_2 \rangle \cdots \langle a_n \rightsquigarrow s_n \rangle)} \\ & = & wgt(s) + wgt(s_1) + wgt(s_2) + \cdots + wgt(s_n) \\ wgt(r_1; r_2) & = & wgt(r_1) + 1 \\ wgt(r_1 \, op \, r_2) & = & wgt(r_1) + wgt(r_2) + 1 \;\; \textit{for } op \in \{ \,\square, \| \,\}. \end{array}$$

Well-definedness of these weight functions can be shown by structural induction, first on guarded statements, then on general statements and finally on resumptions. The following properties are easy to check by using weight induction.

**Lemma 5.2.6**

*(a) A zero step decreases the weight of the configuration, i.e. if* $r \to_0 r'$ *then* $wgt(r) > wgt(r')$.

*(b) The transition system* $\mathcal{T}_{ref}$ *is finitely branching, i.e. the set* $\{ \langle a, r' \rangle \mid r \xrightarrow{a} r' \}$ *is finite for each configuration* $r$.

The operational meaning of a program consists of all possible sequences of actions that the program can produce. Because the transition system is finitely branching, these sequences form a compact set. The domain $\mathbb{P}_o$ of operational processes contains all possible operational meanings.

**Definition 5.2.7** *The domain* $\mathbb{P}_o$ *of operational processes is given by*

$$\mathbb{P}_o \;=\; \mathcal{P}_{nco}(Act^{\infty})$$

The operational model $\mathcal{O}$ yields the operational meaning of a given configuration.

**Definition 5.2.8** *The operational model* $\mathcal{O}$: *Conf* $\to \mathbb{P}_o$ *is given by*

$$\begin{array}{rcl} \mathcal{O}(\mathrm{E}) & = & \{ \epsilon \} \\ \mathcal{O}(r) & = & \{ a\, w \mid r \xrightarrow{a} r', w \in \mathcal{O}(r') \} \end{array}$$

*where* $r$ *is any resumption not equal to* E.

The empty resumption E yields the empty sequence $\epsilon$. For other resumptions $r$, the possible sequences are found by looking at the transitions of $r$. If $r$ takes an $a$ step to $r'$, $r \xrightarrow{a} r'$ then any possible sequence for $r'$ prefixed with $a$ is a possible sequence for $r$.

The well-definedness of $\mathcal{O}$ needs further comment as the definition is recursive. Justification of the definition of $\mathcal{O}$ is given after some examples.

**Examples 5.2.9**

*(a) Since* $(a \,\Box\, (b;c))\langle a \rightsquigarrow d\rangle\langle b \rightsquigarrow e\rangle \xrightarrow{d} \mathrm{E}$ *and* $(a \,\Box\, (b;c))\langle a \rightsquigarrow d\rangle\langle b \rightsquigarrow e\rangle \xrightarrow{e} c : \langle a \rightsquigarrow d\rangle\langle b \rightsquigarrow e\rangle \xrightarrow{c} \mathrm{E}$ *are the computations for* $(a \,\Box\, (b;c))\langle a \rightsquigarrow d\rangle\langle b \rightsquigarrow e\rangle$ *we get*

$$\begin{aligned}
&\mathcal{O}((a \,\Box\, (b;c))\langle a \rightsquigarrow d\rangle\langle b \rightsquigarrow e\rangle)\\
&= d\,\mathcal{O}(\mathrm{E}) \cup e\,\mathcal{O}(c : \langle a \rightsquigarrow d\rangle\langle b \rightsquigarrow e\rangle)\\
&= d\{\,\epsilon\,\} \cup e\,c\,\mathcal{O}(\mathrm{E})\\
&= \{\,d\,\} \cup e\,c\{\,\epsilon\,\}\\
&= \{\,d\,\} \cup \{\,ec\,\}\\
&= \{\,d, ec\,\}
\end{aligned}$$

*(b) Suppose* $D(x) = a\langle b \rightsquigarrow c\rangle$. *Then* $\mathcal{O}((x \,\Box\, b) : \epsilon) = \mathcal{O}(x : \epsilon) \cup \mathcal{O}(b : \epsilon) = \mathcal{O}(a\langle b \rightsquigarrow c\rangle : \epsilon) \cup b\,\mathcal{O}(\mathrm{E}) = \ldots = \{\,a\,\} \cup \{\,b\,\} = \{\,a, b\,\}$.

*(c) Suppose* $D(x) = (a;x) \,\Box\, b$. *We show that the process* $\mathcal{O}(x\langle a \rightsquigarrow c;d\rangle : \epsilon)$ *is equal to the process* $\{\,(cd)^n b \mid n \geq 0\,\} \cup \{\,(cd)^\omega\,\}$ *by showing that their distance equals* 0.

$$\begin{aligned}
&\mathcal{O}(x\langle a \rightsquigarrow c;d\rangle : \epsilon)\\
&= \mathcal{O}(((a;x) \,\Box\, b) : \langle a \rightsquigarrow c;d\rangle)\\
&= \mathcal{O}((a;x) : \langle a \rightsquigarrow c;d\rangle) \,\Box\, b : \langle a \rightsquigarrow c;d\rangle)\\
&= \mathcal{O}((a : \langle a \rightsquigarrow c;d\rangle); (x : \langle a \rightsquigarrow c;d\rangle)) \cup \mathcal{O}(b : \langle a \rightsquigarrow c;d\rangle)\\
&= \mathcal{O}(((c;d) : \epsilon); (x : \langle a \rightsquigarrow c;d\rangle)) \cup b\,\mathcal{O}(\mathrm{E})\\
&= c\,\mathcal{O}((d : \epsilon); (x : \langle a \rightsquigarrow c;d\rangle)) \cup b\{\,\epsilon\,\}\\
&= c\,d\,\mathcal{O}(x : \langle a \rightsquigarrow c;d\rangle) \cup \{\,b\,\}
\end{aligned}$$

*Since* $x\langle a \rightsquigarrow c;d\rangle : \epsilon \rightarrow_0 x : \langle a \rightsquigarrow c;d\rangle$ *we have that* $\mathcal{O}(x\langle a \rightsquigarrow c;d\rangle : \epsilon) = \mathcal{O}(x : \langle a \rightsquigarrow c;d\rangle)$. *Therefore we obtain* $\mathcal{O}(x\langle a \rightsquigarrow c;d\rangle : \epsilon) = c\,d\,\mathcal{O}(x\langle a \rightsquigarrow c;d\rangle : \epsilon) \cup \{\,b\,\}$. *On the other hand, for the set* $X = \{\,(cd)^n b \mid n \geq 0\,\} \cup \{\,(cd)^\omega\,\}$ *we have that*

$$X \;=\; c\,d\,X \cup \{\,b\,\}.$$

*From these equations we derive that the distance of* $\mathcal{O}(x\langle a \rightsquigarrow c;d\rangle : \epsilon)$ *to the set* $X$ *is zero.*

$$\begin{aligned}
&d(\mathcal{O}(x\langle a \rightsquigarrow c;d\rangle : \epsilon), X)\\
&= d(c\,d\,\mathcal{O}(x : \langle a \rightsquigarrow c;d\rangle) \cup \{\,b\,\}, c\,d\,X \cup \{\,b\,\})\\
&\leq \max\{\,d(c\,d\,\mathcal{O}(x : \langle a \rightsquigarrow c;d\rangle), c\,d\,X), d(\{\,b\,\}, \{\,b\,\})\,\}
\end{aligned}$$

$$
\begin{aligned}
&= \max\{ \tfrac{1}{4} d(\mathcal{O}(x : \langle a \rightsquigarrow c; d\rangle), X), 0 \} \\
&= \tfrac{1}{4} d(\mathcal{O}(x : \langle a \rightsquigarrow c; d\rangle, X) \\
&= \tfrac{1}{4} d(\mathcal{O}(x\langle a \rightsquigarrow c; d\rangle : \epsilon, X)
\end{aligned}
$$

*Note that* $0$ *is the only non-negative real number* $\alpha$ *such that* $\alpha \leq \frac{1}{4}\alpha$. *We conclude* $\mathcal{O}(x\langle a \rightsquigarrow c; d\rangle : \epsilon) = \{ (cd)^n b \mid n \geq 0 \} \cup \{ (cd)^\omega \}$.

The recursive definition of the operational model $\mathcal{O}$ can be justified by showing that it is the unique fixed point of a higher-order transformation $\Phi$. The proof is omitted as it is similar to the proof of lemma 4.2.21. (Or see [38], definition 2.32 and lemma 2.33.)

**Lemma 5.2.10** *Put Sem* $=$ *Conf* $\to \mathcal{P}_{nc}(Act^\infty)$. *Use* $S$ *for a typical element of Sem. Let the higher-order transformation* $\Phi$: *Sem* $\to$ *Sem be given by*

$$
\begin{aligned}
\Phi(S)(\mathrm{E}) &= \{ \epsilon \} \\
\Phi(S)(r) &= \bigcup \{ a\, S(r') \mid r \xrightarrow{a} r' \}
\end{aligned}
$$

*Then* $\Phi$ *has a unique fixed point, and therefore there is exactly one function* $\mathcal{O}$ *in Sem which satisfies the equations in definition 5.2.8.*

The operational model $\mathcal{O}$ gives the meaning of resumptions instead of programs in $\mathcal{L}_{ref}$. In a resumptions a refinement sequence is always included. For a closed system, there are no more refinements to execute, so one can start with an empty sequence of refinements. The operational semantics $\mathcal{O}[\![\bullet]\!]$ is obtained by using the empty refinement sequence.

**Definition 5.2.11** *The operational semantics* $\mathcal{O}[\![\bullet]\!]: \mathcal{L}_{ref} \to \mathbb{P}_o$ *is given by*

$$\mathcal{O}[\![s]\!] = \mathcal{O}(s : \epsilon)$$

Note that the operational semantics satisfies the following two properties

$$
\begin{aligned}
a_1 a_2 \cdots a_n \in \mathcal{O}[\![s]\!] \quad &\Longleftrightarrow \quad s : \epsilon = r_0 \xrightarrow{a_1} r_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} r_n = \mathrm{E} \\
&\qquad\quad \text{for some } r_0, r_1, \ldots, r_n \in \mathit{Res} \\
a_1 a_2 \cdots \in \mathcal{O}[\![s]\!] \quad &\Longleftrightarrow \quad s : \epsilon = r_0 \xrightarrow{a_1} r_1 \xrightarrow{a_2} \cdots \\
&\qquad\quad \text{for some } r_0, r_1, r_2, \ldots \in \mathit{Res}
\end{aligned}
$$

The operational semantics $\mathcal{O}[\![\bullet]\!]$ collects the sequences of actions of all the computations that are possible for a given statement. For this the resumption $s : \epsilon$ is used as starting configuration for a computation for $s$.

## 5.2.2 Semantical refinements and denotational semantics

In this subsection a denotational model for $\mathcal{L}_{ref}$ is given. The notion of a *semantical refinements* $\eta$ is introduced that correspond to the refinement sequences $R$ discussed in the previous section. The semantical refinements mimic the role of environments for procedure variables as usually employed in the traditional approach to programming language semantics. In this setting one commonly encounters in the definition for a denotational semantics $\mathcal{D}$ a clause such as

$$\mathcal{D}(x)(\mu) = \mu(x) \tag{2.1}$$

where the so-called environment $\mu$ is a function $\mu\colon PVar \to \mathbb{P}_d$ used to store and retrieve the meaning of a procedure variable (see e.g. [187]). To determine which mapping $\mu_0\colon PVar \to \mathbb{P}_d$ is the proper one an additional fixed point argument is necessary. In the metric methodology the application of environments for the modeling of recursion is generally avoided. As originally proposed by Kok and Rutten [139], one can simply state

$$\mathcal{D}(x) \quad = \quad \mathcal{D}(D(x))$$

provided sufficient contractiveness for the underlying fixed point characterization can be guaranteed. This is achieved here by restricting to guarded recursion.

The use of environments for procedure variables is, in our approach, replaced by the use of environments for actions. That is, the environment now holds the meanings of all actions, taking into account the refinements which they are subject to. Accordingly, for the semantic environment $\eta\colon Act \to \mathbb{P}_d$ we have

$$\mathcal{D}(a)(\eta) \quad = \quad \eta(a)$$

as a clause of our denotational semantics. Note the correspondence with equation (2.1) above. Additionally we will put

$$\mathcal{D}(s_1\langle a \rightsquigarrow s_2\rangle)(\eta) \quad = \quad \mathcal{D}(s_1)(\eta[\mathcal{D}(s_2)(\eta)/a])$$

where, as it were, the binding of $a$ in $s_1$ is dynamically set to the meaning of $s_2$ with respect to the current refinement $\eta$. The construct $\eta[p/a]$ is called a *variant* of $\eta$, which only differs from $\eta$ in the action $a$ for which the element $p$ is delivered. The refinement statements thus effect the interpretation of actions. Clearly, as initial semantical refinement the '*identity refinement*' $\eta_{id}$, i.e., the mapping $\lambda a.\{\, a \,\}$, is the only option, so no fixed point argument is required for this.

Elaboration of the plan as sketched above starts with the definition of the semantical domain $\mathbb{P}_d$ and the semantical operations. For technical reasons explained below we will exclude the empty string of actions and only consider nonempty sets of nonempty sequences.

**Definition 5.2.12** *Let $\mathbb{P}_d = \mathcal{P}_{nco}(\mathbb{Q}_d)$ where $\mathbb{Q}_d = Act^\infty \setminus \{\, \epsilon \,\}$ be the domain of denotational processes. The semantical operations* $;\,, \Box, \|\colon \mathbb{P}^2 \to \mathbb{P}$ *are given as follows, employing the auxiliary operations* $;'\colon \mathbb{Q}^2 \to \mathbb{Q}$ *and* $\|', \mathbin{\lfloor\!\lfloor}'\colon \mathbb{Q}^2 \to \mathbb{P}$

*(a)*
$$\begin{aligned} p_1; p_2 &= \{\, q_1 ;' q_2 \mid q_1 \in p_1, q_2 \in p_2 \,\} \\ a ;' q &= a\, q \\ (a\, q') ;' q &= a\, (q' ;' q) \end{aligned}$$

*(b)*
$$p_1 \Box p_2 \quad = \quad p_1 \cup p_2$$

*(c)*
$$\begin{array}{rcl} p_1 \| p_2 & = & \bigcup \{ q_1 \parallel' q_2 \mid q_1 \in p_1, q_2 \in p_2 \} \\ q_1 \parallel' q_2 & = & q_1 \mathbin{\|\!\lfloor}' q_2 \,\square\, q_2 \mathbin{\|\!\lfloor}' q_1 \\ a \mathbin{\|\!\lfloor}' q & = & \{ a\, q \} \\ (a\, q') \mathbin{\|\!\lfloor}' q & = & a\, (q' \parallel' q) \end{array}$$

Proof of the well-definedness and nonexpansiveness of these operations can be found in [38]. The proof proceeds along the usual way: Each operation is shown to be the unique fixed point of a contractive higher order operation (cf. definition 3.4.9).

**Lemma 5.2.13**

*(a) The semantical operations* $;$, $\square$, *and* $\|$ *are are well-defined and nonexpansive.*

*(b) The mapping* $;$ *is contractive in its second argument, i.e.* $d(p; p_1, p; p_2) \leq \frac{1}{2} d(p_1, p_2)$.

The second property relies on the fact that the empty sequence is excluded from denotational processes. If the empty sequence had not been excluded one can, for example, take $p = \{ \epsilon \}$, $p_1 = \{ a \}$ and $p_2 = \{ b \}$ which gives $d(p; p_1, p; p_2) = d(p_1, p_2) \not\leq \frac{1}{2} d(p_1, p_2)$.

Using the semantical operations the definition of the denotational model $\mathcal{D}$ is given. As mentioned at the start of this subsection the model $\mathcal{D}$ uses an intermediate layer of semantical refinements *SemRef*.

**Definition 5.2.14**

*(a) Put* $\mathit{SemRef} = \mathit{Act} \to \mathbb{P}_d$. *The set SemRef has typical element* $\eta$. *The denotational model* $\mathcal{D}\colon \mathcal{L}_{ref} \to \mathit{SemRef} \to \mathbb{P}_d$ *is given by*

$$\begin{array}{rcl} \mathcal{D}(a)(\eta) & = & \eta(a) \\ \mathcal{D}(x)(\eta) & = & \mathcal{D}(D(x))(\eta) \\ \mathcal{D}(s_1 \mathbin{op} s_2)(\eta) & = & \mathcal{D}(s_1)(\eta) \mathbin{op} \mathcal{D}(s_2)(\eta) \quad \textit{for } op \in \{ ;, \square, \| \} \\ \mathcal{D}(s_1 \langle a \rightsquigarrow s_2 \rangle)(\eta) & = & \mathcal{D}(s_1)(\eta[\mathcal{D}(s_2)(\eta)/a]) \end{array}$$

*(b) The denotational semantics* $\mathcal{D}[\![\bullet]\!]\colon \mathcal{L}_{ref} \to \mathbb{P}_d$ *is given by* $\mathcal{D}[\![s]\!] = \mathcal{D}(s)(\eta_{id})$.

The denotational model $\mathcal{D}$ passes actions as argument to the given semantical refinement. Recursion is handled via body replacement. For the various syntactical constructions the usual compositionality principle applies. As discussed above, a refinement statement $s_1 \langle a \rightsquigarrow s_2 \rangle$ amounts to an update of the current semantical refinement for the action $a$. It is in the resulting semantical refinement that the statement $s_1$ should be evaluated.

The denotational semantics $\mathcal{D}[\![\bullet]\!]$ gives the meaning of a program be evaluating the program starting with the identity refinement $\eta_{id}$. The identity refinement, which assigns $\{ a \}$ to an action $a$, corresponds to the situation where no refinements need to be done.

**Example 5.2.15**

*(a) Note that* $\mathcal{D}(d)(\eta_{id}) = \eta_{id}(d) = \{ d \}$ *and similar for the action* $e$. *Also, by definition of variants,* $(\eta_{id}[\{ e \}/b])(d) = \eta_{id}(d) = d$, *thus we have*

$$\begin{aligned}
&\mathcal{D}((a \,\Box\, (b;c))\langle a \rightsquigarrow d\rangle\langle b \rightsquigarrow e\rangle)(\eta_{id}) \\
&\quad = \mathcal{D}((a \,\Box\, (b;c))\langle a \rightsquigarrow d\rangle)(\eta_{id}[\{\, e \,\}/b]) \\
&\quad = \mathcal{D}(a \,\Box\, (b;c))(\eta_{id}[\{\, e \,\}/b][\{\, d \,\}/a]) \\
&\quad = \mathcal{D}(a)(\eta_{id}[\{\, e \,\}/b][\{\, d \,\}/a]) \,\Box\, \mathcal{D}(b;c)(\eta_{id}[\{\, e \,\}/b][\{\, d \,\}/a]) \\
&\quad = \eta_{id}[\{\, e \,\}/b][\{\, d \,\}/a](a) \cup \\
&\qquad (\,\mathcal{D}(b)(\eta_{id}[\{\, e \,\}/b][\{\, d \,\}/a]); \mathcal{D}(c)(\eta_{id}[\{\, e \,\}/b][\{\, d \,\}/a])\,) \\
&\quad = \{\, d \,\} \cup (\eta_{id}[\{\, e \,\}/b][\{\, d \,\}/a](b); \eta_{id}[\{\, e \,\}/b][\{\, d \,\}/a](c)) \\
&\quad = \{\, d \,\} \cup (\{\, e \,\}; \{\, c \,\}) \\
&\quad = \{\, d \,\} \cup \{\, ec \,\} \\
&\quad = \{\, d, ec \,\}
\end{aligned}$$

*(b) Suppose $D(x) = a\langle b \rightsquigarrow c\rangle$. We then have*

$$\begin{aligned}
&\mathcal{D}(x \,\Box\, b)(\eta_{id}) \\
&\quad = \mathcal{D}(x)(\eta_{id}) \,\Box\, \mathcal{D}(b)(\eta_{id}) \\
&\quad = \mathcal{D}(a\langle b \rightsquigarrow c\rangle)(\eta_{id}) \cup \eta_{id}(b) \\
&\quad = \mathcal{D}(a)(\eta_{id}[\{\, c \,\}/b]) \cup \{\, b \,\} \\
&\quad = \ldots \\
&\quad = \{\, a, b \,\}
\end{aligned}$$

*(c) Suppose $D(x) = (a;x)\Box b$. We argue that $\mathcal{D}[\![x\langle a \rightsquigarrow c;d\rangle]\!] = \{\,(cd)^n b \mid n \geq 0\,\} \cup \{\,(cd)^\omega\,\}$ following the same example for the operational semantics. Note $\mathcal{D}(c;d)(\eta_{id}) = \{\, cd \,\}$. We have*

$$\begin{aligned}
&\mathcal{D}(x\langle a \rightsquigarrow c;d\rangle)(\eta_{id}) \\
&\quad = \mathcal{D}(x)(\eta_{id}[\{\, cd \,\}/a]) \\
&\quad = \mathcal{D}((a;x) \,\Box\, b)(\eta_{id}[\{\, cd \,\}/a]) \\
&\quad = \mathcal{D}(a;x)(\eta_{id}[\{\, cd \,\}/a]) \,\Box\, \mathcal{D}(b)(\eta_{id}[\{\, cd \,\}/a]) \\
&\quad = (\mathcal{D}(a)(\eta_{id}[\{\, cd \,\}/a]); \mathcal{D}(x)(\eta_{id}[\{\, cd \,\}/a])) \cup \{\, b \,\} \\
&\quad = (\{\, cd \,\}; \mathcal{D}(x)(\eta_{id}[\{\, cd \,\}/a])) \cup \{\, b \,\} \\
&\quad = c\,d\,\mathcal{D}(x)(\eta_{id}[\{\, cd \,\}/a]) \cup \{\, b \,\}
\end{aligned}$$

*Thus $\mathcal{D}(x\langle a \rightsquigarrow c;d\rangle)(\eta_{id}) = \mathcal{D}(x)(\eta_{id}[\{\, cd \,\}/a])$ and*

$$\mathcal{D}(x)(\eta_{id}[\{\, cd \,\}/a]) \;=\; c\,d\,\mathcal{D}(x)(\eta_{id}[\{\, cd \,\}/a]) \cup \{\, b \,\}$$

*Therefore we obtain*

$$\mathcal{D}[\![x\langle a \rightsquigarrow c;d\rangle]\!] \;=\; \{\,(cd)^n b \mid n \geq 0\,\} \cup \{\,(cd)^\omega\,\}$$

*by a similar argument as in the previous subsection (see example 5.2.9).*

As usual the well-definedness of the denotational model $\mathcal{D}$ is obtained by showing that $\mathcal{D}$ is the unique fixed point of a contractive higher-order transformation. The higher-order transformation $\Psi$ is a function on *Sem*, $\Psi\colon \mathit{Sem} \to \mathit{Sem}$, where *Sem* is the function space $\mathcal{L}_{\mathit{ref}} \to \mathit{SemRef} \xrightarrow{1} \mathbb{P}_d$. Recall that $\xrightarrow{1}$ indicates a restriction to nonexpansive mappings. This restriction is needed to obtain contractivity of $\Psi$.

The next lemma gives the transformation $\Psi$ and states that this function is contractive and thus has a unique fixed point. Note that the restriction to the function space $\mathcal{L}_{\mathit{ref}} \to \mathit{SemRef} \xrightarrow{1} \mathbb{P}_d$ is essential for the proof of part (b), in particular the induction step for statements $s_1\langle a \rightsquigarrow s_2\rangle$.

**Lemma 5.2.16** *Put Sem* $= \mathcal{L}_{\mathit{ref}} \to \mathit{SemRef} \xrightarrow{1} \mathbb{P}_d$ *and let $S$ be a typical element. The transformation* $\Psi\colon \mathit{Sem} \to \mathit{Sem}$ *is given by*

$$\begin{array}{rcl} \Psi(S)(a)(\eta) & = & \eta(a) \\ \Psi(S)(x)(\eta) & = & \Psi(S)(D(x))(\eta) \\ \Psi(S)(s_1;s_2)(\eta) & = & \Psi(S)(s_1)(\eta); S(s_2)(\eta) \\ \Psi(S)(s_1 \mathbin{op} s_2)(\eta) & = & \Psi(S)(s_1)(\eta) \mathbin{op} \Psi(S)(s_2)(\eta) \quad \textit{for } op \in \{\, \Box, \| \,\} \\ \Psi(S)(s_1\langle a \rightsquigarrow s_2\rangle)(\eta) & = & \Psi(S)(s_1)(\eta[p/a]) \quad \textit{where } p = \Psi(S)(s_2)(\eta) \end{array}$$

*(a) The mapping $\Psi$ is well-defined, i.e., for all $S \in \mathit{Sem}$ and $s \in \mathcal{L}_{\mathit{ref}}$ the mapping $\Psi(S)(s)\colon \mathit{SemRef} \to \mathbb{P}_d$ is nonexpansive.*

*(b) The mapping $\Psi$ is a $\frac{1}{2}$-contraction.*

**Proof** The definition of $\Psi(S)(s)(\eta)$ is by the induction on $\mathit{wgt}(s)$. Note that $\Psi$ is not applied to the $s_2$-component in the right-hand side of the clause for the sequential composition. That $\Psi(S)(s)(\eta)$ is in $\mathbb{P}_d$ for all arguments, can be straightforwardly established by weight-induction using the well-definedness of the semantical operations as given in lemma 5.2.13.

(a) Let $S \in \mathit{Sem}$, $s \in \mathcal{L}_{\mathit{ref}}$. We check

$$d(\Psi(S)(s)(\eta_1), \Psi(S)(s)(\eta_2)) \leq d_F(\eta_1, \eta_2)$$

for all $\eta_1, \eta_2 \in \mathit{SemRef}$, by weight-induction. We only present the more interesting cases of sequential composition and action refinement. Note that, by definition, $\mathit{Sem} = \mathcal{L}_{\mathit{ref}} \to \mathit{SemRef} \xrightarrow{1} \mathbb{P}_d$. Thus we have for each $S \in \mathit{Sem}$, $s \in \mathcal{L}_{\mathit{ref}}$, and $\eta_1, \eta_2 \in \mathit{SemRef}$, that $d(S(s)(\eta_1), S(s)(\eta_2)) \leq d_F(\eta_1, \eta_2)$.

$[s_1; s_2]$

$$\begin{array}{rl} & d(\Psi(S)(s_1;s_2)(\eta_1), \Psi(S)(s_1;s_2)(\eta_2)) \\ = & d(\Psi(S)(s_1)(\eta_1); S(s_2)(\eta_1), \Psi(S)(s_1)(\eta_2); S(s_2)(\eta_2)) \\ \leq & [\,;\ \text{nonexpansive}] \\ & \max\{\ d(\Psi(S)(s_1)(\eta_1), \Psi(S)(s_1)(\eta_2)), \\ & \qquad d(S(s_2)(\eta_1), S(s_2)(\eta_2))\,\} \\ \leq & [\text{induction hypothesis for } s_1, \text{ restriction on } \mathit{Sem}] \\ & \max\{\, d_F(\eta_1,\eta_2), d_F(\eta_1,\eta_2)\,\} \end{array}$$

$= \quad d_F(\eta_1, \eta_2)$

$[s_1\langle a \rightsquigarrow s_2\rangle] \quad d(\Psi(S)(s_1\langle a \rightsquigarrow s_2\rangle)(\eta_1), \Psi(S)(s_1\langle a \rightsquigarrow s_2\rangle)(\eta_2))$

$= \quad d(\Psi(S)(s_1)(\eta_1[p_1/a]), \Psi(S)(s_1)(\eta_2[p_2/a]))$

where $p_i = \Psi(S)(s_2)(\eta_i)$, $i = 1, 2$

$\leq$ [induction hypothesis] $d_F(\eta_1[p_1/a], \eta_2[p_2/a])$

By definition of $d_F$ as supremum (see lemma 2.1.7) it is clear that

$$d_F(\eta_1[p_1/a], \eta_2[p_2/a]) \leq \max\{\, d_F(\eta_1, \eta_2), d(p_1, p_2)\,\}$$

Now, since

$$d(p_1, p_2) = d(\Psi(S)(s_2)(\eta_1), \Psi(S)(s_2)(\eta_2)) \leq d_F(\eta_1, \eta_2)$$

by the induction hypothesis, it follows that $d_F(\eta_1[p_1/a], \eta_2[p_2/a]) \leq d_F(\eta_1, \eta_2)$ and, combining this with the inequation derived above

$$d(\Psi(S)(s_1\langle a \rightsquigarrow s_2\rangle)(\eta_1), \Psi(S)(s_1\langle a \rightsquigarrow s_2\rangle)(\eta_2)) \leq d_F(\eta_1, \eta_2)$$

(b) Let $S_1, S_2 \in \mathit{Sem}$. Using the result from part (a) we prove by weight-induction that

$$d(\Psi(S_1)(s)(\eta), \Psi(S_2)(s)(\eta)) \leq \tfrac{1}{2} d_F(S_1, S_2)$$

for all $s \in \mathcal{L}_{ref}$, $\eta \in \mathit{SemRef}$. From this we get $d_F(\Psi(S_1), \Psi(S_2)) \leq \frac{1}{2} d_F(S_1, S_2)$ by definition of $d_F$. Again we only discuss the cases of sequential composition and action refinement.

$[s_1; s_2] \quad d(\Psi(S_1)(s_1; s_2)(\eta), \Psi(S_2)(s_1; s_2)(\eta))$

$= \quad d(\Psi(S_1)(s_1)(\eta); S_1(s_2)(\eta), \Psi(S_2)(s_1)(\eta); S_2(s_2)(\eta))$

$\leq$ [; nonexpansive in 1st, contractive in 2nd argument]
$\max\{\quad d(\Psi(S_1)(s_1)(\eta), \Psi(S_2)(s_1)(\eta)),$
$\quad \tfrac{1}{2} d(S_1(s_2)(\eta), S_2(s_2)(\eta))\,\}$

$\leq$ [induction hypothesis, definition $d_F$] $\tfrac{1}{2} d_F(S_1, S_2)$.

$[s_1\langle a \rightsquigarrow s_2\rangle] \quad d(\Psi(S_1)(s_1\langle a \rightsquigarrow s_2\rangle)(\eta), \Psi(S_2)(s_1\langle a \rightsquigarrow s_2\rangle)(\eta))$

$= \quad d(\Psi(S_1)(s_1)(\eta[p_1/a]), \Psi(S_2)(s_1)(\eta[p_2/a]))$

where $p_i = \Psi(S_i)(s_2)(\eta)$, i=1,2

$\leq$ [ultrametricity]
$\max\{\ d(\Psi(S_1)(s_1)(\eta[p_1/a]), \Psi(S_2)(s_1)(\eta[p_1/a])),$
$\quad d(\Psi(S_2)(s_1)(\eta[p_1/a]), \Psi(S_2)(s_1)(\eta[p_2/a]))\,\}$

$\leq$ [induction hypothesis, part (a) for $\Psi(S_2)(s_1)$]
$\max\{\, \tfrac{1}{2} d_F(S_1, S_2), d_F(\eta[p_1/a], \eta[p_2/a])\,\}$

By definition of $d_F$ it follows that

$$d_F(\eta[p_1/a], \eta[p_2/a]) = d(\Psi(S_1)(s_2)(\eta), \Psi(S_2)(s_2)(\eta)) \leq \tfrac{1}{2} d_F(S_1, S_2)$$

by the induction hypothesis for $s_2$. Therefore

$$d(\Psi(S_1)(s_1\langle a \rightsquigarrow s_2\rangle)(\eta), \Psi(S_2)(s_1\langle a \rightsquigarrow s_2\rangle)(\eta)) \leq \tfrac{1}{2} d_F(S_1, S_2)$$

Conclusion: $\Psi$ is a $\frac{1}{2}$-contraction. $\square$

It is clear that the fixed point of $\Psi$ satisfies the equations given for $\mathcal{D}$ in definition 5.2.14. This means that these equations have a solution. In previous applications of the higher-order transformation $\Psi$ (cf. lemma 3.4.13) it was also obvious that any model satisfying the equations for $\mathcal{D}$ must be a fixed point of $\Psi$. This gives uniqueness of the solution as $\Psi$ has only one fixed point. Here we have only shown that the equations for $\mathcal{D}$ have a unique solution within the space $\mathcal{L}_{ref} \to \mathit{SemRef} \xrightarrow{1} \mathbb{P}_d$. It remains to be shown that any solution $\mathcal{D}$ for the equations in definition 5.2.14 must be within this space, i.e. that the model $\mathcal{D}$ is nonexpansive in its second argument.

**Lemma 5.2.17** *For $s \in \mathcal{L}_{ref}$, the mapping $\mathcal{D}(s)\colon \mathit{SemRef} \to \mathbb{P}_d$ is nonexpansive.*

**Proof** Let us write $\tilde{\mathcal{D}} = \mathit{fix}(\Psi)$ for short. Note that, by definition of $\Psi$, $\tilde{\mathcal{D}}$ is nonexpansive in its second argument. Put $\varepsilon = d_F(\mathcal{D}, \tilde{\mathcal{D}})$ where $d_F$ is the distance on the function space $\mathcal{L}_{ref} \to \mathit{SemRef} \to \mathbb{P}_d$. We first check

$$\forall \eta \in \mathit{SemRef}\colon d(\mathcal{D}(s)(\eta), \tilde{\mathcal{D}}(s)(\eta)) \leq \tfrac{1}{2}\varepsilon \tag{2.2}$$

by weight-induction on $s$. We only exhibit the two nontrivial cases.

$[s_1; s_2]$ $\quad d(\mathcal{D}(s_1; s_2)(\eta), \tilde{\mathcal{D}}(s_1; s_2)(\eta))$

$$\begin{aligned}
&= \text{[def. 5.2.14(a); } \tilde{\mathcal{D}} = \mathit{fix}(\Psi)] \;\; d(\mathcal{D}(s_1)(\eta); \mathcal{D}(s_2)(\eta), \tilde{\mathcal{D}}(s_1)(\eta); \tilde{\mathcal{D}}(s_2)(\eta)) \\
&\leq \text{[lem. 5.2.13(c)]} \;\; \max\{\, d(\mathcal{D}(s_1)(\eta), \tilde{\mathcal{D}}(s_1)(\eta)), \tfrac{1}{2} d(\mathcal{D}(s_2)(\eta), \tilde{\mathcal{D}}(s_2)(\eta)) \,\} \\
&\leq \text{[ind. hyp. for } s_1\text{; def. } \varepsilon] \;\; \tfrac{1}{2}\varepsilon
\end{aligned}$$

$[s_1\langle a \rightsquigarrow s_2\rangle]$ $\quad d(\mathcal{D}(s_1\langle a \rightsquigarrow s_2\rangle)(\eta), \tilde{\mathcal{D}}(s_1\langle a \rightsquigarrow s_2\rangle)(\eta))$

$$\begin{aligned}
&= \text{[def. 5.2.14(a); } \tilde{\mathcal{D}} = \mathit{fix}(\Psi)] \\
&\quad\; d(\mathcal{D}(s_1)(\eta[\mathcal{D}(s_2)(\eta)/a]), \tilde{\mathcal{D}}(s_1)(\eta[\tilde{\mathcal{D}}(s_2)(\eta)/a])) \\
&\leq \text{[ultrametricity]} \\
&\quad\; \max\{\, d(\mathcal{D}(s_1)(\eta[\mathcal{D}(s_2)(\eta)/a]), \tilde{\mathcal{D}}(s_1)(\eta[\mathcal{D}(s_2)(\eta)/a])), \\
&\quad\qquad d(\tilde{\mathcal{D}}(s_1)(\eta[\mathcal{D}(s_2)(\eta)/a]), \tilde{\mathcal{D}}(s_1)(\eta[\tilde{\mathcal{D}}(s_2)(\eta)/a])) \,\} \\
&\leq \text{[ind. hyp. for } s_1\text{; } \tilde{\mathcal{D}}(s_1) \text{ nonexp., def. } d_F] \\
&\quad\; \max\{\, \tfrac{1}{2}\varepsilon, d(\mathcal{D}(s_2)(\eta), \tilde{\mathcal{D}}(s_2)(\eta)) \,\} \\
&= \text{[ind. hyp. for } s_2] \;\; \tfrac{1}{2}\varepsilon
\end{aligned}$$

Having established equation (2.2) it follows that $d_F(\mathcal{D}, \tilde{\mathcal{D}}) \leq \frac{1}{2}\varepsilon$ by definition of $d_F$. Therefore $\varepsilon \leq \frac{1}{2}\varepsilon$ and since $\varepsilon \geq 0$ we obtain $\varepsilon = 0$, $\mathcal{D} = \tilde{\mathcal{D}}$ and $\mathcal{D} = \mathit{fix}(\Psi)$. We conclude that, for $s \in \mathcal{L}_{ref}$, the mapping $\mathcal{D}(s)\colon \mathit{SemRef} \to \mathbb{P}_d$ is nonexpansive, as was to be shown. $\square$

This completes the justification of the definition of operational model $\mathcal{D}$. The nonexpansiveness result for $\mathcal{D}(s)$ for all programs $s$ is also used in the proof of lemma 5.2.23 in the next section.

### 5.2.3 Correctness and full abstractness of $\mathcal{D}$

Having developed both an operational and a denotational model for $\mathcal{L}_{ref}$ we will now address the question about their relationship. We will show:

(1) The denotational model $\mathcal{D}[\![\bullet]\!]$ is correct with respect to the operational model $\mathcal{O}[\![\bullet]\!]$, i.e., $\mathcal{D}[\![s_1]\!] = \mathcal{D}[\![s_2]\!] \Longrightarrow \mathcal{O}[\![s_1]\!] = \mathcal{O}[\![s_2]\!]$.

(2) The denotational model $\mathcal{D}\colon \mathcal{L}_{ref} \to \mathit{SemRef} \to \mathbb{P}_d$ is fully abstract for $\mathcal{O}[\![\bullet]\!]$, i.e., $\mathcal{D}(s_1) = \mathcal{D}(s_2) \iff \mathcal{O}[\![C[s_1]]\!] = \mathcal{O}[\![C[s_2]]\!]$ for all contexts $C[\bullet]$.

Note that in (1) only the semantical refinement $\eta_{id}$ plays a role, albeit hidden by the definition of $\mathcal{D}[\![\bullet]\!]$. However, in (2) the equality $\mathcal{D}(s_1) = \mathcal{D}(s_2)$ is equality of functions, and hence amounts to $\mathcal{D}(s_1)(\eta) = \mathcal{D}(s_2)(\eta)$ for all $\eta$ in *SemRef*. The latter result thus supports the point of view that two statements should be identified precisely when they have the same computations under all refinements.

Our first result will be shown using Banach's theorem 2.1.9 stating uniqueness of fixed points of contractions. In fact we will prove the stronger $\mathcal{O}[\![\bullet]\!] = \mathcal{D}[\![\bullet]\!]$. In subsection 5.2.1 we have shown $\mathcal{O} = \mathit{fix}(\Phi)$ where $\Phi\colon \mathit{Sem} \to \mathit{Sem}$ is a contraction based on the transition system for $\mathcal{L}_{ref}$, and, $\mathit{Sem} = \mathit{Res} \to \mathcal{P}_{nc}(\mathit{Act}^\infty)$. The idea is to extend the denotational semantics $\mathcal{D}$ acting on *Stat* to a function $\mathcal{E}$ acting on *Res*, and to show that $\Phi(\mathcal{E}) = \mathcal{E}$. By the uniqueness of fixed points the equality $\mathcal{O} = \mathcal{E}$ is obtained and $\mathcal{D}[\![\bullet]\!] = \mathcal{O}[\![\bullet]\!]$ follows.

A technical lemma, that helps with converting syntactical refinements into semantical ones, will be discussed first.

**Lemma 5.2.18** *Let the function* $\bullet \triangleright \bullet\colon \mathit{RefSeq} \times \mathit{SemRef} \to \mathit{SemRef}$ *be inductively given by*

$$\begin{aligned} \epsilon \triangleright \eta &= \eta \\ (R\,\langle a \rightsquigarrow s\rangle) \triangleright \eta &= R \triangleright \eta[\mathcal{D}(s)(\eta)/a] \end{aligned}$$

*Then the following equations hold*

*(a)* $\mathcal{D}(s\langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle)(\eta)$
$= \mathcal{D}(s)(\langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle \triangleright \eta)$

*(b)* $\mathcal{D}(s_1\langle a \rightsquigarrow s_2\rangle)(R \triangleright \eta) = \mathcal{D}(s_1)((\langle a \rightsquigarrow s_2\rangle\, R) \triangleright \eta)$

**Proof**

(a) Induction on $n$.

[0] Clear, since $\epsilon \triangleright \eta = \eta$ by definition.

$[n+1]$ $\mathcal{D}(s\langle a_1 \rightsquigarrow s_1\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle\langle a_{n+1} \rightsquigarrow s_{n+1}\rangle)(\eta)$

$=$ [definition $\mathcal{D}$] $\mathcal{D}(s\langle a_1 \rightsquigarrow s_1\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle)(\eta[\mathcal{D}(s_{n+1})(\eta)/a_{n+1}])$

$=$ [ind. hyp.] $\mathcal{D}(s)((\langle a_1 \rightsquigarrow s_1\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle) \triangleright (\eta[\mathcal{D}(s_{n+1})(\eta)/a_{n+1}]))$

$=$ [definition $\triangleright$] $\mathcal{D}(s)((\langle a_1 \rightsquigarrow s_1\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle\langle a_{n+1} \rightsquigarrow s_{n+1}\rangle) \triangleright \eta)$

(b) By application of part (a), first for $R$ then for $\langle a \rightsquigarrow s_2 \rangle\, R$. $\square$

We are now ready to introduce the extension of the semantical mapping $\mathcal{D}$ to resumptions and to show its coincidence with $\mathcal{O}$.

**Lemma 5.2.19** *Define the mapping $\mathcal{E}\colon Res \to \mathbb{P}_o$ as follows:*

$$\begin{array}{rcl} \mathcal{E}(\mathrm{E}) & = & \{\, \epsilon \,\} \\ \mathcal{E}(s : R) & = & \mathcal{D}(s)(R \triangleright \eta_{id}) \\ \mathcal{E}(r_1 \mathit{op}\, r_2) & = & \mathcal{E}(r_1)\, \mathit{op}\, \mathcal{E}(r_2) \;\; \textit{for } \mathit{op} \in \{\, ;, \square, \|\, \}. \end{array}$$

*Then it holds that $\Phi(\mathcal{E}) = \mathcal{E}$.*

**Proof** Notice that in the third clause for $\mathcal{E}$ the occurrence of $\mathit{op}$ at the right-hand side is a semantical operation given for $\mathbb{P}_d$ (which excludes the empty sequence $\epsilon$). By structural induction one easily verifies that $\mathcal{E}(r) \in \mathbb{P}_d$ for $r \neq \mathrm{E}$, so $\mathcal{E}(r_1 \mathit{op}\, r_2)$ is properly defined. The lemma is proven by weight-induction for configurations. We only exhibit a few typical cases here.

$[a : \langle a \rightsquigarrow s \rangle\, R]$ $\quad \Phi(\mathcal{E})(a : \langle a \rightsquigarrow s \rangle\, R)$

$$\begin{array}{rl} = & [\text{by (Act 2)}] \;\; \Phi(\mathcal{E})(s : R) \\ = & [\text{induction hypothesis}] \;\; \mathcal{E}(s : R) \\ = & \mathcal{D}(s)(R \triangleright \eta_{id}) \\ = & [\eta[p/a](a) = p] \;\; \mathcal{D}(a)((R \triangleright \eta_{id})[\mathcal{D}(s)(R \triangleright \eta_{id})/a]) \\ = & [\text{definition } \mathcal{D}] \;\; \mathcal{D}(a\langle a \rightsquigarrow s \rangle)(R \triangleright \eta_{id}) \\ = & [\text{lemma 5.2.18}] \;\; \mathcal{D}(a)((\langle a \rightsquigarrow s \rangle\, R) \triangleright \eta_{id}) \\ = & \mathcal{E}(a : \langle a \rightsquigarrow s \rangle\, R) \end{array}$$

$[s_1\langle a \rightsquigarrow s_2 \rangle : R]$ $\quad \Phi(\mathcal{E})(s_1\langle a \rightsquigarrow s_2 \rangle : R)$

$$\begin{array}{rl} = & \Phi(\mathcal{E})(s_1 : \langle a \rightsquigarrow s_2 \rangle\, R) \\ = & [\text{induction hypothesis}] \;\; \mathcal{E}(s_1 : \langle a \rightsquigarrow s_2 \rangle\, R) \\ = & \mathcal{D}(s_1)((\langle a \rightsquigarrow s_2 \rangle\, R) \triangleright \eta_{id}) \\ = & [\text{lemma 5.2.18}] \;\; \mathcal{D}(s_1\langle a \rightsquigarrow s_2 \rangle)(R \triangleright \eta_{id}) \\ = & \mathcal{E}(s_1\langle a \rightsquigarrow s_2 \rangle : R) \end{array}$$

$[r_1; r_2]$ $\quad \Phi(\mathcal{E})(r_1; r_2)$

$$\begin{array}{rl} = & \bigcup\{\, a\, \mathcal{E}(r) \mid r_1; r_2 \to a r \,\} \\ = & [\text{inspection of definition 5.2.3}] \;\; \bigcup\{\, a\, \mathcal{E}(r'; r_2) \mid r_1 \to a r' \,\} \\ = & \bigcup\{\, a\, (\mathcal{E}(r'); \mathcal{E}(r_2)) \mid r_1 \to a r' \,\} \;\; (\text{putting } \{\, \epsilon \,\}; p = p \text{ for } r' = \mathrm{E}) \\ = & [\text{definition } ;] \;\; \bigcup\{\, (a \cdot \mathcal{E}(r')); \mathcal{E}(r_2) \mid r_1 \to a r' \,\} \\ = & [\text{property } ;] \;\; (\bigcup\{\, a \cdot \mathcal{E}(r') \mid r_1 \to a r' \,\}); \mathcal{E}(r_2) \end{array}$$

$$
\begin{aligned}
&= \Phi(\mathcal{E})(r_1); \mathcal{E}(r_2) \\
&= \text{[induction hypothesis]} \ \ \mathcal{E}(r_1); \mathcal{E}(r_2) \\
&= \mathcal{E}(r_1; r_2) \qquad \square
\end{aligned}
$$

From lemma 5.2.19 and lemma 5.2.10 we get $\mathcal{O} = \mathcal{E}$ by Banach's theorem. From this the equivalence of the semantical models $\mathcal{O}[\![\bullet]\!]$ and $\mathcal{D}[\![\bullet]\!]$ can readily be derived.

**Theorem 5.2.20** $\mathcal{O}[\![\bullet]\!] = \mathcal{D}[\![\bullet]\!]$ *on* $\mathcal{L}_{ref}$.

**Proof** We have $\mathcal{O}[\![s]\!] = \mathcal{O}(s : \epsilon) = \mathcal{E}(s : \epsilon) = \mathcal{D}(s)(\eta_{id}) = \mathcal{D}[\![s]\!]$ for any $s \in \mathcal{L}_{ref}$. $\square$

The previous theorem will be used for the full abstractness result that we discuss in the remainder of this section. Equality of $\mathcal{D}[\![\bullet]\!]$ and $\mathcal{O}[\![\bullet]\!]$ implies correctness of $\mathcal{D}\colon \mathcal{L}_{ref} \to \mathit{SemRef} \to \mathbb{P}_d$ with respect to $\mathcal{O}[\![\bullet]\!]$ as will be shown in theorem 5.2.29. So, for any two statements $s', s'' \in \mathcal{L}_{ref}$,

$$\mathcal{D}(s') = \mathcal{D}(s'') \Rightarrow \mathcal{O}[\![C[s']]\!] = \mathcal{O}[\![C[s'']]\!] \ \text{ for all contexts } C[\bullet]. \tag{2.3}$$

We will show the reverse of equation (2.3) by contraposition: for any two statements $s', s''$ in $\mathcal{L}_{ref}$,

$$\mathcal{D}(s') \neq \mathcal{D}(s'') \Rightarrow \mathcal{O}[\![C[s']]\!] \neq \mathcal{O}[\![C[s'']]\!] \ \text{ for some context } C[\bullet]. \tag{2.4}$$

The proof of this implication is rather technical. The first observation is that $\mathcal{D}(s') \neq \mathcal{D}(s'')$ implies $\mathcal{D}(s')(\eta) \neq \mathcal{D}(s'')(\eta)$ for some semantical refinement $\eta$. It is from this semantical refinement $\eta$ that we shall construct the context $C[\bullet]$ indicated in equation (2.4). The main point is to identify a finite part of the semantical entity $\eta$ on which we can base the syntactical object $C[\bullet]$.

A second observation pertains to the distance between $\mathcal{D}(s')(\eta)$ and $\mathcal{D}(s'')(\eta)$. Since this distance is non-zero (for otherwise $\mathcal{D}(s')(\eta)$ and $\mathcal{D}(s'')(\eta)$ would be equal which they are not), we have that $d(\mathcal{D}(s')(\eta), \mathcal{D}(s'')(\eta)) = 2^{-n}$ for some $n \in \mathbb{N}$. We will show that only actions occurring in the finite sequences of the finite sets $\mathcal{D}(s')(\eta)[n]$ and $\mathcal{D}(s'')(\eta)[n]$ are of interest for this. As a consequence only a finite number of actions $a$ as argument for $\eta$ have to be considered and of their outcomes $\eta(a)$ only a finite initial segment is important.

Below we will introduce the notion of a finitary semantical refinement which incorporates these properties. We will argue that for a finitary semantical refinement $\eta$ there exists a sequence of action refinements $\langle a_i \rightsquigarrow s_i \rangle_{i=1}^{k}$ which captures sufficiently the role of $\eta$. In fact, it will be the case that the distance between $\mathcal{D}(s')(\eta)$ and $\mathcal{D}(s' \langle a_i \rightsquigarrow s_i \rangle_{i=1}^{k})(\eta_{id})$ is small, and likewise for $s''$. On this we will base our further analysis.

First we introduce a means to denote the actions that may occur in a prefix up to a certain position $n$ in the behavior of a statement $s$. This is $\mathit{act}_n(s)$. The union $\mathit{act}(s)$ over all positions $n$ is similar to the notion of syntactic sort (see e.g. [160]).

**Definition 5.2.21** *For $n \in \mathbb{N}$ and $s \in Stat$, the subset $act_n(s)$ of $Act$ is inductively given by*

$$\begin{array}{rcl} act_0(s) & = & \emptyset \\ act_{n+1}(a) & = & \{\, a \,\} \\ act_{n+1}(x) & = & act_{n+1}(g) \;\; \textit{where } g = D(x) \\ act_{n+1}(s_1; s_2) & = & act_{n+1}(s_1) \cup act_n(s_2) \\ act_{n+1}(s_1 \; op \; s_2) & = & act_{n+1}(s_1) \cup act_{n+1}(s_2) \;\; \textit{for } op = \{\, \Box, \| \,\} \\ act_{n+1}(s_1 \langle a \rightsquigarrow s_2 \rangle) & = & (act_{n+1}(s_1) \setminus \{\, a \,\}) \cup act_{n+1}(s_2). \end{array}$$

*The set $act(s) \subseteq Act$, for $s \in Stat$, is given by $act(s) = \bigcup_n act_n(s)$.*

The idea behind the definition of $act_n(s)$ is that it contains at least the actions that occur in the first $n$ positions in the meaning of $s$, viz. in $\mathcal{O}[\![s]\!]$ or equivalently in $\mathcal{D}(s)(\eta_{id})$. In fact $act_n(s)$ contains the alphabet of $(\mathcal{O}[\![s]\!])[n]$ but maybe more. The clause for $act_{n+1}(s_1; s_2)$ illustrates this most clearly. For example, we have $act_2((a; b); c) = \{\, a, b, c \,\}$ but $(\mathcal{O}[\![(a; b); c]\!])[2] = \{\, abc \,\}[2] = \{\, ab \,\}$ which has $\{\, a, b \,\}$ as its alphabet. One can easily verify by induction on $n$ and $wgt(s)$ that $act_n(s)$ is a finite set.

The next lemma states that if two semantical refinements $\eta_1, \eta_2$ do not differ too much on the $n$-th action set of a statement $s$, the denotational meanings with respect to the respective semantical refinement do not differ much either. For easier readability we introduce for $n \in \mathbb{N}$ the notation $p_1 =_n p_2$ to denote $d(p_1, p_2) \leq 2^{-n}$ and, similarly, $\eta_1 =_n \eta_2$ on $A$ to denote $\forall a \in A$: $d(\eta_1(a), \eta_2(a)) \leq 2^{-n}$ for $\eta_1, \eta_2 \in SemRef$ and $A \subseteq Act$. (Note that by ultrametricity of $d$, the relation $=_n$ is transitive and, moreover, an equivalence relation.)

**Lemma 5.2.22** *Let $n \in \mathbb{N}$ and $s \in Stat$. If $\eta_1 =_n \eta_2$ on $act_n(s)$ then $\mathcal{D}(s)(\eta_1) =_n \mathcal{D}(s)(\eta_2)$, for all $\eta_1, \eta_2 \in SemRef$.*

**Proof** Induction on $n$.

[0] Trivial. We have that $0 \leq d(p_1, p_2) \leq 1$ for all processes $p_1, p_2$ in $\mathbb{P}_d$

[$n + 1$] Sub-induction on $wgt(s)$. We only present the cases for sequential composition and action refinement. The other cases are similar or simpler.

[$s_1; s_2$] By definition of $act_{n+1}(s_1; s_2)$ we obtain from the assumption of the lemma

$$\eta_1(a_1) =_{n+1} \eta_2(a_1) \text{ and } \eta_1(a_2) =_{n+1} \eta_2(a_2), \text{ hence } \eta_1(a_2) =_n \eta_2(a_2)$$

for any $a_1 \in act_{n+1}(s_1)$ and any $a_2 \in act_n(s_2)$. So, by the induction hypotheses for $s_1$ and $n$, respectively, we have

$$\mathcal{D}(s_1)(\eta_1) =_{n+1} \mathcal{D}(s_1)(\eta_2) \text{ and } \mathcal{D}(s_2)(\eta_1) =_n \mathcal{D}(s_2)(\eta_2) \qquad (2.5)$$

Therefore

$$\begin{array}{rl} & d(\mathcal{D}(s_1; s_2)(\eta_1), \mathcal{D}(s_1; s_2)(\eta_2)) \\ = & d(\mathcal{D}(s_1)(\eta_1); \mathcal{D}(s_2)(\eta_1), \mathcal{D}(s_1)(\eta_2); \mathcal{D}(s_2)(\eta_2)) \end{array}$$

$$\begin{aligned} &\leq \text{[lemma 5.2.13]} \\ &\quad \max\{\, d(\mathcal{D}(s_1)(\eta_1), \mathcal{D}(s_1)(\eta_2)), \tfrac{1}{2} d(\mathcal{D}(s_2)(\eta_1), \mathcal{D}(s_2)(\eta_2)) \,\} \\ &\leq \text{[equation (2.5)]} \;\; \max\{\, 2^{-(n+1)}, \tfrac{1}{2} \cdot 2^{-n} \,\} \\ &= 2^{-(n+1)} \end{aligned}$$

$[s_1\langle a \rightsquigarrow s_2\rangle]$ Since, by the definition of $act_{n+1}(s_1\langle a \rightsquigarrow s_2\rangle)$, the assumptions imply that $d(\eta_1(a'), \eta_2(a')) \leq 2^{-(n+1)}$ for $a' \in act_{n+1}(s_2)$, we have

$$d(\mathcal{D}(s_2)(\eta_1), \mathcal{D}(s_2)(\eta_2)) \;\leq\; 2^{-(n+1)}$$

by the induction hypothesis for $s_2$. From this and the condition on each $\eta_1, \eta_2$, we obtain,

$$d(\eta_1[\mathcal{D}(s_2)(\eta_1)/a](a'), \eta_2[\mathcal{D}(s_2)(\eta_2)/a](a')) \;\leq\; 2^{-(n+1)}$$

for any $a' \in act_{n+1}(s_1) \subseteq act_{n+1}(s_1\langle a \rightsquigarrow s_2\rangle) \cup \{\, a \,\}$. Therefore, by the induction hypothesis for $s_1$, we conclude

$$\begin{aligned} &d(\mathcal{D}(s_1\langle a \rightsquigarrow s_2\rangle)(\eta_1), \mathcal{D}(s_1\langle a \rightsquigarrow s_2\rangle)(\eta_2)) \\ &= d(\mathcal{D}(s_1)(\eta_1[\mathcal{D}(s_2)(\eta_1)/a]), \mathcal{D}(s_1)(\eta_2[\mathcal{D}(s_2)(\eta_2)/a])) \\ &\leq 2^{-(n+1)} \end{aligned}$$

□

The next two lemmas, viz. lemmas 5.2.23 and 5.2.24, and their immediate consequence corollary 5.2.25 provide conditions for the interchange of a syntactical refinement sequence $\langle a_i \rightsquigarrow s_i\rangle_{i=1}^n$ and a variant $[p_i/a_i]_{i=1}^n$ of the current semantical refinement. Lemma 5.2.23 paves the way for an inductive argument based on the number of syntactical refinements in the sequence $\langle a_i \rightsquigarrow s_i\rangle_{i=1}^n$ for lemma 5.2.24. In the situation of lemma 5.2.23 we consider $\mathcal{D}(s)(\eta_1)$ versus $\mathcal{D}(s)(\eta_2)$ in the metric space $\mathbb{P}_d$. For the proof we make use of the nonexpansiveness/contractiveness result for the semantical operations as established in subsection 5.2.2.

**Lemma 5.2.23** *It holds that $\mathcal{D}(s)(\eta_1) = \mathcal{D}(s)(\eta_2)$, for $s \in Stat$ and $\eta_1, \eta_2 \in SemRef$ such that $\eta_1 = \eta_2$ on $act(s)$.*

**Proof** Put

$$\varepsilon = \sup\{\, d(\mathcal{D}(s)(\eta_1), \mathcal{D}(s)(\eta_2)) \mid \eta_1 = \eta_2 \text{ on } act(s), s \in Stat, \eta_1, \eta_2 \in SemRef \}$$

We show, by weight-induction, that $d(\mathcal{D}(s)(\eta_1), \mathcal{D}(s)(\eta_2)) \leq \frac{1}{2}\varepsilon$ for any $s, \eta_1, \eta_2$ such that $\eta_1, \eta_2$ coincide on $act(s)$. Hence we have that $0 \leq \varepsilon \leq \frac{1}{2}\varepsilon$ and thus $\varepsilon = 0$ from which the lemma follows.

The case for $a$ is trivial. The case for $x$ is straightforward from the induction hypothesis for $D(x)$, the cases $s_1; s_2$ and $s_1 \mathbin{op} s_2$ (with $op \in \{\, \square, \| \,\}$) can be checked using lemma 5.2.13. We focus on the refinement statement. Note that $act(s_1) \setminus \{\, a \,\}, act(s_2)$ is a subset of $act(s_1\langle a \rightsquigarrow s_2\rangle)$.

$$d(\mathcal{D}(s_1\langle a \rightsquigarrow s_2\rangle)(\eta_1), \mathcal{D}(s_1\langle a \rightsquigarrow s_2\rangle)(\eta_2))$$

$$\begin{aligned}
&\leq \text{[ultrametricity]} \\
&\quad \max\{\ d(\mathcal{D}(s_1)(\eta_1[\mathcal{D}(s_2)(\eta_1)/a]), \mathcal{D}(s_1)(\eta_1[\mathcal{D}(s_2)(\eta_2)/a])), \\
&\qquad\qquad d(\mathcal{D}(s_1)(\eta_1[\mathcal{D}(s_2)(\eta_2)/a]), \mathcal{D}(s_1)(\eta_2[\mathcal{D}(s_2)(\eta_2)/a]))\ \} \\
&\leq [\mathcal{D} \text{ nonexpansive in } \eta, \text{ induction hypothesis for } s_1] \\
&\quad \max\{\ d_F(\eta_1[\mathcal{D}(s_2)(\eta_1)/a], \eta_1[\mathcal{D}(s_2)(\eta_2)/a]), \tfrac{1}{2}\varepsilon\ \} \\
&= \max\{\ d(\mathcal{D}(s_2)(\eta_1), \mathcal{D}(s_2)(\eta_2)), \tfrac{1}{2}\varepsilon\ \} \\
&\leq \text{[induction hypothesis for } s_2] \quad \max\{\ \tfrac{1}{2}\varepsilon, \tfrac{1}{2}\varepsilon\ \} \\
&= \tfrac{1}{2}\varepsilon \qquad \square
\end{aligned}$$

Recall that the notation $\langle a_i \rightsquigarrow s_i\rangle_{i=1}^n$ is used to abbreviate the syntactical refinement sequence $\langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle$. In general, for a statement $s\langle a_i \rightsquigarrow s_i\rangle_{i=1}^n$, i.e. for $s\langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle$, the order in which the constituent refinements $\langle a_i \rightsquigarrow s_i\rangle$ occur cannot be changed without altering the meaning of $s\langle a_i \rightsquigarrow s_i\rangle_{i=1}^n$. Lemma 5.2.24 provides sufficient conditions to ensure that such interchanging does not matter. In fact, under the restriction of the lemma, we have that syntactical *iterated* refinement coincides with semantical *simultaneous* refinement. The result will be used for the transition from a syntactical refinement $s\langle \bar{a}_i \rightsquigarrow \bar{s}_i\rangle_{i=1}^\ell$ with respect to some $\eta$ to a semantical refinement $\eta[\bar{p}_i/\bar{a}_i]_{i=1}^\ell$ for the statement $s$.

**Lemma 5.2.24** *Suppose $a_1, \ldots, a_n \in Act$ are pairwise distinct, and $s_1, \ldots, s_n \in Stat$ are such that $act(s_i) \cap \{ a_1, \ldots, a_n \} = \emptyset$ for $1 \leq i \leq n$. Then it holds that*

$$\mathcal{D}(s\langle a_i \rightsquigarrow s_i\rangle_{i=1}^n)(\eta) = \mathcal{D}(s)(\eta[p_i/a_i]_{i=1}^n)$$

*where $p_i = \mathcal{D}(s_i)(\eta)$ for $1 \leq i \leq n$.*

**Proof** Induction on $n$. The case [0] is trivial.

$[n+1]$ We have

$$\begin{aligned}
&\mathcal{D}(s\langle a_i \rightsquigarrow s_i\rangle_{i=1}^n\langle a_{n+1} \rightsquigarrow s_{n+1}\rangle)(\eta) \\
&= \quad \mathcal{D}(s\langle a_i \rightsquigarrow s_i\rangle_{i=1}^n)(\eta[p_{n+1}/a_{n+1}]) \\
&= \quad \text{[ind. hyp.]} \quad \mathcal{D}(s)(\eta[p_{n+1}/a_{n+1}][\mathcal{D}(s_i)(\eta[p_{n+1}/a_{n+1}]/a_i]_{i=1}^n)
\end{aligned}$$

Since the action $a_{n+1}$ is not an element of $act(s_i)$, we have that the refinements $\eta[p_{n+1}/a_{n+1}]$ and $\eta$ coincide on $act(s_i)$. Therefore, by lemma 5.2.23, it follows that $\mathcal{D}(s_i)(\eta[p_{n+1}/a_{n+1}]) = \mathcal{D}(s_i)(\eta) = p_i$ for $1 \leq i \leq n$. So

$$\begin{aligned}
&\mathcal{D}(s\langle a_i \rightsquigarrow s_i\rangle_{i=1}^n\langle a_{n+1} \rightsquigarrow s_{n+1}\rangle)(\eta) \\
&= \quad \mathcal{D}(s)(\eta[p_{n+1}/a_{n+1}][p_i/a_i]_{i=1}^n) \\
&= \quad [a_{n+1} \neq a_1, \ldots, a_n] \quad \mathcal{D}(s)(\eta[p_i/a_i]_{i=1}^{n+1}) \qquad \square
\end{aligned}$$

As a special case of lemma 5.2.24 we have the following result.

**Corollary 5.2.25** *Suppose $\bar{a}_i, a'_j$ $(1 \leq i, j \leq n)$ are pairwise distinct. Then it holds that*

$$\mathcal{D}(s\langle \bar{a}_i \rightsquigarrow a'_i \rangle_{i=1}^n)(\eta) = \mathcal{D}(s)(\eta[\eta(a'_i)/\bar{a}_i]_{i=1}^n).$$

**Proof** Since $a'_j \notin \{ \bar{a}_1, \ldots, \bar{a}_n \}$, $1 \leq j \leq n$, and $\bar{a}_1, \ldots, \bar{a}_n$ are pairwise distinct we obtain the result directly from lemma 5.2.24. □

To facilitate the syntactical representation of a semantical refinement that essentially involves only finitely many actions we have the following definition. In the case where we can focus on a fixed and finite set of actions we can assure for any semantical refinement $\eta$ the existence of a so-called finitary semantical refinement, say $\eta'$, arbitrarily close to $\eta$ on this fixed set of actions. This does not hold in general, as the set *Act* is assumed to be infinite. (In case the set *Act* is finite one has to introduce auxiliary actions in order to deal with iterated versus simultaneous refinements.)

**Definition 5.2.26** *A semantical refinement $\eta \in SemRef$ is called* finitary *if the following conditions are fulfilled:*

- *for all $a \in Act$ it holds that $\eta(a) \subseteq \mathcal{P}_f(Act^*)$, i.e. $\eta(a)$ is a finite set of finite sequences over Act;*
- *$\eta(a) \neq \{ a \}$ for finitely many $a \in Act$.*

*For a finitary semantical refinement $\eta$ its domain $dom(\eta)$ is given by*

$$dom(\eta) \ = \ \{ a \in Act \mid \eta(a) \neq \{ a \} \}$$

For the full abstractness result (cf. lemma 5.2.28 and theorem 5.2.29) we want to exploit the relationship between $\mathcal{D}[\![\bullet]\!]$ and $\mathcal{O}[\![\bullet]\!]$ already established in theorem 5.2.20. As the definition of $\mathcal{D}[\![\bullet]\!]$ is based on the identity refinement $\eta_{id}$ the next lemma is useful for the translation of an arbitrary (finitary) refinement to this origin.

**Lemma 5.2.27** *Let $\eta \in SemRef$ be a finitary semantical refinement and $A \subseteq Act$ a finite set of actions. Then there exists, for all $n \in \mathbb{N}$, a refinement sequence $\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k$ such that*

$$\mathcal{D}(s)(\eta) \ =_n \ \mathcal{D}(s\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k)(\eta_{id})$$

*for all $s \in Stat$ with $act_n(s) \subseteq A$.*

**Proof** For any finite set $p = \{ q_1, \ldots, q_n \}$ of non-empty finite sequences over *Act* we can construct a statement *stat*$(p)$ such that

$$\mathcal{D}(stat(p))(\eta_{id}) \ = \ p \tag{2.6}$$

as follows: Put $stat(p) = stat'(q_1) \Box \cdots \Box stat'(q_n)$ (with association to the right say) where, for $q \in Act^+$, the statement $stat'(q)$ is given by $stat'(a) = a$ and $stat'(a\,q) = a; stat'(q)$. It is straightforward (e.g., by simultaneous induction on the number and length of the $q_i$'s) to show that *stat*$(p)$ satisfies equation (2.6).

Now let $\eta$ be a finitary semantical refinement, $A$ a finite set of actions and $n$ a number in $\mathbb{N}$. Suppose $dom(\eta) = \{ \bar{a}_1, \ldots, \bar{a}_\ell \}$ and $\eta(\bar{a}_i) = \bar{p}_i$, $1 \leq i \leq \ell$. Let $B$ be any finite set of actions such that $\bar{p}_1, \ldots, \bar{p}_\ell \subseteq B^*$. Pick $a'_1, \ldots, a'_\ell \in \mathit{Act} \setminus (\mathit{dom}(\eta) \cup A \cup B)$. Note that $\mathit{Act}$ is infinite whereas $\mathit{dom}(\eta)$, $A$ and $B$ are finite. Put $\bar{s}_i = \mathit{stat}(\bar{p}_i)$ for $1 \leq i \leq \ell$. *Claim*: for $s \in \mathit{Stat}$ such that $\mathit{act}_n(s) \subseteq A$ it holds that

$$\mathcal{D}(s)(\eta) \ =_n \ \mathcal{D}(s\langle \bar{a}_i \rightsquigarrow a'_i \rangle_{i=1}^{\ell} \langle a'_i \rightsquigarrow \bar{s}_i \rangle_{i=1}^{\ell})(\eta_{id})$$

*Proof of the claim*: We have

$$\begin{aligned}
&\mathcal{D}(s\langle \bar{a}_i \rightsquigarrow a'_i \rangle_{i=1}^{\ell} \langle a'_i \rightsquigarrow \bar{s}_i \rangle_{i=1}^{\ell})(\eta_{id}) \\
&\quad = \text{[lemma 5.2.24]} \ \ \mathcal{D}(s\langle \bar{a}_i \rightsquigarrow a'_i \rangle_{i=1}^{\ell})(\eta_{id}[\bar{p}_i/a'_i]_{i=1}^{\ell}) \\
&\quad = \text{[corollary 5.2.25, note } \mathcal{D}(a)(\eta[p/a]) = p\,] \ \ \mathcal{D}(s)(\eta_{id}[\bar{p}_i/a'_i]_{i=1}^{\ell}[\bar{p}_i/\bar{a}_i]_{i=1}^{\ell}) \\
&\quad = [\mathit{dom}(\eta) = \{\, \bar{a}_1, \ldots, \bar{a}_\ell \,\}] \ \ \mathcal{D}(s)(\eta[\bar{p}_i/a'_i]_{i=1}^{\ell})
\end{aligned}$$

Note that $\{\, a'_1, \ldots, a'_\ell \,\} \cap \mathit{act}_n(s) = \emptyset$. So, by lemma 5.2.22,

$$\mathcal{D}(s)(\eta) =_n \mathcal{D}(s)(\eta[\bar{p}_i/a'_i]_{i=1}^{\ell})$$

and, consequently,

$$\mathcal{D}(s)(\eta) =_n \mathcal{D}(s\langle \bar{a}_i \rightsquigarrow a'_i \rangle_{i=1}^{\ell} \langle a'_i \rightsquigarrow \bar{s}_i \rangle_{i=1}^{\ell})(\eta_{id})$$

This proves the claim.

So, if we put $k = 2\ell$ and define $a_i = \bar{a}_i$, $a_{\ell+i} = a'_i$, $s_i = a'_i$, $s_{\ell+i} = \bar{s}_i$ for $1 \leq i \leq \ell$, we obtain from the claim

$$\mathcal{D}(s)(\eta) =_n \mathcal{D}(s\langle a_i \rightsquigarrow s_i \rangle_{i=1}^{k})(\eta_{id})$$

which was to be shown. $\square$

By now we have gathered sufficient technical results to prove that two statements that differ semantically for the model $\mathcal{D}\colon \mathcal{L}_{ref} \to \mathit{SemRef} \to \mathbb{P}_d$ can also be distinguished by the model $\mathcal{O}[\![\bullet]\!]$ in some context $C[\bullet]$.

**Lemma 5.2.28** *If $s', s'' \in \mathit{Stat}$ satisfy $\mathcal{D}(s') \neq \mathcal{D}(s'')$ then $\mathcal{O}[\![C[s']]\!] \neq \mathcal{O}[\![C[s'']]\!]$ for some context $C[\bullet]$.*

**Proof** Suppose $s', s''$ are statements with $\mathcal{D}(s') \neq \mathcal{D}(s'')$. Pick $\eta \in \mathit{SemRef}$ such that $\mathcal{D}(s')(\eta) \neq \mathcal{D}(s'')(\eta)$. Let $n$ be such that $d(\mathcal{D}(s')(\eta), \mathcal{D}(s'')(\eta)) = 2^{-n}$. Define the finitary semantical refinement $\eta'$ by

$$\eta'(a) \quad = \quad \begin{cases} \eta(a)[n+1] & \text{if } a \in \mathit{act}_n(s', s'') \\ \{\, a \,\} & \text{otherwise} \end{cases}$$

Then $\mathit{dom}(\eta') \subseteq \mathit{act}_n(s', s'')$ and, using lemma 5.2.22, we have

$$\mathcal{D}(s')(\eta) =_{n+1} \mathcal{D}(s')(\eta') \text{ and } \mathcal{D}(s'')(\eta) =_{n+1} \mathcal{D}(s'')(\eta')$$

Therefore, by ultrametricity, we get

$$d(\mathcal{D}(s')(\eta'), \mathcal{D}(s'')(\eta')) = 2^{-n}$$

Now let, using lemma 5.2.27, $\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k$ be a refinement sequence such that

$$\begin{array}{l} \mathcal{D}(s')(\eta') =_{n+1} \mathcal{D}(s'\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k)(\eta_{id}) \text{ and} \\ \mathcal{D}(s'')(\eta') =_{n+1} \mathcal{D}(s''\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k)(\eta_{id}) \end{array}$$

Again we obtain by ultrametricity

$$d(\mathcal{D}(s'\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k)(\eta_{id}), \mathcal{D}(s''\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k)(\eta_{id})) = 2^{-n}$$

So, in particular, $\mathcal{D}(s'\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k)(\eta_{id}) \neq \mathcal{D}(s''\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k)(\eta_{id})$. By definition of $\mathcal{D}[\![\bullet]\!]$ we thus have

$$\mathcal{D}[\![s'\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k]\!] \neq \mathcal{D}[\![s''\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k]\!]$$

from which $\mathcal{O}[\![s'\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k]\!] \neq \mathcal{O}[\![s''\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k]\!]$ follows by theorem 5.2.20. Hence,

$$\mathcal{O}[\![C[s']]\!] \neq \mathcal{O}[\![C[s'']]\!]$$

if we put $C[\bullet] = (\bullet)\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k$. □

Finally we are in a position to finish the plan set out earlier, in order to establish full abstractness of $\mathcal{D}$.

**Theorem 5.2.29** *$\mathcal{D}\colon \mathcal{L}_{ref} \to SemRef \to \mathbb{P}_d$ is fully abstract with respect to $\mathcal{O}[\![\bullet]\!]$.*

**Proof** Let $s_1, s_2 \in Stat$ and $C[\bullet]$ a $\mathcal{L}_{ref}$-context. If we have $\mathcal{D}(s_1) = \mathcal{D}(s_2)$ then by compositionality $\mathcal{D}(C[s_1]) = \mathcal{D}(C[s_2])$ holds. This means that $\mathcal{D}(C[s_1])(\eta) = \mathcal{D}(C[s_2])(\eta)$ holds for all semantical refinements $\eta$ and in particular for $\eta_{id}$. Thus we have $\mathcal{D}[\![C[s_1]]\!] = \mathcal{D}(C[s_1])(\eta_{id}) = \mathcal{D}(C[s_2])(\eta_{id}) = \mathcal{D}[\![C[s_2]]\!]$, and, by theorem 5.2.20, $\mathcal{O}[\![C[s_1]]\!] = \mathcal{O}[\![C[s_2]]\!]$. If $\mathcal{D}(s_1) \neq \mathcal{D}(s_2)$, then, by lemma 5.2.28, it follows that $\mathcal{O}[\![C[s_1]]\!] \neq \mathcal{O}[\![C[s_2]]\!]$ for some context $C[\bullet]$. Therefore we have, for all $s_1, s_2 \in Stat$,

$$\mathcal{D}(s_1) = \mathcal{D}(s_2) \iff \mathcal{O}[\![C[s_1]]\!] = \mathcal{O}[\![C[s_2]]\!] \text{ for all } \mathcal{L}_{ref}\text{-contexts } C[\bullet]$$

Thus $\mathcal{D}\colon \mathcal{L}_{ref} \to SemRef \to \mathbb{P}_d$ is fully abstract with respect to $\mathcal{O}[\![\bullet]\!]$. □

## 5.3 Action refinement and probabilistic choice

In this section the applicability of metric techniques for dealing with probability in the development of an operational and a denotational semantics in the setting of a language with discrete probabilistic choice and action refinement is studied. The previous section illustrated how the metric machinery can be used for construction of an operational and a denotational semantics, establishing correctness of the denotational model with respect to the operational one and obtaining a full abstractness result. The aim of this section is to investigate the flexibility of this metric machinery as well as the metric tools for modeling probabilistic choice developed in chapter 3. To this end the results of chapter 3 and section 5.2 are combined and adapted. It turns out that the various techniques are indeed orthogonal: replacing nondeterminacy by probability does not affect the proof methods.

In the previous section the language $\mathcal{L}_{ref}$ with action refinement and nondeterministic and parallel composition was treated. The domain of meanings consists of compact sets of sequences of actions. The denotational semantics given for $\mathcal{L}_{ref}$ was shown to be fully abstract with respect to the operational semantics presented there. The present section, devoted to the process language $\mathcal{L}_{pr}$, seeks to adapt these results from a nondeterministic framework to a probabilistic one. The constructs of nondeterministic and parallel composition of $\mathcal{L}_{ref}$ are removed and probabilistic choice is added in $\mathcal{L}_{pr}$. Now, probability measures of compact support over sequences of actions are used as semantical objects, but the results in this section show that essentially the metrical instruments, such as the use of Banach's fixed point theorem, obtaining equality by reasoning about distances and the method of proving full abstraction as introduced in [112] and used in the previous section, remain the same. Moreover, our analysis illustrates that the techniques used in the previous section to deal with action refinement may be mixed with other elements of the metric approach.

In chapter 1 related work in the area of probabilistic choice was discussed. Several papers deal with full abstractness in a setting with probabilistic choice. In [147], extending the earlier [142], a full abstractness result is obtained for a metric denotational model with respect to a variant of probabilistic bisimulation as proposed by [149]. The semantical interpretation of probability in [147] is based on a different quantitative paradigm than the one of the present section, namely worst-case best-case intervals. Moreover, the denotational semantics is developed ad hoc and does not appeal to a general methodology of constructing operational and denotational semantics. In [28] a metric denotational semantics for an extension of CCS with action guarded probabilistic choice is shown to be fully abstract with respect to probabilistic bisimulation. The present section establishes full abstraction with respect to an operational model. Identification of bisimilar processes (here for probabilistic bisimulation à la Larsen and Skou) is automatic when working with metric domains obtained as final coalgebras of contracting functors (cf. [180, 191, 110]). The domains in this section can be obtained in this way. The papers [57, 168, 99] deal with fully abstract models for probabilistic choice in the setting of testing semantics.

The remainder of this section is organized as follows. The language $\mathcal{L}_{pr}$ and its operational semantics are introduced in subsection 5.3.1, while subsection 5.3.2 is devoted to the denotational semantics $\mathcal{D}$ for $\mathcal{L}_{pr}$. The correctness of the model $\mathcal{D}$ with respect to the semantics $\mathcal{O}$ is subject of subsection 5.3.3, while the full abstraction result is discussed in subsection 5.3.4.

### 5.3.1 Syntax and operational semantics of $\mathcal{L}_{pr}$

In this subsection we introduce the process language $\mathcal{L}_{pr}$ and present its operational semantics $\mathcal{O}$. The model $\mathcal{O}$ will serve as a point of reference for our understanding of $\mathcal{L}_{pr}$ and for the semantical considerations in later subsections. We start with the syntax for $\mathcal{L}_{pr}$. As usual *Act* and *PVar* denote the set of actions and procedure variables respectively. Again the set *Act* is assumed to be infinite. (As in the previous section, the full abstractness result requires that a fresh action from outside some given finite set of actions can always be picked.)

**Definition 5.3.1**

*(a) The set of statements Stat, ranged over by s, is given by*

$$s \ ::= \ a \mid x \mid s;s \mid s \oplus_\rho s \mid s\langle a \rightsquigarrow s\rangle$$

*with* $0 < \rho < 1$.

*(b) The set of guarded statements GStat, ranged over by g, is given by*

$$g \ ::= \ a \mid g;s \mid g \oplus_\rho g \mid g\langle a \rightsquigarrow g\rangle$$

*with* $0 < \rho < 1$.

*(c) The set of declarations Decl, ranged over by D, is given by*

$$\mathit{Decl} \quad = \quad \mathit{PVar} \to \mathit{GStat}$$

*(d) The language* $\mathcal{L}_{pr}$ *is given by*

$$\mathcal{L}_{pr} \quad = \quad \mathit{Decl} \times \mathit{Stat}$$

The language $\mathcal{L}_{pr}$ contains the usual ingredients of abstract, uninterpreted actions $a$, sequential composition $s_1; s_2$ and recursion via procedure variables. More specific constructions in $\mathcal{L}_{pr}$ are the construction of probabilistic choice $s_1 \oplus_\rho s_2$ and of action refinement $s_1\langle a \rightsquigarrow s_2\rangle$.

The intuition behind the construct $s_1 \oplus_\rho s_2$ (see chapter 3) is that upon its execution, with probability $\rho$ the alternative $s_1$ is taken, and with the complementary probability $1-\rho$ the alternative $s_2$ is executed. The idea underlying action refinement (see section 5.2) is that in $s_1\langle a \rightsquigarrow s_2\rangle$ the actions of $s_1$ are performed, but with the execution of $s_2$ replacing the execution of actions $a$ of $s_1$. So, for example, $a \oplus_{\frac{1}{4}} (b;c)$ delivers, on the average, in 25% of the cases $a$ and in 75% the sequence $bc$. Instead $((a \oplus_{\frac{1}{4}} (b;c))\langle a \rightsquigarrow b;d\rangle)\langle c \rightsquigarrow d\rangle$ will yield $bd$ for all of its executions.

The body of a procedure variable is given by a declaration in *Decl*. As usual a fixed declaration $D$ is assumed and dropped from the notation.

As in the transition system $\mathcal{T}_{\text{ref}}$, the transition system for $\mathcal{L}_{pr}$ makes use of refinement sequences in order to keep track of the relevant action refinements. Refinement sequences are as in the previous section but are now based on statements in $\mathcal{L}_{pr}$. Resumptions are also similar.

**Definition 5.3.2**

*(a) The class RefSeq of refinement sequences, ranged over by R, is given by*

$$R \ ::= \ \epsilon \mid \langle a \rightsquigarrow s\rangle\, R$$

*(b) The class Res of resumptions, ranged over by r, is given by*

$$r \ ::= \ \mathrm{E} \mid s : R \mid r;r \mid r \oplus_\rho r$$

Below we also employ the notation $\langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle$ for arbitrary refinement sequences, and the notion $R\,\langle a \rightsquigarrow s\rangle$ for nonempty refinement sequences. A configuration of the transition system is a resumption with a declaration, thus $\mathit{Conf} = \mathit{Decl} \times \mathit{Res}$. The declaration part is suppressed in the notation. A transition label is either an action in $\mathit{Act}$ or a ratio in $(0,1)$, i.e. $\mathit{Lab} = \mathit{Act} \cup (0,1)$.

Recall that $r_1 \rightarrow_0 r_2$ is a shorthand notation for the rule

$$\frac{r_2 \xrightarrow{\lambda} r}{r_1 \xrightarrow{\lambda} r}$$

**Definition 5.3.3** *The transition system $\mathcal{T}_{pr}$ is given by $\mathcal{T}_{pr} = (\mathit{Conf}, \mathit{Lab}, \rightarrow, \mathit{Spec})$ where Spec contains the following axioms and rules:*

- $$a : \epsilon \xrightarrow{a} \mathrm{E} \qquad \textit{(Act 1)}$$

  $$a : \langle a' \rightsquigarrow s'\rangle \cdot R \rightarrow_0 s' : R \qquad \textit{if } a = a' \qquad \textit{(Act 2)}$$

  $$a : \langle a' \rightsquigarrow s'\rangle \cdot R \rightarrow_0 a : R \qquad \textit{if } a \neq a' \qquad \textit{(Act 3)}$$

- $$x : R \rightarrow_0 D(x) : R \qquad \textit{(Rec)}$$

- $$(s_1\ \mathit{op}\ s_2) : R \rightarrow_0 (s_1 : R)\ \mathit{op}\ (s_2 : R) \qquad \textit{for } \mathit{op} \in \{;, \oplus_\rho\} \qquad \textit{(Op)}$$

- $$s\langle a' \rightsquigarrow s'\rangle : R \rightarrow_0 s : \langle a' \rightsquigarrow s'\rangle \cdot R \qquad \textit{(Ref)}$$

- $$\frac{r_1 \xrightarrow{\lambda} r_1'}{r_1; r_2 \xrightarrow{\lambda} r_1'; r_2} \qquad \textit{(Seq)}$$

  *where $r_1'; r_2$ should be read as $r_2$ if $r_1' = \mathrm{E}$.*

- $$r_1 \oplus_\rho r_2 \xrightarrow{\rho} r_1 \qquad r_1 \oplus_\rho r_2 \xrightarrow{1-\rho} r_2 \qquad \textit{(PChoice 1,2)}$$

The axiom and rules dealing with action refinement, recursion and sequential composition are as in the previous section but now referring to configurations and labels for $\mathcal{T}_{\mathrm{pr}}$. The axiom (Act 1) and rules (Act 2) and (Act 3) reflect the stack-like bookkeeping for action refinement. The leftmost component of a refinement sequence applies, if the action to be refined, viz. $a'$, matches the action in the control part of the resumption, viz. $a$; otherwise the action refinement is skipped. If no action refinement is left on the stack, i.e. the refinement sequence in the resumption is the empty sequence $\epsilon$, the action $a$ itself is executed as indicated by the label $a \in \mathit{Act} \subseteq \mathit{Lab}$ of the axiom (Act 1).

Procedure variables are handled by means of body replacement. Sequential and probabilistic composition in the control part of a resumption, i.e. for resumptions of the format $(s_1\ \mathit{op}\ s_2) : R$, distribute over the pair-constructor $:$ of resumptions 'yielding' $(s_1 : R)\ \mathit{op}\ (s_2 : R)$. Similarly, an action refinement $s\langle a' \rightsquigarrow s'\rangle$ in the control part of a resumption $s\langle a' \rightsquigarrow s'\rangle : R$ amounts to an update of the refinement sequence of the resumption, where the action refinement $\langle a' \rightsquigarrow s' \rangle$ is prefixed to the sequence $R$. A sequential composition of resumptions is handled as usual.

A probabilistic choice between resumptions is resolved by selection of one of the probabilistic alternatives while delivering the probability of the alternative as a label $\rho$ or $1-\rho$ in the open interval $(0,1) \subseteq \mathit{Lab}$. A similar treatment of probabilistic choice can be found in sections 4.3 and 4.4.

**Examples 5.3.4** *Consider the resumption $(a \oplus_{\frac{1}{4}} (b;c)) : \epsilon$. Since*

$$(a : \epsilon) \oplus_{\frac{1}{4}} ((b;c) : \epsilon) \xrightarrow{\frac{1}{4}} (a : \epsilon)$$

*by (PChoice 1), we have*

$$(a \oplus_{\frac{1}{4}} (b;c)) : \epsilon \xrightarrow{\frac{1}{4}} a : \epsilon.$$

*Similarly, since*

$$(a : \epsilon) \oplus_{\frac{1}{4}} ((b;c) : \epsilon) \xrightarrow{\frac{3}{4}} ((b;c) : \epsilon)$$

*by (PChoice 2), we have*

$$(a \oplus_{\frac{1}{4}} (b;c)) : \epsilon \xrightarrow{\frac{3}{4}} (b;c) : \epsilon$$

*For the resumption $(((a \oplus_{\frac{1}{4}} (b;c))\langle a \rightsquigarrow b;d \rangle)\langle c \rightsquigarrow d \rangle) : \epsilon$ we have, applying the shorthand of the $\rightarrow_0$ -notation, for example*

$$\begin{array}{ll}
& (((a \oplus_{\frac{1}{4}} (b;c))\langle a \rightsquigarrow b;d \rangle)\langle c \rightsquigarrow d \rangle) : \epsilon \\
\rightarrow_0 & ((a \oplus_{\frac{1}{4}} (b;c))\langle a \rightsquigarrow b;d \rangle) : \langle c \rightsquigarrow d \rangle \quad \textit{by (Ref)} \\
\rightarrow_0 & (a \oplus_{\frac{1}{4}} (b;c)) : \langle a \rightsquigarrow b;d \rangle\langle c \rightsquigarrow d \rangle \quad \textit{by (Ref)} \\
\rightarrow_0 & (a : \langle a \rightsquigarrow b;d \rangle\langle c \rightsquigarrow d \rangle) \oplus_{\frac{1}{4}} ((b;c) : \langle a \rightsquigarrow b;d \rangle\langle c \rightsquigarrow d \rangle) \quad \textit{by (Op)} \\
\xrightarrow{\frac{1}{4}} & a : \langle a \rightsquigarrow b;d \rangle\langle c \rightsquigarrow d \rangle \quad \textit{by (PChoice 1)}
\end{array}$$

*In turn, considering the resumption $a : \langle a \rightsquigarrow b;d \rangle\langle c \rightsquigarrow d \rangle$ we have*

$$\begin{array}{ll}
& a : \langle a \rightsquigarrow b;d \rangle\langle c \rightsquigarrow d \rangle \\
\rightarrow_0 & (b;d) : \langle c \rightsquigarrow d \rangle \quad \textit{by (Act 2)} \\
\rightarrow_0 & (b : \langle c \rightsquigarrow d \rangle); (d : \langle c \rightsquigarrow d \rangle) \quad \textit{by (Op)} \\
\rightarrow_0 & (b : \epsilon); (d : \langle c \rightsquigarrow d \rangle) \quad \textit{by (Act 3),(Seq)} \\
\xrightarrow{b} & d : \langle c \rightsquigarrow d \rangle \quad \textit{by (Act 1),(Seq)}
\end{array}$$

The complexity function *wgt* is defined for statements and resumptions.

**Definition 5.3.5**

*(a) The function* $wgt\colon \mathcal{L}_{ref} \to \mathbb{N}$ *is given by*

$$\begin{array}{rcl} wgt(a) & = & 1 \\ wgt(s_1; s_2) & = & wgt(s_1) + 1 \\ wgt(s_1 \oplus_\rho s_2) & = & wgt(s_1) + wgt(s_2) + 1 \\ wgt(s_1 \langle a \rightsquigarrow s_2 \rangle) & = & wgt(s_1) + wgt(s_2) + 1 \\ wgt(x) & = & wgt(D(x)) + 1 \end{array}$$

*(b) The function* $wgt\colon Res \to \mathbb{N}$ *is given by*

$$\begin{array}{rcl} wgt(\mathrm{E}) & = & 0 \\ \multicolumn{3}{l}{wgt(s : \langle a_1 \rightsquigarrow s_1 \rangle \langle a_2 \rightsquigarrow s_2 \rangle \cdots \langle a_n \rightsquigarrow s_n \rangle)} \\ & = & wgt(s) + wgt(s_1) + wgt(s_2) + \cdots + wgt(s_n) \\ wgt(r_1; r_2) & = & wgt(r_1) + 1 \\ wgt(r_1 \oplus_\rho r_2) & = & wgt(r_1) + wgt(r_2) + 1 \end{array}$$

Below we will adopt the notation $r \Rightarrow \rho r_1 + (1-\rho) r_2$ in case both $r \xrightarrow{\rho} r_1$ and $r \xrightarrow{1-\rho} r_2$. Here it is not necessarily the case that $r$ equals $r_1 \oplus_\rho r_2$ (cf. examples 5.3.4). Note that this notation expresses a property similar to the property expressed by $s \Rightarrow T$ introduced in chapter 4 (see definition 4.4.7). For ease of notation, the fact that a probabilistic choice in $\mathcal{T}_{\mathrm{pr}}$ always has two options is exploited here.

A first application of *wgt*-induction is the following structural property of the transition system.

**Lemma 5.3.6** *For all* $r \in Res$ *exactly one of the following cases holds:*

- $r = \mathrm{E}$
- $r \xrightarrow{a} r'$ *for some* $a \in Act$, $r' \in Res$
- $r \Rightarrow \rho r' + (1-\rho) r''$ *for some* $r', r'' \in Res$ *and* $\rho \in (0,1)$

**Proof** We only consider the cases for a sequential composition of resumptions. The other cases are straightforward. Suppose $r = r_1; r_2$. Note, $r_1 \neq \mathrm{E}$. As $wgt(r_1) < wgt(r)$ we have by the induction hypothesis that either $r_1 \xrightarrow{a} r_1'$ or $r_1 \Rightarrow \rho r_1' \oplus (1-\rho) r_1''$. Therefore, by (Seq), $r_1; r_2 \xrightarrow{a} r_1'; r_2$ or $r_1; r_2 \Rightarrow \rho(r_1'; r_2) + (1-\rho)(r_1''; r_2)$. From inspection of the transition system we obtain that, for a resumption of the format of $r$ only rule (Seq) of the transition system applies. Hence, if $r \xrightarrow{\lambda} r'$ then there exist $r_1, r_2, r_1'$ such that $r = r_1; r_2$, $r_1 \xrightarrow{\lambda} r_1'$ and $r' \equiv r_1'; r_2$ from which it follows that exactly one of the three cases above holds for a sequential composition of resumptions. □

The lemma above states that a resumption is either a terminating resumption (i.e. $r \equiv \mathrm{E}$), a deterministic resumption (i.e. $\exists a, r' \colon r \xrightarrow{a} r'$), or a probabilistic resumption (i.e. $\exists \rho, r', r'' \colon r \Rightarrow \rho r' + (1-\rho) r''$). This fact is exploited in the definition of the operational model $\mathcal{O}$ for $\mathcal{L}_{pr}$.

**Definition 5.3.7**

*(a) The operational domain $\mathbb{P}_o$ is given by $\mathbb{P}_o = \mathcal{M}(Act^\infty)$.*

*(b) The semantical mapping $\mathcal{O}\colon Res \to \mathbb{P}_o$ is given by*

$$\begin{array}{rcl} \mathcal{O}(\mathrm{E}) & = & \Delta_\epsilon \\ \mathcal{O}(r) & = & \mathcal{O}(r')/a \quad \textit{if}\; r \xrightarrow{a} r' \\ \mathcal{O}(r) & = & \rho\mathcal{O}(r') + (1-\rho)\mathcal{O}(r'') \quad \textit{if}\; r \;\Rightarrow\; \rho r' + (1-\rho) r'' \end{array}$$

*(c) The operational semantics $\mathcal{O}[\![\bullet]\!]\colon Stat \to \mathbb{P}_o$ is given by*

$$\mathcal{O}[\![s]\!] \;=\; \mathcal{O}(s:\epsilon)$$

The meaning of a program in $\mathcal{L}_{pr}$ is given as a probability measure over finite and infinite sequences of actions. This is the same domain as used in chapter 3. The operational model $\mathcal{O}$ is defined based on $\mathcal{T}_{\mathrm{pr}}$ split in the three cases of terminating, deterministic and probabilistic transitions. The operational semantics $\mathcal{O}[\![\bullet]\!]$ gives the process corresponding to the execution starting with the empty refinement sequence.

**Examples 5.3.8**

*(a)*
$$\begin{array}{rl} & \mathcal{O}((a \oplus_{\frac{1}{4}} (b;c)) : \epsilon) \\ = & \frac{1}{4}\mathcal{O}(a:\epsilon) + \frac{3}{4}\mathcal{O}((b;c):\epsilon) \\ & \quad \textit{as}\; (a \oplus_{\frac{1}{4}} (b;c)) : \epsilon \;\Rightarrow\; \frac{1}{4}(a:\epsilon) + \frac{3}{4}((b;c):\epsilon) \\ = & \frac{1}{4}(\mathcal{O}(\mathrm{E})/a) + \frac{3}{4}(\mathcal{O}(c:\epsilon)/b) \quad \textit{as}\; a:\epsilon \xrightarrow{a} \mathrm{E} \;\textit{and}\; (b;c):\epsilon \xrightarrow{b} c:\epsilon \\ = & \frac{1}{4}(\Delta_\epsilon/a) + \frac{3}{4}((\Delta_\epsilon/c)/b) \quad \textit{as}\; \mathcal{O}(\mathrm{E}) = \Delta_\epsilon \;\textit{and}\; c:\epsilon \xrightarrow{c} \mathrm{E} \\ = & \frac{1}{4}\Delta_a + \frac{3}{4}\Delta_{bc} \end{array}$$

*(b)*
$$\begin{array}{rl} & \mathcal{O}((((a \oplus_{\frac{1}{4}} (b;c))\langle a \rightsquigarrow b;d\rangle)\langle c \rightsquigarrow d\rangle) : \epsilon) \\ = & \frac{1}{4}\mathcal{O}(a : \langle a \rightsquigarrow b;d\rangle\langle c \rightsquigarrow d\rangle) + \frac{3}{4}\mathcal{O}((b;c) : \langle a \rightsquigarrow b;d\rangle\langle c \rightsquigarrow d\rangle) \\ & \quad \textit{as}\; (a \oplus_{\frac{1}{4}} (b;c))\langle a \rightsquigarrow b;d\rangle\langle c \rightsquigarrow d\rangle \;\Rightarrow \\ & \qquad \frac{1}{4}(a : \langle a \rightsquigarrow b;d\rangle\langle c \rightsquigarrow d\rangle) + \frac{3}{4}((b;c) : \langle a \rightsquigarrow b;d\rangle\langle c \rightsquigarrow d\rangle) \\ = & \frac{1}{4}(\mathcal{O}(d : \langle c \rightsquigarrow d\rangle)/b) + \frac{3}{4}(\mathcal{O}(c : \langle a \rightsquigarrow b;d\rangle\langle c \rightsquigarrow d\rangle)/b) \\ & \quad \textit{as}\; a : \langle a \rightsquigarrow b;d\rangle\langle c \rightsquigarrow d\rangle \xrightarrow{b} d : \langle c \rightsquigarrow d\rangle \\ & \quad \textit{and}\; (b;c) : \langle a \rightsquigarrow b;d\rangle\langle c \rightsquigarrow d\rangle \xrightarrow{b} c : \langle a \rightsquigarrow b;d\rangle\langle c \rightsquigarrow d\rangle \\ = & \frac{1}{4}((\mathcal{O}(\mathrm{E})/d)/b) + \frac{3}{4}((\mathcal{O}(\mathrm{E})/d)/b) \\ & \quad \textit{as}\; d : \langle c \rightsquigarrow d\rangle \xrightarrow{d} \mathrm{E} \;\textit{and}\; c : \langle a \rightsquigarrow b;d\rangle\langle c \rightsquigarrow d\rangle \xrightarrow{d} \mathrm{E} \\ = & \frac{1}{4}\Delta_{bd} + \frac{3}{4}\Delta_{bd} \\ = & \Delta_{bd} \end{array}$$

(c) *Next we compute $\mathcal{O}(x : \epsilon)$ where $D(x) = a \oplus_{\frac{1}{2}} (a; x)$. For this it turns out to be convenient to exploit the metric foundation of $\mathcal{O}$. On the one hand we have*

$$\mathcal{O}(x : \epsilon) = \tfrac{1}{2}(\mathcal{O}(\epsilon)/a) + \tfrac{1}{2}(\mathcal{O}(x : \epsilon)/a) \tag{3.7}$$

*On the other hand we have for the probability measure $p \in \mathcal{M}(Act^\infty)$ given by $p = \frac{1}{2}\Delta_a + \frac{1}{4}\Delta_{aa} + \frac{1}{8}\Delta_{aaa} + \cdots = \sum_{n=1}^{\infty} \frac{1}{2}^n \Delta_{a^n}$ that $p$ satisfies*

$$p = \tfrac{1}{2}\Delta_a + \tfrac{1}{2}(p/a) \tag{3.8}$$

*We claim that $\mathcal{O}(x : \epsilon) = p$. This can be shown by the following metric argument:*

$$\begin{aligned}
& d(\mathcal{O}(x : \epsilon), p) \\
&= \text{[equations (3.7), (3.8)]}\ d(\tfrac{1}{2}\Delta_a + \tfrac{1}{2}(\mathcal{O}(x : \epsilon)/a), \tfrac{1}{2}\Delta_a + \tfrac{1}{2}(p/a) \\
&= \max\{\, d(\Delta_a, \Delta_a), d(\mathcal{O}(x : \epsilon)/a, p/a) \,\} \\
&= \tfrac{1}{2} d(\mathcal{O}(x : \epsilon), p)
\end{aligned}$$

*We conclude that $d(\mathcal{O}(x : \epsilon), p) = 0$ and hence, since $\mathcal{M}(Act^\infty)$ is a metric space, that $\mathcal{O}(x : \epsilon) = p$.*

Due to its recursive nature, definition 5.3.7 needs further justification. As usual this is done using a contractive higher-order transformation $\Phi\colon Sem \to Sem$.

**Lemma 5.3.9** *Put $Sem = Res \to \mathbb{P}_o$ and let $S$ range over $Sem$. The higher-order transformation $\Phi\colon Sem \to Sem$ is given by*

$$\begin{aligned}
\Phi(S)(\mathrm{E}) &= \Delta_\epsilon \\
\Phi(S)(r) &= S(r')/a \quad \text{if } r \xrightarrow{a} r' \\
\Phi(S)(r) &= \rho\Phi(S)(r') + (1-\rho)\Phi(S)(r'') \quad \text{if } r \Rightarrow \rho r' + (1-\rho) r''
\end{aligned}$$

*Then $\Phi$ has a unique fixed point, and therefore there is exactly one function $\mathcal{O}$ in $Sem$ which satisfies the equations in definition 5.2.8.*

**Proof** Well-definedness of $\Phi$ follows from the fact that $wgt(r'), wgt(r'') < wgt(r)$ if $r \Rightarrow \rho r' + (1-\rho) r''$. In order to show $\frac{1}{2}$-contractiveness of $\Phi$ we check

$$d(\Phi(S_1)(r), \Phi(S_2)(r)) \ \leq\ \tfrac{1}{2} d(S_1, S_2)$$

for arbitrary $S_1, S_2 \in Sem$ by distinguishing three cases. The case for E is clear.

$$\begin{aligned}
[r \xrightarrow{a} r'] \quad & d(\Phi(S_1)(r), \Phi(S_2)(r)) \\
=\ & d(S_1(r')/a, S_2(r')/a) \\
=\ & \tfrac{1}{2} d(S_1(r'), S_2(r')) \\
\leq\ & \text{[definition } d \text{ on } Sem]\ \tfrac{1}{2} d(S_1, S_2)
\end{aligned}$$

$$\begin{aligned}
[r \Rightarrow \rho r' + (1-\rho) r''] \quad & d(\Phi(S_1)(r), \Phi(S_2)(r)) \\
=\ & d(\rho\Phi(S_1)(r') + (1-\rho)\Phi(S_1)(r''), \rho\Phi(S_2)(r') + (1-\rho)\Phi(S_2)(r'')) \\
=\ & \max\{\, d(\Phi(S_1)(r'), \Phi(S_2)(r')), d(\Phi(S_1)(r''), \Phi(S_2)(r'')) \,\} \\
\leq\ & \text{[induction hypothesis on } r', r''] \ \tfrac{1}{2} d(S_1, S_2)
\end{aligned}$$

□

### 5.3.2 Denotational semantics

In this subsection a denotational semantics for $\mathcal{L}_{pr}$ is developed. As in the previous section, the denotational model is constructed using semantical operations ; and $\oplus_\rho$ to find the meaning of programs built with the syntactical operators ; and $\oplus_\rho$. As in the previous section semantical refinements are used to deal with action refinement. A semantical refinement yields, for a given action, a process representing the meaning of the action. The process representing an action is a denotational process in the domain $\mathbb{P}_d$. First the domain $\mathbb{P}_d$ and the operations on this domain are introduced. Then the notion of semantical refinement is given along with the definition of the denotational model $\mathcal{D}$ and denotational semantics $\mathcal{D}[\![\bullet]\!]$.

**Definition 5.3.10** *The domain of denotational processes $\mathbb{P}_d$ is given by $\mathbb{P}_d = \mathcal{M}(\mathbb{Q}_d)$ where $\mathbb{Q}_d = Act^\infty \setminus \{\epsilon\}$.*

The meta variables $p$ and $q$ are used to range over $\mathbb{P}_d$ and $\mathbb{Q}_d$, respectively. Like the operational domain $\mathbb{P}_o$, the domain $\mathbb{P}_d$ is based on sequences of actions. The only difference is that the empty sequence is excluded. This restriction to measures over nonempty words is necessary for proving lemma 5.3.15 which is in turn crucial for the justification of the model $\mathcal{D}$ trough the fixed point characterization. We have that every element in $\mathbb{P}_d$ can be written in one of the following three forms:

- the indicator function $\Delta_a$ for some $a \in Act$
- $p/a$ for some $a \in Act$, $p \in \mathbb{P}$
- $\sum_{i=1}^m \rho_i p_i$ for $\rho_i \in (0,1)$, $p_i = \Delta_{a_i}$ or $p_i = p'_i/a_i$ with $a_i \in Act$, $p'_i \in \mathbb{P}$ and $\sum_{i=1}^m \rho_i = 1$

The fact that a finite combination in the third clause for the representation of elements in $\mathbb{P}_d$ suffices, follows from the observation that in $\mathbb{P}_d$ only measures of compact support are considered.

Next we provide semantical counterparts of the syntactical construction of the sequential and probabilistic compositions ; and $\oplus_\rho$.

**Definition 5.3.11**

*(a) The semantical operation $;\colon \mathbb{P}_d \times \mathbb{P}_d \to \mathbb{P}_d$ is given by*

$$\begin{array}{rcl} \Delta_a; p & = & p/a \\ (p/a); p' & = & (p; p')/a \\ (\sum_{i=1}^m \rho_i p_i); p' & = & \sum_{i=1}^m \rho_i (p_i; p') \end{array}$$

*(b) The semantical operation $\oplus_\rho\colon \mathbb{P}_d \times \mathbb{P}_d \to \mathbb{P}_d$, for $\rho \in (0,1)$, is given by*

$$p \oplus_\rho p' \quad = \quad (\rho p) + ((1-\rho)p')$$

Note that the definition of ; in the first clause of this definition uses the characterization of processes introduced in 5.3.2 above. The definition of $\oplus_\rho$ in the second clause simply uses scalar multiplication and addition of functions. (Recall that a measure $p$ is a function to $[0,1]$.)

The recursive definition of ; is justified by showing that ; is the unique fixed point of a higher-order transformation $\Omega_;$.

**Lemma 5.3.12** *Put* $Op = \mathbb{P}_d \times \mathbb{P}_d \to \mathbb{P}_d$. *Define the higher-order transformation* $\Omega_; \colon Op \to Op$ *by*

$$\begin{array}{rcl} \Omega_;(\phi)(\Delta_a, p') & = & p'/a \\ \Omega_;(\phi)(\bar{p}/a, p') & = & \phi(\bar{p}, p')/a \\ \Omega_;(\phi)(\sum_{i=1}^m \rho_i p_i, p') & = & \sum_{i=1}^m \rho_i \Omega_;(\phi)(p_i, p') \end{array}$$

*Then* $\Omega_;$ *is well-defined and* $\frac{1}{2}$*-contractive and thus has a unique fixed point. There is, therefore, exactly one function* ; $\in Op$ *which satisfies the equations in definition 5.3.11.*

**Proof** Well-definedness of $\Omega_;$ is clear. In the third clause each $p_i$ is either of the format $p_i = \Delta_{a_i}$ or $p_i = \bar{p}_i/a_i$, which are covered by the other clauses.

To show that $\Omega_;$ is contractive we prove, for arbitrary $\phi_1, \phi_2 \in Op$, $p, p' \in \mathbb{P}_d$,

$$d(\Omega_;(\phi_1)(p,p'), \Omega_;(\phi_2)(p,p')) \leq \tfrac{1}{2} d(\phi_1, \phi_2)$$

from which, by the definition of $d$ on $Op$, it follows that $d(\Omega_;(\phi_1), \Omega_;(\phi_2)) \leq \frac{1}{2} d(\phi_1, \phi_2)$. We distinguish three cases:

$[a]$ $\quad d(\Omega_;(\phi_1)(a,p'), \Omega_;(\phi_2)(a,p')) = d(p'/a, p'/a) = 0$

$[\bar{p}/a]$ $\quad d(\Omega_;(\phi_1)(\bar{p}/a), \Omega_;(\phi_2)(\bar{p}/a))$

$$\begin{array}{rl} = & d(\phi_1(\bar{p},p')/a, \phi_2(\bar{p},p')/a) \\ = & \frac{1}{2} d(\phi_1(\bar{p},p'), \phi_2(\bar{p},p')) \\ \leq & \frac{1}{2} d(\phi_1, \phi_2) \end{array}$$

$[\sum_{i=1}^m \rho_i p_i]$ $\quad d(\Omega_;(\phi_1)(\sum_{i=1}^m \rho_i p_i, p'), \Omega_;(\phi_2)(\sum_{i=1}^m \rho_i p_i, p'))$

$$\begin{array}{rl} = & d(\sum_{i=1}^m \rho_i \Omega_;(\phi_1)(p_i,p'), \sum_{i=1}^m \rho_i \Omega_;(\phi_2)(p_i,p')) \\ \leq & \max\{\, d(\Omega_;(\phi_1)(p_i,p'), \Omega_;(\phi_2)(p_i,p') \mid i \in \{\,1, \ldots n\,\}\,\} \\ \leq & \text{[previous cases]} \;\; \frac{1}{2} d(\phi_1, \phi_2) \end{array}$$

□

A semantical refinement specifies how an action should be refined by giving the meaning of an action as a denotational process. The denotational model $\mathcal{D}$ uses a semantical refinement as an extra argument. The denotational semantics $\mathcal{D}[\![\bullet]\!]$ does not use this additional argument but instead starts with the 'identity refinement'. The identity refinement yields, for a given action $a$, the processes that executes only the action $a$ with a probability of 1.

**Definition 5.3.13** *Let the collection SemRef of semantical refinements, ranged over by $\eta$, be given by SemRef $=$ Act $\to \mathbb{P}_d$. In particular we distinguish $\eta_{id} \in$ SemRef such that $\eta_{id}(a) = \Delta_a$ for all $a \in$ Act. The semantical mapping $\mathcal{D}\colon \mathcal{L}_{pr} \to$ SemRef $\to \mathbb{P}_d$ is given by*

$$\begin{array}{rcl}
\mathcal{D}(a)(\eta) & = & \eta(a) \\
\mathcal{D}(s;s')(\eta) & = & \mathcal{D}(s)(\eta);\mathcal{D}(s')(\eta) \\
\mathcal{D}(s \oplus_\rho s')(\eta) & = & \mathcal{D}(s)(\eta) \oplus_\rho \mathcal{D}(s')(\eta) \\
\mathcal{D}(s\langle a \rightsquigarrow s'\rangle)(\eta) & = & \mathcal{D}(s)(\eta[\mathcal{D}(s')(\eta)/a]) \\
\mathcal{D}(x)(\eta) & = & \mathcal{D}(D(x))(\eta)
\end{array}$$

*The denotational semantics $\mathcal{D}[\![\bullet]\!]\colon \mathcal{L}_{pr} \to \mathbb{P}_d$ is given by $\mathcal{D}[\![s]\!] = \mathcal{D}(s)(\eta_{id})$.*

Note that definition 5.3.13 does not go by structural induction (cf. the clause for $x$) nor does it go by *wgt*-induction (cf. the clause for $s;s'$). We first provide some examples of $\mathcal{D}$ before delving into the well-definedness of $\mathcal{D}$.

**Examples 5.3.14**

*(a)*
$$\begin{array}{l}
\mathcal{D}(a \oplus_{\frac{1}{4}} (b;c))(\eta_{id}) \\
\quad = \mathcal{D}(a)(\eta_{id}) \oplus_{\frac{1}{4}} \mathcal{D}(b;c)(\eta_{id}) \\
\quad = \frac{1}{4}\eta_{id}(a) + \frac{3}{4}(\mathcal{D}(b)(\eta_{id});\mathcal{D}(c)(\eta_{id})) \\
\quad = \frac{1}{4}\Delta_a + \frac{3}{4}(\eta_{id}(b);\eta_{id}(c)) \\
\quad = \frac{1}{4}\Delta_a + \frac{3}{4}(\Delta_b;\Delta_c) \\
\quad = \frac{1}{4}\Delta_a + \frac{3}{4}(\Delta_{bc})
\end{array}$$

*(b)*
$$\begin{array}{l}
\mathcal{D}(((a \oplus_{\frac{1}{4}} (b;c))\langle a \rightsquigarrow b;d\rangle)\langle c \rightsquigarrow d\rangle)(\eta_{id}) \\
\quad = \mathcal{D}((a \oplus_{\frac{1}{4}} (b;c))\langle a \rightsquigarrow b;d\rangle)(\eta_{id}[\mathcal{D}(d)(\eta_{id})/c]) \\
\quad = \mathcal{D}((a \oplus_{\frac{1}{4}} (b;c))\langle a \rightsquigarrow b;d\rangle)(\eta_{id}[\Delta_d/c]) \\
\quad = \mathcal{D}((a \oplus_{\frac{1}{4}} (b;c))(\eta_{id}[\Delta_d/c][\mathcal{D}(b;d)(\eta_{id}[\Delta_d/c])/a])) \\
\quad = \ldots \\
\quad = \mathcal{D}(a \oplus_{\frac{1}{4}} (b;c))(\eta_{id}[\Delta_d/c, \Delta_{bd}/a]) \\
\quad = \frac{1}{4}\mathcal{D}(a)(\eta_{id}[\Delta_d/c, \Delta_{bd}/a]) + \\
\qquad \frac{3}{4}\mathcal{D}(b;c)(\eta_{id}[\Delta_d/c, \Delta_{bd}/a]) \\
\quad = \ldots \\
\quad = \frac{1}{4}\Delta_{bd} + \frac{3}{4}(\Delta_b;\Delta_d) \\
\quad = \frac{1}{4}\Delta_{bd} + \frac{3}{4}\Delta_{bd} \\
\quad = \Delta_{bd}
\end{array}$$

*(c) Suppose $D(x) = a \oplus_{\frac{1}{2}} (a;x)$. On the one hand we have*

$$\mathcal{D}(x)(\eta_{id})$$

$$\begin{aligned}
&= \mathcal{D}(a \oplus_{\frac{1}{2}} (a;x))(\eta_{id}) \\
&= \tfrac{1}{2}\mathcal{D}(a)(\eta) + \tfrac{1}{2}(\mathcal{D}(a)(\eta_{id}); \mathcal{D}(x)(\eta_{id})) \\
&= \tfrac{1}{2}\Delta_a + \tfrac{1}{2}(\mathcal{D}(x)(\eta_{id})/a)
\end{aligned}$$

*On the other hand we have for $p = \sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^n \Delta_{a^n}$ that $p = \frac{1}{2}\Delta_a + \frac{1}{2}(p/a)$. Hence, as for the same example in the context of subsection 5.3.1, we have*

$$\begin{aligned}
& d(\mathcal{D}(x)(\eta_{id}), p) \\
&= d(\tfrac{1}{2}\Delta_a + \tfrac{1}{2}(\mathcal{D}(x)(\eta_{id})/a), \tfrac{1}{2}\Delta_a + \tfrac{1}{2}(p/a)) \\
&\leq \max\{\, d(\Delta_a, \Delta_a), d(\mathcal{D}(x)(\eta_{id})/a, p/a) \,\} \\
&= \tfrac{1}{2}d(\mathcal{D}(x)(\eta_{id}), p)
\end{aligned}$$

*Therefore, $d(\mathcal{D}(x)(\eta_{id}), p) = 0$ and $\mathcal{D}(x)(\eta_{id}) = \sum_{i=1}^{\infty} \left(\frac{1}{2}\right)^n \Delta_{a^n}$.*

We first establish some nonexpansiveness/contractivity properties and distributivity results of the semantical operations that are needed for the justification of the definition of $\mathcal{D}$ in the sequel.

**Lemma 5.3.15**

(a) *The semantical operation $\oplus_\rho$ is nonexpansive for all $\rho \in (0,1)$.*

(b) *The semantical operation ; is nonexpansive in its first argument and $\frac{1}{2}$-contractive in its second argument.*

**Proof** Nonexpansiveness of the operation $\oplus_\rho$ is straightforward. For the sequential composition we use the same approach as in lemma 3.4.11. Define the subset $Op_0 \subseteq Op$ by $\phi \in Op_0 \iff d(\phi(p,p''), \phi(p',p'')) \leq d(p,p') \wedge d(\phi(p,p'), \phi(p,p'')) \leq \frac{1}{2}d(p',p'')$. Note that $Op_0 \subseteq Op$ is a nonempty and closed subset. We check that $\phi \in Op_0$ implies $\Omega_;(\phi) \in Op_0$. This implies that the fixed point ; of $\Omega_;$ must also be in $Op_0$.

Pick any $\phi \in Op_0$ and choose arbitrary $p, p', p'' \in \mathbb{P}_d$. We verify the inequality $d(\Omega_;(\phi)(p,p''), \Omega_;(\phi)(p',p'')) \leq d(p,p')$. Without loss of generality (leaving the details to the reader) we can assume $d(p,p') < \frac{1}{2}$. We distinguish three cases.

$[p = \Delta_a, p' = \Delta_a]$ Clear.

$[p = \bar{p}/a, p' = \bar{p}'/a]$ We have that $d(p,p') = \frac{1}{2}d(\bar{p},\bar{p}')$.

$$\begin{aligned}
& d(\Omega_;(\phi)(p,p''), \Omega_;(\phi)(p',p'')) \\
=\ & d(\phi(\bar{p},p')/a, \phi(\bar{p}',p'')/a) \\
=\ & \tfrac{1}{2}d(\phi(\bar{p},p''), \phi(\bar{p}',p'')) \\
\leq\ & [\text{property } \phi]\ \tfrac{1}{2}d(\bar{p},\bar{p}') \\
=\ & d(p,p')
\end{aligned}$$

$[p = \sum_{i=1}^{m} \rho_i p_i, p' = \sum_{i=1}^{m} \rho_i p'_i]$ Note that, for $i$, $1 \leq i \leq m$, $p_i, p'_i$ are either both of the format $p_i = \Delta_{a_i}$, $p'_i = \Delta_{a_i}$ or both of the format $p_i = \bar{p}_i/a_i, p'_i = \bar{p}'_i/a_i$.

$$\begin{array}{ll} & d(\Omega_;(\phi)(p,p''),\Omega_;(\phi)(p',p'')) \\ = & d(\sum_{i=1}^m \rho_i\Omega_;(\phi)(p_i,p''),\sum_{i=1}^m \rho_i\Omega_;(\phi)(p'_i,p'')) \\ = & \max\{\, d(\Omega_;(\phi)(p_i,p''),\Omega_;(\phi)(p'_i,p'')) \mid i=1,\dots m\,\} \\ \leq & \text{[earlier cases]} \;\; d(p,p') \end{array}$$

We conclude that $d(\Omega_;(\phi)(p,p''),\Omega_;(\phi)(p',p'')) \leq d(p,p')$. Similarly, one can prove that $d(\Omega_;(\phi)(p,p'),\Omega_;(\phi)(p,p'')) \leq \frac{1}{2}d(p',p'')$. □

The results gathered so far are sufficient to justify the denotational model $\mathcal{D}$ by showing that $\mathcal{D}$ it is the unique fixed point of a contractive higher order transformation $\Psi$. This justification is omitted, see e.g. lemmas 3.4.13 and 5.2.16 for similar results. Next we establish that the operation probabilistic composition distributes over the operation sequential composition. This fact will be used in the next subsection to show the correctness of the denotational model.

**Lemma 5.3.16** *For all $p,p',p'' \in \mathbb{P}_d$ it holds that $(p \oplus_\rho p');p'' = (p;p'') \oplus_\rho (p';p'')$.*

**Proof** Suppose $p = \sum_{i=1}^m \rho_i p_i$, $p' = \sum_{j=1}^n \sigma_j p'_j$. Then, for any $\rho \in (0,1)$,

$$\begin{array}{ll} & (p \oplus_\rho p');p'' \\ = & ((\sum_{i=1}^m(\rho\rho_i)p_i) + (\sum_{j=1}^n((1-\rho)\sigma_j)p'_j));p'' \\ = & (\sum_{i=1}^m(\rho\rho_i)(p_i;p'')) + (((1-\rho)\sigma_j)(p'_j;p'')) \\ = & \rho(\sum_{i=1}^m \rho_i(p_i;p'')) + (1-\rho)(\sum_{j=1}^n \sigma_j(p'_j;p'')) \\ = & \big(\rho(p;p'')\big) + \big((1-\rho)(p';p'')\big) \\ = & (p;p'') \oplus_\rho (p';p'') \end{array}$$

□

## 5.3.3 Correctness

In this subsection we will establish the correctness of the denotational semantics $\mathcal{D}$ for $\mathcal{L}_{pr}$ with respect to its operational model $\mathcal{O}[\![\bullet]\!]$. As the functionality of $\mathcal{D}$ differs form that of $\mathcal{O}$, viz. $\mathcal{D}: \mathcal{L}_{pr} \to \mathit{SemRef} \to \mathcal{M}(\mathit{Act}^\infty \setminus \{\epsilon\})$ versus $\mathcal{O}: \mathit{Res} \to \mathcal{M}(\mathit{Act}^\infty)$ we will use an intermediate function $\mathcal{E}$ that is based on $\mathcal{D}$ for its definition, but agrees with $\mathcal{O}$ for its functionality. The main lemma of this section, lemma 5.3.18, exploits Banach's fixed point theorem to show that $\mathcal{O}$ and $\mathcal{E}$ in fact coincide.

As in the previous section a mechanism to combine syntactical refinement sequences $R \in \mathit{RefSeq}$ and semantical refinements $\eta \in \mathit{SemRef}$ is given.

**Lemma 5.3.17** *Let the function $\bullet \triangleright \bullet: \mathit{RefSeq} \times \mathit{SemRef} \to \mathit{SemRef}$ be inductively given by*

$$\begin{array}{rcl} \epsilon \triangleright \eta & = & \eta \\ (R\,\langle a \rightsquigarrow s\rangle) \triangleright \eta & = & R \triangleright \eta[\mathcal{D}(s)(\eta)/a] \end{array}$$

*Then it holds that*

*(a)* $\mathcal{D}(s\langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle)(\eta)$
$= \mathcal{D}(s)(\langle a_1 \rightsquigarrow s_1\rangle\langle a_2 \rightsquigarrow s_2\rangle \cdots \langle a_n \rightsquigarrow s_n\rangle \triangleright \eta)$

*(b)* $\mathcal{D}(s_1\langle a \rightsquigarrow s_2\rangle)(R \triangleright \eta) = \mathcal{D}(s_1)((\langle a \rightsquigarrow s_2\rangle\, R) \triangleright \eta)$

Next we present the intermediate semantical mapping $\mathcal{E}$ and prove that $\mathcal{E}$ is a fixed point of the higher order transformation $\Phi$ of lemma 5.3.9. Since, by Banach's fixed point theorem, $\Phi$ has exactly one fixed point —which is $\mathcal{O}$— it follows that $\mathcal{O} = \mathcal{E}$. The definition of $\mathcal{E}$ makes both use of the denotational semantics $\mathcal{D}$, for resumptions of the format $s : R$, and of the semantical operations defined on the domain $\mathbb{P}_d$, for resumptions of the format $r_1\, op\, r_2$ where $op$ is either a sequential or a probabilistic operator.

**Lemma 5.3.18** *Let the mapping* $\mathcal{E}\colon Res \to \mathcal{M}(Act^\infty)$ *be given as follows:*

$$\begin{array}{rcl}
\mathcal{E}(\mathrm{E}) & = & \Delta_\epsilon \\
\mathcal{E}(s : R) & = & \mathcal{D}(s)(R \triangleright \eta_{id}) \\
\mathcal{E}(r_1\, op\, r_2) & = & \mathcal{E}(r_1)\, op\, \mathcal{E}(r_2) \;\; \textit{for } op \in \{;, \oplus_\rho\}
\end{array}$$

*Then it holds that* $\Phi(\mathcal{E}) = \mathcal{E}$.

**Proof** It is straightforwardly checked that $\mathcal{E}(r) \in \mathbb{P}_d$ for $r \neq \mathrm{E}$, hence $\mathcal{E}$ is well-defined. We prove that $\Phi(\mathcal{E}) = \mathcal{E}$ by weight-induction for resumptions. We only treat the cases for sequential composition and probabilistic choice. The other cases are the same as in lemma 5.2.19.

$[r_1; r_2]$ Suppose $r_1 \xrightarrow{a} r_1'$. Then $r_1; r_2 \xrightarrow{a} r_1'; r_2$.

$$\begin{array}{rl}
& \Phi(\mathcal{E})(r_1; r_2) \\
= & [\text{transition rule (Seq)}]\;\; \mathcal{E}(r_1'; r_2)/a \\
= & \big(\mathcal{E}(r_1'); \mathcal{E}(r_2)\big)/a \\
= & [\text{definition } ;]\;\; (\mathcal{E}(r_1')/a); \mathcal{E}(r_2) \\
= & [\text{definition } \Phi]\;\; \Phi(\mathcal{E})(r_1); \mathcal{E}(r_2) \\
= & [\text{induction hypothesis for } r_1]\;\; \mathcal{E}(r_1); \mathcal{E}(r_2) \\
= & [\text{definition } \mathcal{E}]\;\; \mathcal{E}(r_1; r_2)
\end{array}$$

Suppose $r_1 \Rightarrow \rho r_1' + (1-\rho) r_1''$. Then $r_1; r_2 \Rightarrow \rho(r_1'; r_2) + (1-\rho)(r_1''; r_2)$.

$$\begin{array}{rl}
& \Phi(\mathcal{E})(r_1; r_2) \\
= & [\text{definition } \Phi]\;\; \rho\Phi(\mathcal{E})(r_1'; r_2) + (1-\rho)\Phi(\mathcal{E})(r_1''; r_2) \\
= & [\text{induction hypothesis}]\;\; \rho\mathcal{E}(r_1'; r_2) + (1-\rho)\mathcal{E}(r_1''; r_2) \\
= & [\text{definition of } \mathcal{E}]\;\; \rho\big(\mathcal{E}((r_1'); \mathcal{E}(r_2)\big) + (1-\rho)\big(\mathcal{E}(r_1''); \mathcal{E}(r_2)\big) \\
= & [\text{lemma 5.3.16}]\;\; \big(\rho\mathcal{E}(r_1') + (1-\rho)\mathcal{E}(r_1'')\big); \mathcal{E}(r_2) \\
= & [\text{definition } \Phi]\;\; \Phi(\mathcal{E})(r_1); \mathcal{E}(r_2) \\
= & [\text{induction hypothesis for } r_1]\;\; \mathcal{E}(r_1); \mathcal{E}(r_2)
\end{array}$$

$$\begin{array}{lll} & = & [\text{definition } \mathcal{E}] \;\; \mathcal{E}(r_1; r_2) \\ [r_1 \oplus_\rho r_2] & & \Phi(\mathcal{E})(r_1 \oplus_\rho r_2) \\ & = & [\text{transition rules (PChoice)}] \;\; \rho\mathcal{E}(r_1) + (1-\rho)\mathcal{E}(r_2) \\ & = & \mathcal{E}(r_1) \oplus_\rho \mathcal{E}(r_2) \\ & = & \mathcal{E}(r_1 \oplus_\rho r_2) \end{array}$$ □

From the lemma we immediately obtain the correctness result for the denotational semantics $\mathcal{D}$ for $\mathcal{L}_{pr}$.

**Theorem 5.3.19** *For all* $s \in \mathcal{L}_{pr}$,$\mathcal{O}[\![s]\!] = \mathcal{D}(s)(\eta_{id})$ *on* $\mathcal{L}_{pr}$.
**Proof** We have $\mathcal{O}[\![s]\!] = \mathcal{O}(s : \epsilon) = \mathcal{E}(s : \epsilon) = \mathcal{D}(s)(\eta_{id})$ for any $s \in \mathcal{L}_{pr}$. □

## 5.3.4 Full abstractness

In this subsection we establish for the semantical mapping $\mathcal{D}\colon \mathcal{L}_{pr} \to \mathit{SemRef} \to \mathbb{P}_d$ full abstraction with respect to $\mathcal{O}[\![\bullet]\!]$, i.e. we will show, for any $s_1, s_2 \in \mathcal{L}_{pr}$, that

$$\mathcal{D}(s_1) = \mathcal{D}(s_2) \iff \mathcal{O}[\![C[s_1]]\!] = \mathcal{O}[\![C[s_2]]\!] \text{for all contexts } C[\bullet]$$

The approach is the same as in subsection 5.2.3. The discussion here is restricted to the general outline of the technique and to the particularities for the probabilistic setting of $\mathcal{L}_{pr}$. The route to the main technical lemma of this section, namely lemma 5.3.26, passes the following ideas:

- If a statement $s$ has denotations with respect to the semantical refinements $\eta_1, \eta_2$ of distance $2^{-(n+1)}$ then the semantical refinements $\eta_1, \eta_2$ have a distance of at least $2^{-(n+1)}$ on the collection of the first $n$ actions occurring in any run of $s$.
- A finitary semantical refinement, i.e. a semantical refinement which delivers $\Delta_a$ for all actions $a$ except for finitely many actions for which a measure is returned which assigns all probability to a finite set of finite sequences, can be represented by a syntactical refinement sequence.

We start with a definition for $act_n(s)$ which indicates the first $n$ actions that may occur in a run of a statement $s$.

**Definition 5.3.20** *For* $n \in \mathbb{N}$ *and* $s \in \mathit{Stat}$, *the subset* $act_n(s)$ *of* $\mathit{Act}$ *is inductively given as in definition 5.2.21 except for*

$$act_{n+1}(s_1 \oplus_\rho s_2) \quad = \quad act_{n+1}(s_1) \cup act_{n+1}(s_2)$$

*The set* $act(s) \subseteq \mathit{Act}$ *is given by* $act(s) = \bigcup_n act_n(s)$.

The next lemma handles the first idea for the full abstractness theorem below.

**Lemma 5.3.21** *Let* $n \in \mathbb{N}$ *and* $s \in \mathit{Stat}$. *If* $d(\eta_1, \eta_2) \leq 2^{-n}$ *on* $act_n(s)$ *then it holds that* $d(\mathcal{D}(s)(\eta_1), \mathcal{D}(s)(\eta_2)) \leq 2^{-n}$, *for all* $\eta_1, \eta_2 \in \mathit{SemRef}$.

**Proof** Induction on $n$ and subinduction on $wgt(s)$. In order to illustrate a typical argument, we exhibit the subcase for $s_1 \oplus_\rho s_2$ in the case for $n+1$: If $d(\eta_1, \eta_2) \leq 2^{-(n+1)}$ on $act_{n+1}(s_1 \oplus_\rho s_2)$, then, since $act_{n+1}(s_1 \oplus_\rho s_2) = act_{n+1}(s_1) \cup act_{n+1}(s_2)$, we have that $d(\eta_1, \eta_2) \leq 2^{-(n+1)}$ on $act_{n+1}(s_1, s_2)$. Thus, it follows that $d(\mathcal{D}(s_i)(\eta_1), \mathcal{D}(s_i)(\eta_2)) \leq 2^{-(n+1)}$ for $i = 1, 2$, by the induction hypothesis for $s_1$ and $s_2$. By compositionality of $\mathcal{D}$ and nonexpansiveness of $\oplus_\rho$, we obtain

$$\begin{aligned}
& d(\mathcal{D}(s_1 \oplus_\rho s_2)(\eta_1), \mathcal{D}(s_1 \oplus_\rho s_2)(\eta_2)) \\
& \quad \leq \max\{\, d(\mathcal{D}(s_1)(\eta_1), \mathcal{D}(s_1)(\eta_2)), d(\mathcal{D}(s_2)(\eta_1), \mathcal{D}(s_2)(\eta_2)) \,\} \\
& \quad \leq 2^{-(n+1)}
\end{aligned}$$

□

The following lemma prepares for the second idea for theorem 5.3.27 as reflected by lemma 5.3.25 by showing that only the refinement of actions in $act(s)$ matters in finding the denotational meaning of a program $s$.

**Lemma 5.3.22** *For all $s \in Stat$ and $\eta_1, \eta_2 \in SemRef$ such that $\eta_1 = \eta_2$ on $act(s)$ we have $\mathcal{D}(s)(\eta_1) = \mathcal{D}(s)(\eta_2)$.*

**Proof** Define $\varepsilon = \sup\{\, d(\mathcal{D}(s)(\eta_1), \mathcal{D}(s)(\eta_2)) \mid \eta_1 = \eta_2 \text{ on } act(s) \,\}$. One shows by induction on $wgt(s)$ that $d(\mathcal{D}(s)(\eta_1), \mathcal{D}(s)(\eta_2)) \leq \frac{1}{2}\varepsilon$. From this it follows that $\varepsilon = 0$. Hence $d(\mathcal{D}(s)(\eta_1), \mathcal{D}(s)(\eta_2)) = 0$ and $\mathcal{D}(s)(\eta_1) = \mathcal{D}(s)(\eta_2)$ for $s, \eta_1, \eta_2$ such that $\eta_1 = \eta_2$ on $act(s)$. By way of example we provide the case for $s_1; s_2$:

$$\begin{aligned}
& d(\mathcal{D}(s_1; s_2)(\eta_1), \mathcal{D}(s_1; s_2)(\eta_2)) \\
& \quad \leq [\,;\ \text{nonexpansive/contractive}] \\
& \qquad \max\{\, d(\mathcal{D}(s_1)(\eta_1), \mathcal{D}(s_1)(\eta_2)), \tfrac{1}{2} d(\mathcal{D}(s_2)(\eta_1), \mathcal{D}(s_2)(\eta_2)) \,\} \\
& \quad = [\text{ind. hyp. for } s_1,\ act(s_2) \subseteq act(s_1; s_2),\ \text{def. } \varepsilon]\ \tfrac{1}{2}\varepsilon
\end{aligned}$$

□

The properties given in lemma 5.2.24 and corollary 5.2.25, dealing with technicalities having to do with iterated refinement versus simultaneous substitution, also hold in this setting.

**Lemma 5.3.23**

*(a) Suppose $a_1, \ldots, a_n \in Act$ are pairwise distinct, and $s_1, \ldots, s_n \in Stat$ are such that $act(s_i) \cap \{\, a_1, \ldots, a_n \,\} = \emptyset$ for $1 \leq i \leq n$. Then it holds that*

$$\mathcal{D}(s\langle a_i \rightsquigarrow s_i \rangle_{i=1}^n)(\eta) \ = \ \mathcal{D}(s)(\eta[p_i/a_i]_{i=1}^n)$$

*where $p_i = \mathcal{D}(s_i)(\eta)$ for $1 \leq i \leq n$.*

*(b) Suppose $\bar{a}_i, a'_j$ $(1 \leq i, j \leq n)$ are pairwise distinct. Then it holds that*

$$\mathcal{D}(s\langle \bar{a}_i \rightsquigarrow a'_i \rangle_{i=1}^n)(\eta) = \mathcal{D}(s)(\eta[\eta(a'_i)/\bar{a}_i]_{i=1}^n)$$

We have now arrived at the second idea on our way to the full abstractness of $\mathcal{D}$. First we need a definition.

**Definition 5.3.24** *A semantical refinement $\eta \in SemRef$ is called* finitary *if the following conditions are fulfilled:*

- *for all $a \in Act$ it holds that $\eta(a)$ is a finitary probability measure (the support of the measure is finite) over $Act^+$*
- *$\eta(a) \neq \Delta_a$ for finitely many $a \in Act$*

The crux underlying the proof of lemma 5.3.25 is that any finitary probability measure over finite words can be syntactically represented. For example, the measure $\frac{1}{2}\Delta_a + \frac{1}{3}\Delta_{bc} + \frac{1}{6}\Delta_{ade}$ can be represented by the statement $a \oplus_{\frac{1}{2}} \big((b;c) \oplus_{2/3} (a;d;e)\big)$, in the sense that $\mathcal{D}(a \oplus_{\frac{1}{2}} \big((b;c) \oplus_{2/3} (a;d;e)\big))(\eta_{id}) = \frac{1}{2}\Delta_a + \frac{1}{3}\Delta_{bc} + \frac{1}{6}\Delta_{ade}$.

**Lemma 5.3.25** *Let $\eta \in SemRef$ be a finitary semantical refinement and $A \subseteq Act$ a finite set of actions. Then there exists, for all $n \in \mathbb{N}$, a refinement sequence $\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k$ such that*

$$d(\mathcal{D}(s)(\eta), \mathcal{D}(s\langle a_i \rightsquigarrow s_i \rangle_{i=1}^k)(\eta_{id})) \leq \tfrac{1}{2}^n$$

*for all $s \in Stat$ with $act_n(s) \subseteq A$.*

**Proof** If $p$ is a finitary probability measure over $Act^+$, say $p = \sum_{i=1}^m \rho_i q_i$ for $m \geq 1$, $\rho_1, \ldots, \rho_m > 0$ such that $\rho_1 + \cdots + \rho_m = 1$, $q_1, \ldots, q_m \in Act^+$, the statement $stat(p)$ is given by $stat(p) = \sum_{i=1}^m \rho_i stat'(q_i)$ where $stat'(\Delta_a) = a$, $stat'(q/a) = a; stat'(q)$. (Note that $\mathcal{L}_{pr}$ provides binary probabilistic composition only. Further details on transforming arbitrary finite probabilistic composition into repeated binary probabilistic are omitted here.) It is straightforwardly checked that $\mathcal{D}(stat(p))(\eta_{id}) = p$ by simultaneous induction on $m$ and the lengths of the $q_i$'s.

Suppose $\bar{a}_1, \ldots, \bar{a}_\ell$ are all actions $a$ such that $\eta(s) \neq a$. We are done if we show

$$d(\mathcal{D}(s)(\eta), \mathcal{D}(s\langle \bar{a}_i \rightsquigarrow stat(\eta(\bar{a}_i)) \rangle_{i=1}^\ell)(\eta_{id})) \leq \tfrac{1}{2}^n$$

(Some caution has to be taken though, in order to prevent clashes of actions. See 5.2.27.) As $\eta$ is finitary, only finitely many actions are involved. The lemma then follows using lemma 5.3.21 and lemma 5.3.23. □

We have now arrived at the main technical result of this section.

**Lemma 5.3.26** *If $s', s'' \in Stat$ satisfy $\mathcal{D}(s') \neq \mathcal{D}(s'')$ then $\mathcal{O}[\![C[s']]\!] \neq \mathcal{O}[\![C[s'']]\!]$ for some context $C[\bullet]$.*

**Proof** Suppose $\mathcal{D}(s') \neq \mathcal{D}(s'')$. We can choose $\eta \in SemRef$ and $n \in \mathbb{N}$ such that $d(\mathcal{D}(s')(\eta), \mathcal{D}(s'')(\eta)) = 2^{-n}$. Let $\eta'$ be finitary such that $d(\eta, \eta') \leq 2^{-(n+1)}$ on $act_n(s') \cup act_n(s'')$. By lemma 5.3.21 and ultrametricity we then obtain

$$d(\mathcal{D}(s')(\eta'), \mathcal{D}(s'')(\eta') = 2^{-n}$$

Pick, applying lemma 5.3.25 and again ultrametricity, a refinement sequence $\langle a_i \rightsquigarrow s_i \rangle_{i=1}^{k}$ such that

$$d(\mathcal{D}(s'\langle a_i \rightsquigarrow s_i \rangle_{i=1}^{k})(\eta_{id}), \mathcal{D}(s''\langle a_i \rightsquigarrow s_i \rangle_{i=1}^{k})(\eta_{id})) = 2^{-n}$$

hence $\mathcal{D}[\![s'\langle a_i \rightsquigarrow s_i \rangle_{i=1}^{k}]\!] \neq \mathcal{D}[\![s''\langle a_i \rightsquigarrow s_i \rangle_{i=1}^{k}]\!]$. Define the context $C[\bullet] = (\bullet)\langle a_i \rightsquigarrow s_i \rangle_{i=1}^{k}$. Then, by theorem 5.3.19, it follows that $\mathcal{O}[\![C[s']]\!] \neq \mathcal{O}[\![C[s'']]\!]$. □

Lemma 5.3.26 and the correctness result obtained in theorem 5.3.19 have paved the way for a proof of the full abstractness of the semantical mapping $\mathcal{D}$ with respect to the operational semantics $\mathcal{O}[\![\bullet]\!]$ of $\mathcal{L}_{pr}$.

**Theorem 5.3.27** *$\mathcal{D}\colon \mathcal{L}_{ref} \to SemRef \to \mathbb{P}_d$ is fully abstract with respect to $\mathcal{O}[\![\bullet]\!]$.*

**Proof** Suppose that $\mathcal{D}(s_1) \neq \mathcal{D}(s_2)$ for two statements $s_1, s_2 \in Stat$. By lemma 5.3.26 there exists a context $C[\bullet]$ such that $\mathcal{O}[\![C[s_1]]\!] \neq \mathcal{O}[\![C[s_2]]\!]$. Suppose $\mathcal{D}(s_1) = \mathcal{D}(s_2)$ for two statements $s_1, s_2 \in Stat$. By definition of $\mathcal{D}$ we have, for any context $C[\bullet]$ and semantical refinement $\eta$, that $\mathcal{D}(C[s_1])(\eta) = \mathcal{D}(C[s_2])(\eta)$. In particular $\mathcal{D}[\![C[s_1]]\!] = \mathcal{D}[\![C[s_2]]\!]$, hence $\mathcal{O}[\![C[s_1]]\!] = \mathcal{O}[\![C[s_2]]\!]$ by theorem 5.3.19. We conclude that, for any $s_1, s_2 \in Stat$,

$$\mathcal{D}(s_1) = \mathcal{D}(s_2) \iff \mathcal{O}[\![C[s_1]]\!] = \mathcal{O}[\![C[s_2]]\!] \text{ for any context } C[\bullet]$$

i.e., $\mathcal{D}\colon Stat \to SemRef \to \mathbb{P}_d$ is fully abstract with respect to $\mathcal{O}[\![\bullet]\!]$. □

## 5.4 Conclusions and bibliographical remarks

For both the abstract process languages $\mathcal{L}_{ref}$ with nondeterministic choice, parallel composition and action refinement and $\mathcal{L}_{pr}$ with probabilistic choice and action refinement we have developed an operational semantics $\mathcal{O}$ using refinement sequences and a denotational semantics $\mathcal{D}$ using semantical refinements. In both cases the denotational model is shown to be fully abstract with respect to the operational model; the denotational semantics identifies exactly those statements that have the same operational meaning in all contexts. The work on $\mathcal{L}_{ref}$ in section 5.2 is mainly based on the paper [112]. It shows that the modeling of action refinement does not enforce a true concurrency setting. Instead an interleaving setting with identification of statements which coincide under all refinements can also be used. Also the advantages of the metric setting in the definition of operational and denotational semantics and in obtaining the full abstractness result are illustrated.

The language $\mathcal{L}_{ref}$ contains parallel composition but does not allow communication between parallel components. Synchronization of parallel processes, can however be added at the cost of complicating the denotational model and the full abstractness proof. The paper [112] provides this addition of synchronization to the language $\mathcal{L}_{ref}$ by extending work reported in [178, 38, 50]. Failure sets are employed to obtain a fully abstract model for the extended language (cf. [192, 193, 195] for a similar result for a CSP-style process language).

Related work in the area of action refinement was already discussed in the introduction of this chapter 5.1. An extensive overview of work done in the area of action refinement may also be found in [97].

The work on $\mathcal{L}_{pr}$ in section 5.3 comes mainly from [113]. This work shows that the general techniques for metric operational and denotational semantics remain in place in the setting of probabilistic programming. In fact, the domain of probability measures of compact support is a suitable complete ultrametric space for the modeling of discrete probabilistic choice, also in the presence of a specific construct such as action refinement. In particular, the method for proving full abstractness of the denotational model of section 5.2 —based on the distance of two statements that have different meanings in the denotational model— carries over to the setting of the probabilistic domain. This also shows the suitability of the domain of compact support measures in the metric modeling of discrete probabilistic choice.

Several papers deal with full abstractness in a setting with probabilistic choice. In [147], extending the earlier [142], a full abstractness result is obtained for a metric denotational model with respect to probabilistic bisimulation from [149]. In [28] a metric denotational semantics for an extension of CCS with action guarded probabilistic choice is shown to be fully abstract with respect to probabilistic bisimulation. The papers [57, 58, 168, 99] deal with fully abstract models for probabilistic choice in the setting of testing semantics.

For a discussion on related work in the area of probabilistic languages the reader is referred to the beginning of section 5.3 as well as the discussion in section 3.6.

# Chapter 6

# A probabilistic Hoare-style logic

## 6.1 Introduction

Probabilistic elements in a program usually presents additional problems in understanding and testing of the program. Additional tools, such as formal verification techniques, are useful in monitoring and directing the software development process. Several approaches, discussed in the introduction of this thesis, exist that start from a mathematical reconstruction of the system under consideration. Models that are often used are Markov chains and Markov decision processes (see, e.g., [117, 29]) and probabilistic input-output automata (cf. [181, 199], for example). Model checking based techniques provide a logical language to characterize program properties and exploit automated tools to exhaustively search the state space (consult, e.g., [174, 129, 3, 114]).

For a wide range of programs the construction of the mathematical model can already be problematic. A systematic approach to simplify the program, or to obtain properties without having to actually calculate the semantics is useful. Approaches in this area are probabilistic process algebra and stochastic process algebra (see, for example, [19, 168, 8, 70, 64]) where equivalences of programs can be checked syntactically by equational reasoning.

Another approach is to introduce a calculus or a proof system as a vehicle to reason about the probabilistic programs directly. Earlier work on the proof theory for probabilistic programs that has inspired the present work, can be found in e.g. [141, 133]. In [141] Kozen proposes a probabilistic dynamic logic in which arithmetical laws govern the program analysis. The thesis work of Jones [133], presents a proof system for probability in a state-less setting. An important strand of research in the syntactic approach (cf. [162, 163]) is focused on predicate transformers. In [162], extending the predicate transformer work of [163], a calculus of greatest pre-expectations is given for a language with both probabilistic choice and nondeterminism. This calculus is illustrated by its application to the examples of an erratic 'sequence accumulator', an example recurring here, and Rabin's 'choice coordination' algorithm.

The main contribution of this thesis on this topic is the proposal of a sound proof system for reasoning about a probabilistic extension of sequential programming. The assertion language, because of the introduction of probabilistic predicates, allows for expression of probabilities of deterministic predicates, which includes the possibility to state that a program variable has a certain standard probability distribution. For a fragment of the language a weakest precondition calculus and a proof of completeness are also provided. In some examples the relative strength of the probabilistic Hoare logic is illustrated.

The main difference between the work of Morgan et al. and ours is that we take Hoare logic with probabilistic predicates as our starting point. We are therefore in a position to justify the proof rules with respect to a denotational semantics. Moreover, from a correctness point of view, our probabilistic predicates seem intuitively more attractive. The expectation based calculus on the other hand provides a way to generate useful quantitative information, whereas the emphasis in our approach is on the verification of given quantitative information. Further study should shed more light on the relative merits of the two methods.

In general Hoare logic is a system to derive (partial) correctness formulae, also known as Hoare triples (see [126, 34]). A formula $\{\, p \,\}\ s\ \{\, q \,\}$ states that the predicate $p$ is a sufficient precondition for the program $s$ to guarantee that predicate $q$ holds after termination. What the values of the variables in a program, i.e. the (deterministic) state of the program, will be, cannot be fully determined if the program is probabilistic. Only the probability of being in a certain state can be given, thus yielding the notion of a probabilistic state. In a probabilistic state, a deterministic predicate will no longer be either true or false, but it is true with a certain probability. This can, for example, be dealt with by changing the interpretation of validity of a predicate to a function to $[0, 1]$ or $\mathbb{R}$ instead of a function to $\{\, \mathit{true}, \mathit{false} \,\}$ as in [141, 162]. The approach chosen here is to extend the syntax of predicates to allow making claims about the probability that a certain deterministic predicate holds. We refer to the extended form of predicates as probabilistic predicates. A proof system for probabilistic programs should take these probabilistic predicates into account.

After mathematical preliminaries in section 6.2, the language $\mathcal{L}_{\mathrm{pif}}$ with conditional and probabilistic choice is introduced in section 6.3. Additionally, a denotational semantics for $\mathcal{L}_{\mathrm{pif}}$ is provided. Section 6.4 discusses probabilistic predicates and presents the Hoare-logic for $\mathcal{L}_{\mathrm{pif}}$. The relationship with weakest preconditions and the completeness of the Hoare-logic is the subject of section 6.5. Section 6.6 extends $\mathcal{L}_{\mathrm{pif}}$ with the construct of iteration, yielding $\mathcal{L}_{\mathrm{pw}}$. Furthermore a denotational semantics for $\mathcal{L}_{\mathrm{pw}}$ and a proof system are included. Section 6.7 extends $\mathcal{L}_{\mathrm{pif}}$ with an operator for nondeterministic choice. Section 6.8 finishes this chapter with concluding remarks and indicates further work.

## 6.2 Mathematical preliminaries

In previous chapters the semantical models were given using a metrical approach. In the metric setting it is easy to deal with infinite behavior. The metric setting also provides techniques for the comparison of different models, in particular the operational and denotational semantics. In this chapter we are dealing with a functional, input-output view of programs, not with the control flow of the programs. As such we are not interested in infinite behavior (as this produces no final output). Furthermore only a denotational semantics will be considered. As such the advantages of the metric approach mentioned above do not apply. A complete partial order based approach is used instead of a metric approach.

A complete partially ordered set (cpo) is a set with a partial ordering $\leq$ that has a least element and for which each ascending chain has a least upper bound within the set. The least upperbound of a chain $(x_i)_{i\in\mathbb{N}}$ is denoted by $\lim_{i\to\infty} x_i$. A function $f$ from a cpo $X$ to a cpo $Y$ is called monotone if for all $x, x' \in X$ we have that $x \leq x'$ implies $f(x) \leq f(x')$. A monotone function $f : X \to X$ on a cpo $X$ has a least fixed point $\mathit{fix}(f) \in X$.

An ordering on $Y$ is extended pointwise to functions from $X$ to $Y$ (for $f, g : X \to Y$ we have $f \leq g$ if $f(x) \leq g(x)$ for all $x \in X$). The support of a function $f : X \to [0,1]$ is defined as those $x \in X$ for which $f(x) \neq 0$. The set of all functions from $X$ to $[0,1]$ with countable support is denoted by $X \to_{CS} [0,1]$. Given a function $f : X \to_{CS} [0,1]$ and a set $Y \subseteq X$ the sum $\sum f[Y] = \sum_{y\in Y} f(y)$ is well-defined (allowing the value $\infty$). The set of (pseudo) probabilistic distributions $\mathit{Dist}(X)$ over a set $X$ is defined as the subset of functions in $X \to_{CS} [0,1]$ with sum at most 1, i.e.,

$$\mathit{Dist}(X) = \{\, f \in X \to_{CS} [0,1] \mid \textstyle\sum f[X] \leq 1 \,\}$$

For a distribution $f \in \mathit{Dist}(X)$ and an element $x \in X$, $f(x)$ is interpreted as the probability that $x$ occurs. The sum $\sum f[X]$ is called the total probability of the distribution $f$. The set $\mathit{Dist}(X)$ endowed with the ordering induced by $\leq$ on $[0,1]$ is a cpo. The minimal element of $\mathit{Dist}(X)$ is $\underline{0}$, the function that assigns 0 to each element of $X$. For each ascending sequence in $\mathit{Dist}(X)$ the limit exists within $\mathit{Dist}(X)$ and corresponds to the least upper bound of the sequence. A distribution with total probability less than one, indicates a situation with partial information. Intuitively, part of the distribution is not known; either because it is calculated elsewhere or because it is not reached at all (due to non-termination).

For elements $x \in X$ and $y \in Y$ and a function $f : X \to Y$, the function $f[y/x]$, called a variant of $f$, is defined by

$$f[y/x](x') \quad = \quad \begin{cases} y & \text{if } x = x' \\ f(x') & \text{otherwise} \end{cases}$$

## 6.3 Syntax and semantics of $\mathcal{L}_{\mathrm{pif}}$

In this section the language $\mathcal{L}_{\mathrm{pif}}$ is introduced and a denotational semantics $\mathcal{D}$ for $\mathcal{L}_{\mathrm{pif}}$ is given. The language $\mathcal{L}_{\mathrm{pif}}$ is a basic programming language containing skip, assignment, sequential composition, conditional choice and probabilistic choice.

Programs in the language $\mathcal{L}_{\mathrm{pif}}$ are interpreted as state transformers, where the state is comprised of the data stored in the variables. The value of a variable can be changed by executing an assignment. An assignment, written as $\mathtt{x} := e$, can change the value belonging to the variable $\mathtt{x}$ to the value represented by the expression $e$. Each variable has a data type associated with it. Each data type has its own syntax for the expressions.

Assignments are assumed to be type correct, i.e. only expressions of the right type are assigned to a variable. For readability, the definitions will be given only for a single type of data, the integers. The notation *Int* is used for the set of all integers. In the examples other types, like arrays of integers, will also be used.

The variables used in the programs in $\mathcal{L}_{\mathrm{pif}}$ are from a given set *PVar*, called the set of *program variables*. For a deterministic program, the values of all the variables form the state of the computation, i.e. a deterministic state is a function which assigns an integer to each program variable.

**Definition 6.3.1** *The set of program variables is denoted by PVar and ranged over by x and y. The set of integers, denoted by Int is extended with the element $\perp$ to the set $Int_\perp$. The ordering on the integers is extended by putting $\perp < n$ for each integer $n$.*

*The set of deterministic states $\mathcal{S}$, ranged over by $\sigma$, is given by*

$$\mathcal{S} \;=\; PVar \to Int_\perp$$

The set $Int_\perp$ is the collection of integers to which $\perp$ is added. The symbol $\perp$ is used for undefinedness. Operations on the integers yield $\perp$ when the operation is not defined, e.g. for division by zero, and when any of the arguments is undefined. To avoid many technical complications associated with undefined expressions, the symbol $\perp$ is treated as an ordinary value which is smaller than any integer $n \in Int$ thus extending the ordinary ordering on *Int*. The choice to make $\perp$ the smallest element of $Int_\perp$ is an arbitrary one.

The conditional choice uses boolean conditions to choose between statements. Below the syntax for integer expressions and boolean conditions is given, followed by the syntax of the language $\mathcal{L}_{\mathrm{pif}}$.

As expressions are needed over several sets of variables, the definition of expressions uses a general set of variables. Given a set of variables, say *Var*, the set of integer expressions over *Var* is denoted by $Exp\langle Var\rangle$. The value of an expression can be found using the evaluation function $\mathcal{V}$.

**Definition 6.3.2** *Let Var denote a set of variables and let v range over Var. The set of integer expressions over Var, denoted by $Exp\langle Var\rangle$ and ranged over by e, is given by*

$$e \;=\; n \mid v \mid e + e \mid e - e \mid e \cdot e \mid e \,\texttt{div}\, e \mid e \,\texttt{mod}\, e \mid \ldots$$

*where $n$ denotes any element of $Int_\perp$. The evaluation function $\mathcal{V} : Exp\langle Var\rangle \to (Var \to Int_\perp) \to Int_\perp$ that computes the value of an expression given the values of the variables is defined by:*

$$\mathcal{V}(n)(f) = n \qquad \mathcal{V}(v)(f) = f(v) \qquad \mathcal{V}(e \;\texttt{op}\; e')(f) = \mathcal{V}(e)(f)\; op\, \mathcal{V}(e')(f)$$

*for $f \in Var \to Int_\perp$ and $op \in \{ +, -, \cdot, \texttt{div}, \texttt{mod} \}$.*

A basic integer expression is a constant $n$ or a variable $v$. Integer expressions can be combined with the usual operators on integers. Basically any function on integers that is deemed useful could also be added. The evaluation function calculates the value of an expression given the value of the variables. A function $f$ supplies the values of the variables. In the special case that $Var = PVar$ the function $f$ is exactly a state in $\mathcal{S}$.

**Example 6.3.3** *Let $f$ denote a function giving the values of the variables, $f : \mathit{Var} \to \mathit{Int}_\bot$. The value of the expression $(1 \texttt{ div } 0) + 1$ is $\mathcal{V}((1 \texttt{ div } 0) + 1)(f) = \bot + 1 = \bot$. The value of the expression $v - 2$ depends on the value of $v$ given by $f$: $\mathcal{V}(v-2)(f) = f(v) - 2$.*

Conditional choices in the program are based on boolean conditions. As with integer expressions the boolean conditions over a set of variables are given for a general set of variables $\mathit{Var}$. The set of all boolean conditions over the set of variables $\mathit{Var}$ is denoted by $BC\langle \mathit{Var} \rangle$ and a typical boolean condition is denoted by $c$. True, false and relational expressions are the basic conditions. Conditions can be combined using the standard logical operators.

**Definition 6.3.4** *The set of boolean conditions over Var, denoted by $BC\langle \mathit{Var} \rangle$ and ranged over by $c$, is given by*

$$c \quad = \quad \texttt{true} \mid \texttt{false} \mid e \texttt{=} e \mid e < e \mid \ldots \mid c \wedge c \mid c \vee c \mid \neg c \mid c \to c$$

*where $e$ is an expression in $\mathit{Exp}\langle \mathit{Var} \rangle$. The value of a boolean condition is a boolean in $\mathit{Bool} = \{\, \mathit{true}, \mathit{false} \,\}$. The evaluation function $\mathcal{B} : BC\langle \mathit{Var} \rangle \to (\mathit{Var} \to \mathit{Int}_\bot) \to \mathit{Bool}$, that computes the value of a boolean condition given the values of the variables provided by $f$, is defined by:*

$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{B}(\texttt{true})(f) & = & \mathit{true} & \mathcal{B}(e \texttt{ rel } e')(f) & = & \mathcal{V}(e)(f) \mathrel{\mathit{rel}} \mathcal{V}(e')(f) \\
\mathcal{B}(\texttt{false})(f) & = & \mathit{false} & \mathcal{B}(c \texttt{ op } c')(f) & = & \mathcal{B}(c)(f) \mathbin{\mathit{op}} \mathcal{B}(c')(f) \\
\mathcal{B}(\neg c)(f) & = & \neg \mathcal{B}(c)(f) & & &
\end{array}$$

*where $f$ is a function in $\mathit{Var} \to \mathit{Int}_\bot$, `rel` is `=`, $<$, ... and `op` is $\wedge$, $\vee$ or $\to$.*

The meaning of `true`, `false` and the logical connectives is clear. The value of the variables is needed to evaluate the expressions that occur in the boolean conditions. The boolean condition $e \texttt{=} e'$ is true when $e$ and $e'$ evaluate to the same value. The condition $e < e'$ is true when the value $e$ evaluates to is smaller than the value that $e'$ evaluates to.

**Example 6.3.5** *For $x, y \in \mathit{Var}$, the boolean condition $(x > 0)$ states that $x$ is positive, $(x \texttt{ mod } y = 0)$ expresses that $x$ can be divided by $y$ and $\neg\ (x \texttt{=} y)$ expresses that $x$ and $y$ are different. Note that $(x \texttt{ div } 0) \leq y$ is always true, as $\bot$ is the smallest element of $\mathit{Int}_\bot$.*

Having defined expressions and boolean conditions, the syntax of the language $\mathcal{L}_{\text{pif}}$ can now be made precise. As mentioned, the language $\mathcal{L}_{\text{pif}}$ contains skip, assignment, sequential composition, conditional choice and probabilistic choice.

**Definition 6.3.6** *The programs in $\mathcal{L}_{\text{pif}}$, ranged over by $s$, are given by:*

$$s ::= \texttt{skip} \mid x \texttt{ := } e \mid s \texttt{ ; } s \mid \texttt{if } c \texttt{ then } s \texttt{ else } s \texttt{ fi} \mid s \oplus_\rho s$$

*where $x$ is a program variable in PVar, $e$ is an expression in $\mathit{Exp}\langle \mathit{PVar} \rangle$, $c$ is a condition in $BC\langle \mathit{PVar} \rangle$ and $\rho$ is a ratio in the open interval $(0, 1)$.*

The program `skip` does nothing. The program $x$ `:=` $e$ assigns the value of the expression $e$ to the variable $x$. The program $s$ `;` $s'$ is executed by first executing $s$ and then executing $s'$. The program `if` $c$ `then` $s$ `else` $s'$ `fi` is executed by evaluating the condition $c$ and executing $s$ if the condition is true and $s'$ if the condition is false. The program $s \oplus_\rho s'$ denotes the probabilistic choice between the programs $s$ and $s'$; with probability $\rho$ the program $s$ is chosen and the program $s'$ is chosen with probability $1 - \rho$.

For a deterministic program that is interpreted as a state transformer, the state of the computation is given by the values of the variables. This is why the state space for deterministic programs $\mathcal{S}$ (see definition 6.3.1) was chosen as $\mathcal{S} = \mathit{PVar} \to \mathit{Int}_\perp$. For a probabilistic program, the values of the variables are no longer determined. For example, after executing $x$ `:= 0` $\oplus_{\frac{1}{2}}$ $x$ `:= 1`, the value of $x$ could be zero but it could also be one. Instead of giving the value of a variable, a distribution over possible values should be given. A first idea may be to take as a state space $\mathit{PVar} \to \mathit{Dist}(\mathit{Int}_\perp)$. This does give, for each variable $x$, the chance that $x$ takes a certain value but it does not describe the possible dependencies between the variables.

Let us, in order to clarify this matter, consider the following example. In the left situation (see below), a fair coin is thrown and a second coin is put beside it with the same side up. In the right situation, two fair coins are thrown. The two situations are indistinguishable if the dependency between the two coins is not known; the probability of heads or tails is $\frac{1}{2}$ for both coins in both situations. The difference between the situations is important, e.g. if the next step is comparing the coins. In the first situation the coins are always equal, in the second situation they are equal with probability $\frac{1}{2}$ only.

| | | coin 1 | |
|---|---|---|---|
| | | heads | tails |
| coin 2 | heads | $\frac{1}{2}$ | 0 |
| | tails | 0 | $\frac{1}{2}$ |

| | | coin 1 | |
|---|---|---|---|
| | | heads | tails |
| coin 2 | heads | $\frac{1}{4}$ | $\frac{1}{4}$ |
| | tails | $\frac{1}{4}$ | $\frac{1}{4}$ |

Clearly the dependencies between the variables must also be expressed in the probabilistic state. Therefore, the more general state space $\Theta = \mathit{Dist}(\mathit{PVar} \to \mathit{Int}_\perp)$ is required. In $\theta \in \Theta$, instead of giving the distributions for the variables separately, the probability of being in a certain deterministic state is given. The chance that a variable $x$ takes value $n$ can be found by summing the probabilities of all deterministic states which assign $n$ to $x$.

**Definition 6.3.7** *The set of probabilistic states $\Theta$, ranged over by $\theta$, is given by*

$$\Theta \;=\; \mathit{Dist}(\mathcal{S})$$

As a program is interpreted as a state transformer, the meaning of a program is a function from states to states. This function returns the end state after execution of the program given a start state. The meaning of a program in $\mathcal{L}_{\mathrm{pif}}$ is given by a denotational semantics $\mathcal{D}$.

A key step in the definition of the denotational semantics is to give the meaning of a basic program consisting of a single assignment, $x$ `:=` $e$. For a deterministic state $\sigma$, the state resulting from executing the program $x$ `:=` $e$ in state $\sigma$ is a variant of the state $\sigma$ where the value of $e$ is assigned to the variable $x$: $\sigma[\mathcal{V}(e)(\sigma)/x]$. In a deterministic state $\sigma$ the value of an integer expression $e$ can be found by using the evaluation function $\mathcal{V}$.

One would also like to define a notion of a variant of a probabilistic state which can be used to define the effect of assignment in a probabilistic state. Simply using a construction $[v/\mathrm{x}]$ where a (fixed) value is assigned to the variable, however, is not sufficient. In a probabilistic state $\theta$ the values of the variables are, in general, not fixed and a single value cannot be found for an integer expression.

The variant of a probabilistic state should be the probabilistic state after assignment of an expression $e$ to a variable $\mathrm{x}$. To find the probabilistic state after executing an assignment, the assignment has to be done in each deterministic state which has a positive probability. For each of these deterministic states, the assignment is done by taking a variant of the state. The value assigned to $\mathrm{x}$ depends on the deterministic state. This gives the form the variant of a probabilistic state should have: a variant of a probabilistic state should assign a new value to a variable, depending on the deterministic states being considered.

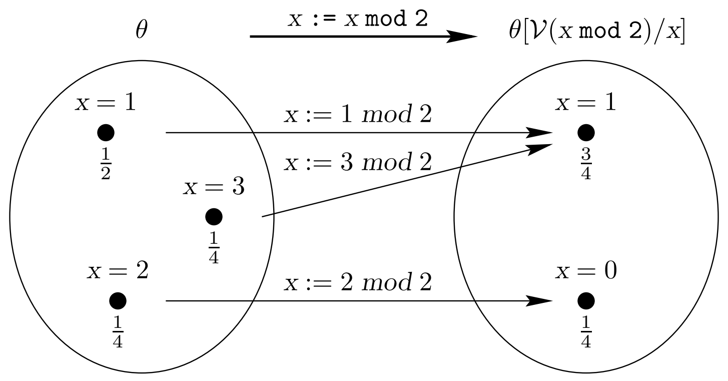
**Definition 6.3.8** *Let $f : \mathcal{S} \to Int_\perp$ and let $\theta$ be a probabilistic state in $\Theta$. The variant of $\theta$, denoted by $\theta[f/\mathrm{x}]$, is given by: $\theta[f/\mathrm{x}](\sigma) = \sum_{\sigma' \in V} \theta(\sigma')$, with $V = \{\, \sigma' \in \Theta \mid \sigma'[f(\sigma')/\mathrm{x}] = \sigma \,\}$.*

The function $f$ gives the value to use for a given deterministic state. To find the probability of ending up in a deterministic state $\sigma$, the probability of all deterministic states $\sigma'$ that become $\sigma$ after taking the variant using the value $f(\sigma')$, have to be added. Straightforward calculation shows that $\theta[f/\mathrm{x}]$ is again a pseudo probabilistic distribution in $\Theta$. In fact, $\theta[f/\mathrm{x}]$ has the same total probability as $\theta$. The following example shows how the variant of a probabilistic state can be used to calculate the effect of assignment on a probabilistic state.

**Example 6.3.9** *As we have*

$$\begin{array}{lclcl}
\langle \mathrm{x}=1\rangle[\mathcal{V}(\mathrm{x}\,\texttt{mod}\,2)(\langle \mathrm{x}=1\rangle)/\mathrm{x}] & = & \langle \mathrm{x}=1\rangle[1/\mathrm{x}] & = & \langle \mathrm{x}=1\rangle \\
\langle \mathrm{x}=2\rangle[\mathcal{V}(\mathrm{x}\,\texttt{mod}\,2)(\langle \mathrm{x}=2\rangle)/\mathrm{x}] & = & \langle \mathrm{x}=2\rangle[0/\mathrm{x}] & = & \langle \mathrm{x}=0\rangle \\
\langle \mathrm{x}=3\rangle[\mathcal{V}(\mathrm{x}\,\texttt{mod}\,2)(\langle \mathrm{x}=3\rangle)/\mathrm{x}] & = & \langle \mathrm{x}=3\rangle[1/\mathrm{x}] & = & \langle \mathrm{x}=1\rangle
\end{array}$$

*assigning $\mathrm{x}\,\texttt{mod}\,2$ to $\mathrm{x}$ in the probabilistic state $\theta = \frac{1}{2}\langle \mathrm{x}=1\rangle + \frac{1}{4}\langle \mathrm{x}=2\rangle + \frac{1}{4}\langle \mathrm{x}=3\rangle$ gives the probabilistic state $\theta[\mathcal{V}(\mathrm{x}\,\texttt{mod}\,2)/\mathrm{x}] = \frac{1}{2}\langle \mathrm{x}=1\rangle + \frac{1}{4}\langle \mathrm{x}=0\rangle + \frac{1}{4}\langle \mathrm{x}=1\rangle = \frac{3}{4}\langle \mathrm{x}=1\rangle + \frac{1}{4}\langle \mathrm{x}=0\rangle$.*



The evaluation function $\mathcal{V}$ for a fixed expression $e$ is a function $\mathcal{V}(e)$ from $\mathcal{S}$ to $Int_\perp$; $\mathcal{V}(e)$ is a function of the right type for use in the variant of a probabilistic state.

Having described the meaning of the basic assignment statement, the next step in giving the denotational semantics for $\mathcal{L}_{\text{pif}}$ is defining the effect of the operations in $\mathcal{L}_{\text{pif}}$. Sequential composition is not difficult to deal with. The following operations on probabilistic states are used to define the effect of the operators probabilistic choice and conditional choice.

**Definition 6.3.10** *The operations probabilistic choice $\oplus_\rho : \Theta \times \Theta \to \Theta$ for $\rho \in (0,1)$ and unscaled conditional $c? : \Theta \to \Theta$ for $c \in BC\langle PVar\rangle$ are defined as follows:*

$$\begin{array}{rcl} \theta_1 \oplus_\rho \theta_2 & = & \rho\theta_1 + (1-\rho)\theta_2 \\ c?\theta(\sigma) & = & \begin{cases} \theta(\sigma) & \textit{if } c \textit{ is true in } \sigma \textit{ i.e. } \mathcal{B}(c)(\sigma) = \textit{true} \\ 0 & \textit{otherwise} \end{cases} \end{array}$$

*where $+$ is standard addition of functions and $\rho\cdot$ is scalar multiplication.*

Recall that $\theta \in \Theta$ is a function from $\mathcal{S}$ to $[0,1]$. The value $\theta(\sigma)$ returned by $\theta$ is the probability of being in the deterministic state $\sigma$. The operation $\oplus_\rho$ simply combines two probabilistic states with the appropriate probabilities. The probabilistic state $c?\theta$ is obtained from $\theta$ by removing any probability for deterministic states not satisfying $c$. As straightforward calculations show, for any probabilistic state $\theta$ the equations $\theta \oplus_\rho \theta = \theta$ and $c?\theta + \neg c?\theta = \theta$ hold.

**Example 6.3.11** *Using the notation of example 6.3.9 the effect of the operators $\oplus_\rho$ and $c?$ is easy to see:*

$$\begin{array}{rcl} 1\langle \mathrm{x}=1\rangle \;\oplus_{\frac{1}{2}}\; (\frac{1}{2}\langle \mathrm{x}=2\rangle + \frac{1}{2}\langle \mathrm{x}=3\rangle) & = & \frac{1}{2}\langle \mathrm{x}=1\rangle + \frac{1}{4}\langle \mathrm{x}=2\rangle + \frac{1}{4}\langle \mathrm{x}=3\rangle \\ (\mathrm{x}\;\mathtt{mod}\;\mathtt{2}\;\mathtt{=}\;\mathtt{1})?\big(\frac{1}{2}\langle \mathrm{x}=1\rangle + \frac{1}{4}\langle \mathrm{x}=2\rangle + \frac{1}{4}\langle \mathrm{x}=3\rangle\big) & = & \frac{1}{2}\langle \mathrm{x}=1\rangle + \frac{1}{4}\langle \mathrm{x}=3\rangle \end{array}$$

The denotational semantics $\mathcal{D}$ for $\mathcal{L}_{\text{pif}}$ gives, for each program $s$ in $\mathcal{L}_{\text{pif}}$ and state $\theta$ in $\Theta$, the state $\mathcal{D}(s)(\theta)$ resulting from executing $s$ starting in state $\theta$.

**Definition 6.3.12** *The denotational semantics $\mathcal{D} : \mathcal{L}_{\text{pif}} \to (\Theta \to \Theta)$ is given by*

$$\begin{array}{rcl} \mathcal{D}(\mathtt{skip})(\theta) & = & \theta \\ \mathcal{D}(\mathrm{x} \mathrel{\mathtt{:=}} e)(\theta) & = & \theta[\mathcal{V}(e)/\mathrm{x}] \\ \mathcal{D}(s \mathrel{;} s')(\theta) & = & \mathcal{D}(s')(\mathcal{D}(s)(\theta)) \\ \mathcal{D}(s \oplus_\rho s')(\theta) & = & \mathcal{D}(s)(\theta) \oplus_\rho \mathcal{D}(s')(\theta) \\ \mathcal{D}(\mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s'\ \mathtt{fi})(\theta) & = & \mathcal{D}(s)(c?\theta) + \mathcal{D}(s')(\neg c?\theta) \end{array}$$

The clause for `skip` is clear; the execution of `skip` leaves the state unchanged. The clause for assignment uses the notion of a variant of a probabilistic state introduced above. The evaluation function for the expression $e$ is assigned to the variable $\mathrm{x}$. The clause for sequential composition is as usual. To find the state after executing $s$ ; $s'$ in state $\theta$, $s'$ is executed in the state resulting from executing $s$, i.e. in the state $\mathcal{D}(s)(\theta)$. The clause for the probabilistic choice between $s$ and $s'$ uses the probabilistic choice between states as introduced in definition 6.3.10. To execute the program `if` $c$ `then` $s$ `else` $s'$ `fi`, the program $s$ is executed in the part of the state satisfying $c$ and the program $s'$ is executed in the part of the state satisfying $\neg c$.

**Example 6.3.13** *Using definition 6.3.12 and examples 6.3.9 and 6.3.11:*

$$\begin{aligned}\mathcal{D}((x \texttt{ := } 1 \oplus_{\frac{1}{2}} \texttt{skip}); x \texttt{ := } x \bmod 2)(\tfrac{1}{2}\langle x = 2\rangle + \tfrac{1}{2}\langle x = 3\rangle) \\ = \mathcal{D}(x \texttt{ := } x \bmod 2)(\tfrac{1}{2}\langle x = 1\rangle + \tfrac{1}{4}\langle x = 2\rangle + \tfrac{1}{4}\langle x = 3\rangle) \\ = \tfrac{3}{4}\langle x = 1\rangle + \tfrac{1}{4}\langle x = 0\rangle\end{aligned}$$

The following property of the denotational semantics can easily be checked by structural induction.

**Lemma 6.3.14** *The denotational semantics is linear in the probabilistic state:*

$$\mathcal{D}(s)(\theta + \theta') = \mathcal{D}(s)(\theta) + \mathcal{D}(s)(\theta') \quad \textit{and} \quad \mathcal{D}(s)(\rho\theta) = \rho\mathcal{D}(s)(\theta)$$

In contrast we have that in general $\mathcal{D}(s)(c?\theta) \neq c?\mathcal{D}(s)(\theta)$ as the execution of $s$ may influence the value of the condition $c$. Note that $\theta + \theta'$ is only a state in $\Theta$ if the total probability of the two states combined does not exceed 1. For all cases considered here this will be the case. To make + well-defined for all pairs of states, one can define $\theta + \theta'$ to be $\underline{0}$ for all pairs of states $\theta$, $\theta'$ which have a combined total probability greater than one.

The denotational semantics can be used to check properties of programs and to check whether two programs have the same semantics, i.e. whether two programs are *equivalent*. In the next section probabilistic predicates are introduced to express probabilistic properties and a proof system is introduced to be able to deduce properties of programs without having to calculate the semantics of the program.

## 6.4 Probabilistic predicates and Hoare logic

The denotational semantics gives the meaning of programs in $\mathcal{L}_{\text{pif}}$. One would like to be able to check claims about a program like “after executing the program property $p$ will hold”. In this section a probabilistic logic is given. The predicates in the logic specify properties of probabilistic states. The claim above becomes “the predicate $p$ holds in the state resulting from executing the program”. The state resulting from the execution of the program may depend on the state at the start of the program. A more precise claim about the behavior of a program is therefore “if the state before execution satisfies the predicate $q$ then after executing the program, the state will satisfy the predicate $p$”. Hoare triples, also known as correctness formulae, express exactly such claims.

To deduce which claims, i.e. which Hoare triples, are valid, a proof system $pH$ is introduced. The proof system $pH$ is based on standard Hoare logic for non-probabilistic programs. The rules known from Hoare logic are adapted to fit the probabilistic setting and several new rules are added. The proof system is shown to be sound (with respect to the denotational semantics) by showing that any Hoare triple that can be deduced from the proof system is valid.

For a deterministic state a basic property that one wants to check is e.g. "variable $x$ has value $n$". For a probabilistic state, an example of a basic property is "variable $x$ has value $n$ with probability at least $\frac{1}{2}$". Below the exact syntax of deterministic predicates and Hoare triples are given after which the probabilistic predicates are introduced.

The property "variable $x$ has value $n$" can be captured by the boolean condition $x \mathtt{=} n$. The deterministic predicates extend the boolean conditions by adding the quantifies $\forall$ and $\exists$ ranging over the integers. A set of integer valued variables is used to introduce these quantifiers.

**Definition 6.4.1** *Let IVar be a set of integer valued variables ranged over by* $\mathtt{i}$. *The set of deterministic predicates, denoted by DPred and ranged over by dp, is given by*

$$\begin{aligned} dp \quad ::= \quad & \mathtt{true} \mid \mathtt{false} \mid e \mathtt{=} e \mid e < e \mid \ldots \mid dp \wedge dp \mid dp \vee dp \mid \neg\, dp \mid \\ & dp \rightarrow dp \mid \forall \mathtt{i} : dp \mid \exists \mathtt{i} : dp \end{aligned}$$

*where* $e \in Exp\langle PVar \cup IVar\rangle$ *is an expression over program variables and integer valued variables. The set* $\mathcal{I}$ *of all interpretations of integer valued variables is given by* $\mathcal{I} = IVar \rightarrow Int_\perp$. *A typical interpretation is denoted by* $I$. *The satisfaction relation for deterministic predicates,* $\models$, *works as expected*

$$\begin{array}{llll}
(\sigma, I) & \models & \mathtt{true} & \\
(\sigma, I) & \not\models & \mathtt{false} & \\
(\sigma, I) & \models & e\,\mathtt{rel}\,e' & \textit{when} \quad \mathcal{V}(e)(\sigma, I)\ \textit{rel}\ \mathcal{V}(e')(\sigma, I) \\
(\sigma, I) & \models & dp \wedge dp' & \textit{when} \quad (\sigma, I) \models dp \textit{ and } (\sigma, I) \models dp' \\
(\sigma, I) & \models & dp \vee dp' & \textit{when} \quad (\sigma, I) \models dp \textit{ or } (\sigma, I) \models dp' \\
(\sigma, I) & \models & \neg\, dp & \textit{when} \quad (\sigma, I) \not\models dp \\
(\sigma, I) & \models & dp \rightarrow dp' & \textit{when} \quad (\sigma, I) \models dp \textit{ implies } (\sigma, I) \models dp' \\
(\sigma, I) & \models & \forall \mathtt{i} : dp & \textit{when} \quad (\sigma, I[\mathtt{i}/n]) \models dp \textit{ for all } n \in Int \\
(\sigma, I) & \models & \exists \mathtt{i} : dp & \textit{when} \quad (\sigma, I[\mathtt{i}/n]) \models dp \textit{ for some } n \in Int
\end{array}$$

*where* $\mathtt{rel}$ *is* $\mathtt{=}, <, \ldots$ .

As mentioned the deterministic predicates are the boolean conditions extended with the quantifiers $\forall$ and $\exists$. The interpretation of deterministic predicates is as usual. Because both integer valued variables and program variables may be used, both a deterministic state $\sigma$ and an interpretation of the integer valued variables $I$ are required to evaluate a deterministic predicate.

**Example 6.4.2** *As* $(\mathtt{i} < 4) \wedge (\mathtt{i} > x)$ *holds in* $(\langle x = 2\rangle, \langle \mathtt{i} = 3\rangle)$, *we have that for any interpretation* $I$: $(\langle x = 2\rangle, I) \models \exists \mathtt{i} : (\mathtt{i} < 4) \wedge (\mathtt{i} > x)$.

Substituting an expression $e$ for the variable $x$ in the predicate is denoted by $[e/x]$. An important property of substitution on deterministic predicates is given by the substitution lemma (cf. [34]) which states that $(\sigma, I) \models dp[e/x]$ holds exactly when $(\sigma[\mathcal{V}(e)(\sigma)/x], I) \models dp$ holds. This property is essential for the soundness of the assignment rule in Hoare logic and for the completeness of Hoare logic. Note that here $[e/x]$ is substitution in a predicate, while $[\mathcal{V}(e)(\sigma)/x]$ is taking a variant of a state.

A deterministic Hoare triple or correctness formula, $\{\, dp \,\}\ s\ \{\, dp' \,\}$, describes that $dp$ is a precondition and $dp'$ is a postcondition of program $s$. The Hoare triple is said to be correct or valid, denoted by $\models \{\, dp \,\}\ s\ \{\, dp' \,\}$, if execution of $s$ in any state that satisfies $dp$ will lead to a state satisfying $dp'$. The execution of the program $s$ can change the value of the program variables, i.e. can change the state. The integer valued variables are not affected by the execution of the program, so their interpretation remains the same. Therefore we have for any program $s$, using $\sigma'$ to denote the state resulting from executing the program $s$ starting in some state $\sigma$,

$$\models \{\, dp \,\}\ s\ \{\, dp' \,\} \quad \text{when} \quad (\sigma, I) \models dp \Rightarrow (\sigma', I) \models dp' \ \ (\text{for all } \sigma \in \mathcal{S},\ I \in \mathcal{I})$$

To extend the Hoare triples to probabilistic programs, a notion of probabilistic predicate has to be introduced. One option is to use the same predicates as for deterministic programs but to change the interpretation of a predicate. A deterministic predicate can be seen as a function from states to $\{\, 0, 1 \,\}$, returning 1 if the state satisfies the predicate and 0 otherwise. The predicates can be made probabilistic by making them into functions to $[0, 1]$, returning the probability that the predicate is satisfied in a probabilistic state (see e.g. [141, 163]). This approach is useful to describe the arithmetical aspects of the probabilities involved in a probabilistic program. There are however two drawbacks. The predicates cannot express claims about the probability, only the value of the predicate gives information about the probabilities. Secondly, to be able to give a compositional definition of the semantics of predicates, the normal logical operators like $\wedge$ have to be extended to work on $[0, 1]$. There seems to be no way to do this which, at the same time, preserves the logical aspects of these operators and does not make assumptions about dependencies between the predicates. This issue is further illustrated in example 6.4.8 below.

The key observation that the chance that a deterministic predicate holds in a probabilistic state is a real number in $[0, 1]$ is also used here, but only as the basis for building probabilistic predicates. Probabilistic predicates as used here are predicates in the usual sense and can only have a truth value, i.e. true or false. The extension to probabilistic predicates is made in the syntax where constructs are added to express claims about probabilities.

Comparison of (integer) expressions forms the basis for deterministic predicates. For probabilistic predicates, comparison of real expressions is used as a starting point.

**Definition 6.4.3** *Let RVar be a set of a real valued variables, ranged over by* $\mathtt{r}$*. The set of real expressions RealExp, ranged over by* $e_r$ *is given by*

$$e_r \quad ::= \quad \rho \mid \mathtt{r} \mid \mathbb{P}(dp) \mid e_r + e_r \mid e_r - e_r \mid e_r * e_r \mid e_r / e_r \mid e_r^{e} \mid \ldots$$

*where* $\rho$ *is a real number and* $e$ *is an integer expression over integer valued variables,* $e \in Exp\langle IVar\rangle$*.*

A basic real expression is a constant $\rho$, a real valued variable $\mathtt{r}$ or the probability $\mathbb{P}(dp)$ of a deterministic predicate $dp$. The expressions can be combined using functions on real numbers like $+$, $-$, $*$ and $/$. Integer expressions over integer valued variables are introduced in the real expressions when using operators which take an integer argument as in $e_r^{e}$. Other operators can be added as needed.

**Example 6.4.4** *The expressions $\frac{1}{4}$, $\frac{1}{2} * \mathbb{P}(x \texttt{ = } \texttt{i})$ and $\mathbb{P}(x < 5) + \texttt{r}^{\texttt{i}}$ are all correct real expressions but $\frac{1}{2}^{x}$ is not, as program variables cannot be used in real expressions, except within a deterministic predicate in the $\mathbb{P}(dp)$ construct.*

To evaluate a real expression, like $\mathbb{P}(x{<}5){+}\texttt{r}^{\texttt{i}}$, one needs to know the values of the integer valued and real valued variables that are used in the expression. For a program variable, not the value but rather its distribution is required. The values of the integer valued and real valued variables are given by an interpretation function $J$. The distribution of the program variables is given by a probabilistic state $\theta$. This also explains why a program variable cannot be used outside of the $\mathbb{P}(dp)$ construct; the value of a program variable is not fixed in a probabilistic state $\theta$, only its distribution.

**Definition 6.4.5**

*(a) An interpretation $J$ of integer valued and real valued variables is a function which assigns an element of $\mathrm{Int}_\perp$ to each integer valued variable and an element of $\mathbb{R}_\perp$ to each real valued variable. The set $\mathbb{R}_\perp$ is the reals extended with the element $\perp$ denoting undefined. The restriction of $J$ to integer valued variables is denoted by $J_{\mathcal{I}}$. The set of all interpretations is denoted by $\mathcal{J}$.*

*(b) The evaluation function for real expressions $\mathcal{V}_r : \mathit{RealExp} \to (\Theta \times \mathcal{J}) \to \mathbb{R}_\perp$ is given by*

$$\begin{array}{rcl} \mathcal{V}_r(\rho)(\theta, J) & = & \rho \\ \mathcal{V}_r(\texttt{r})(\theta, J) & = & J(\texttt{r}) \\ \mathcal{V}_r(\mathbb{P}(dp))(\theta, J) & = & \sum_{(\sigma, J_{\mathcal{I}}) \models dp} \theta(\sigma) \\ \mathcal{V}_r(\mathrm{e}_r \; op \; \mathrm{e}'_r)(\theta, J) & = & \mathcal{V}_r(\mathrm{e}_r)(\theta, J) \; op \; \mathcal{V}_r(\mathrm{e}'_r)(\theta, J) \\ \mathcal{V}_r({\mathrm{e}_r}^{e'})(\theta, J) & = & \mathcal{V}_r(\mathrm{e}_r)(\theta, J)^{\mathcal{V}(e')(J_{\mathcal{I}})} \end{array}$$

*with $op \in \{+, -, *, /\}$.*

*(c) The meta variable $\texttt{j}$ is used to range over the union of the set of integer valued variables and the set of real valued variables: $\texttt{j} ::= \texttt{i} \mid \texttt{r}$.*

The value of the constant $\rho$ is $\rho$. The value of the variable $r$ is given by the interpretation $J$. The value of $\mathbb{P}(dp)$ is the probability that $dp$ holds in the given state $\theta$. This probability is found by summing the probabilities of all deterministic states which satisfy $dp$. The operations $+$, $-$, $*$ and $/$ are the normal operations on the reals where $\perp$ is used again to denote that the operation or one of its arguments is undefined, e.g. when dividing by zero.

**Example 6.4.6** *In the state $\frac{1}{4}\langle \mathrm{x} = 1\rangle + \frac{1}{4}\langle \mathrm{x} = 2\rangle + \frac{1}{2}\langle \mathrm{x} = 3\rangle$ and with interpretation $J$ which assigns 1 to $\texttt{i}$, the expression $\frac{1}{2} * \mathbb{P}(\mathrm{x} > \texttt{i})$ evaluates to $\frac{1}{2}(\frac{1}{4} + \frac{1}{2}) = \frac{3}{8}$.*

The real expressions already show how chances are incorporated. The chance on a deterministic predicate holding is simply used as part of the expressions. Using the real expressions, real based conditions are built similarly to the way boolean conditions were built using integer expressions.

**Definition 6.4.7** *The set of real based conditions, denoted by RC and ranged over by $c_r$, is given by*

$$c_r \quad ::= \quad c \mid e_r = e_r \mid e_r < e_r \mid \ldots \mid c_r \vee c_r \mid c_r \wedge c_r \mid \neg c_r \mid c_r \to c_r$$

*where c is a condition over integer valued variables; $c \in BC\langle IVar\rangle$. The evaluation function $\mathcal{B}_r : RC \to (\Theta \times \mathcal{J}) \to Bool$ that computes the value of a real based condition given the values of the variables is given by*

$$\begin{array}{rcl} \mathcal{B}_r(c)(\theta, J) & = & \mathcal{B}(c)(J_{\mathcal{I}}) \\ \mathcal{B}_r(e_r \,\mathtt{rel}\, e_r')(\theta, J) & = & \mathcal{V}_r(e_r)(\theta, J) \; rel \; \mathcal{V}_r(e_r')(\theta, J) \\ \mathcal{B}_r(c_r \,\mathtt{op}\, c_r')(\theta, J) & = & \mathcal{B}_r(c_r)(\theta, J) \; op \; \mathcal{B}_r(c_r')(\theta, J) \\ \mathcal{B}_r(\neg c_r)(\theta, J) & = & \neg\mathcal{B}_r(c_r)(\theta, J) \end{array}$$

*where* `rel` *is* =, $<$, ... *and* `op` *is* $\wedge$, $\vee$ *or* $\to$.

The evaluation of real based conditions is very similar to the evaluation of boolean conditions. The following example shows that real based conditions can express both claims about the distribution of variables as well as claims about the dependencies between variables.

**Example 6.4.8** *Examples of real based conditions are $\big(\mathbb{P}(x = 1) = \frac{1}{2}\big) \wedge \big(\mathbb{P}(y = 1) = \mathbb{P}(x = 1)\big)$ and $\mathbb{P}(x = 1 \wedge y = 1) = \frac{1}{2}$. The first condition gives information about the distributions of x and y and is equivalent with $\big(\mathbb{P}(x = 1) = \frac{1}{2}\big) \wedge \big(\mathbb{P}(y = 1) = \frac{1}{2}\big)$ while the second condition also gives information about the dependency between x and y.*

*In the state $\frac{1}{2}\langle x = 1, y = 1\rangle + \frac{1}{2}\langle x = 2, y = 2\rangle$ both of these conditions are satisfied. In the state $\frac{1}{2}\langle x = 1, y = 2\rangle + \frac{1}{2}\langle x = 2, y = 1\rangle$ only the first condition holds. In the state $\frac{1}{2}\langle x = 1, y = 1\rangle + \frac{1}{2}\langle x = 1, y = 2\rangle$ only the second condition is true.*

The operator $\wedge$ is used in two different contexts. In the real condition $\big(\mathbb{P}(x = 1) = \frac{1}{2}\big) \wedge \big(\mathbb{P}(y = 1) = \frac{1}{2}\big)$ the operator $\wedge$ is used to combine claims about the distribution of $x$ and the distribution of $y$. In the real condition $\mathbb{P}(x = 1 \wedge y = 1) = \frac{1}{2}$. the operator $\wedge$ is used to express a claim about the simultaneous distribution of $x$ and $y$.

Consider again the approach of extending the deterministic predicates to functions to $[0, 1]$ mentioned in the introduction to probabilistic predicates given before definition 6.4.3. The two different uses of the operator $\wedge$ are not incorporated within that approach. Only the second use of the operator, i.e. the use within the construct $\mathbb{P}(\bullet)$, is available. This also shows why a compositional definition of the meaning of such predicates cannot be given without additional assumptions: To find the probability of $x = 1 \wedge y = 1$ one needs the simultaneous distribution of $x$ and $y$. Knowing the probability that $x = 1$ holds and the probability that $y = 1$ holds is insufficient to find this probability. In [163] assumptions about dependencies are made to solve this problem , assumptions that are not satisfied (and also not needed) in our setting.

Real based conditions can be used to express claims about probabilities. The conditions also capture the logical combinations of such claims. Probabilities, however, also have an arithmetical aspect which prompts the interpretation of predicates as real numbers. If interpreted as real numbers, the predicates can be added and scaled, which are typical operations one wants to use when calculating probabilities. These operations are possible on real expressions, but not on the conditions. To allow for arithmetical manipulation of conditions themselves, arithmetical operators are incorporated in the syntax of probabilistic predicates. Nevertheless, probabilistic predicates are still interpreted as functions to truth values.

**Definition 6.4.9** *The set of probabilistic predicates Pred, ranged over by $p$ and by $q$, is given by*

$$p \quad ::= \quad c_r \mid p \wedge p \mid p \vee p \mid \exists j : p \mid \forall j : p \mid \rho \cdot p \mid p + p \mid p \oplus_\rho p \mid c?p$$

*where $c$ is a boolean condition over program variables; $c \in BC\langle PVar\rangle$. The satisfaction relation for probabilistic predicates $\models$ is given by*

$$\begin{array}{rll} (\theta, J) \models c_r & \textit{when} & \mathcal{B}_r(c_r)(\theta, J) = \textit{true} \\ (\theta, J) \models \rho \cdot p & \textit{when} & \exists \theta' : \theta = \rho \cdot \theta' \textit{ and } (\theta', J) \models p \\ (\theta, J) \models p + p' & \textit{when} & \exists \theta_1, \theta_2 : \theta_1 + \theta_2 = \theta, \textit{ and } (\theta_1, J) \models p \textit{ and } (\theta_2, J) \models p' \\ (\theta, J) \models p \oplus_\rho p' & \textit{when} & \exists \theta_1, \theta_2 : \theta_1 \oplus_\rho \theta_2 = \theta \textit{ and } (\theta_1, J) \models p \textit{ and } (\theta_2, J) \models p' \\ (\theta, J) \models c?p & \textit{when} & \exists \theta' : \theta = c?\theta' \textit{ and } (\theta', J) \models p \end{array}$$

*The other operators are as usual (cf. definition 6.4.1).*

The operator $+$ is used for addition of predicates, the operator $\rho\cdot$ performs scaling. A weighted sum $\oplus_\rho$ combines scaling and addition. Finally $c?p$ gives conditional probabilities without normalizing. A state satisfies the predicate $\rho \cdot p$ if it is a scaled version of a state satisfying $p$. A state $\theta$ satisfies the predicate $p + p'$ if it can be split into parts satisfying $p$ and $p'$. The operators $\oplus_\rho$ and $c?$ are similar.

Two predicates which specify the same property are called *equivalent*. Formally predicates $p$ and $q$ are called equivalent if for all states $\theta$ in $\Theta$ and interpretations $J$ in $\mathcal{J}$ we have that $(\theta, J) \models p$ exactly when $(\theta, J) \models q$.

As with deterministic predicates, substitution is denoted by $[e/x]$. As program variables can only occur within the deterministic predicate in the $\mathbb{P}(dp)$ construct, only the deterministic predicates that occur in a probabilistic predicate are affected by the substitution.

**Example 6.4.10** *The predicate $\forall i : (i \leq 0) \vee (\mathbb{P}(\mathrm{x} = i) = \frac{1}{2}^i)$, states that the variable x is geometrically distributed. In subsection 6.6.1 a program that results in a variable having a geometric distribution will be discussed.*

*The predicate $(\mathbb{P}(\mathrm{x} = \mathtt{1}) = \frac{1}{2}) + (\mathbb{P}(\mathrm{x} = \mathtt{1}) = \frac{1}{2})$ is equivalent with the predicate $\mathbb{P}(\mathrm{x} = \mathtt{1}) = \mathtt{1}$, but addition is not always so straightforward: The predicate $(\mathbb{P}(\mathrm{x} = \mathtt{1}) = \frac{1}{2}) + (\mathbb{P}(y = \mathtt{2}) = \frac{1}{2})$ means that the state can be split into two parts. One part where x is one with probability $\frac{1}{2}$ and one part where y is two with probability $\frac{1}{2}$. This predicate is satisfied by $\frac{1}{2}\langle \mathrm{x} = 1, y = 1\rangle + \frac{1}{2}\langle \mathrm{x} = 2, y = 2\rangle$ but also by $1\langle \mathrm{x} = 1, y = 2\rangle = \frac{1}{2}\langle \mathrm{x} = 1, y = 2\rangle + \frac{1}{2}\langle \mathrm{x} = 1, y = 2\rangle$. The state $\frac{1}{2}\langle \mathrm{x} = 1, y = 2\rangle + \frac{1}{2}\langle \mathrm{x} = 3, y = 3\rangle$ does not satisfy the predicate.*

When reasoning about probabilistic predicates, caution is advised. Some equivalences which may seem true at first sight do not hold. The most important of these is that in general $p \oplus_\rho p$ does not imply $p$. Take for example $\mathbb{P}(x = 1) = 1 \vee \mathbb{P}(x = 2) = 1$ for $p$. Then the state $\frac{1}{2}\langle x = 1\rangle + \frac{1}{2}\langle x = 2\rangle$ satisfies the predicate $p \oplus_{\frac{1}{2}} p$ but not the predicate $p$. This does not only occur for predicates of the form $q \vee q'$. More complicated examples involving the other operators can also be constructed.

That $p \oplus_\rho p$ is not equivalent with $p$ is not a shortcoming of the predicates, it is a general phenomenon when mixing nondeterminism and probability (see e.g. chapter 4, [109]). A predicate has an intrinsic nondeterminism, as it describes a set of states which satisfy a property. In the combination $p \oplus_\rho p$ selection of an element from the set of states that satisfy $p$ acts as a nondeterministic choice. Different elements can be selected for the left and right side, possibly resulting in a combination which no longer satisfies $p$.

Using probabilistic predicates the Hoare triples as introduced for deterministic programs, can be extended to probabilistic programs. The Hoare triple $\{\, p \,\}\ s\ \{\, q \,\}$ indicates that $p$ is a precondition and $q$ is a postcondition for the probabilistic program $s$. The Hoare triple is said to hold, denoted by $\models \{\, p \,\}\ s\ \{\, q \,\}$, if the precondition $p$ guarantees that postcondition $q$ holds after execution of $s$:

$$\models \{\, p \,\}\ s\ \{\, q \,\} \quad \text{if} \quad \forall \theta \in \Theta, J \in \mathcal{J} : (\theta, J) \models p \Rightarrow (\mathcal{D}(s)(\theta), J) \models q.$$

For example $\models \{\, p \,\}\ \texttt{skip}\ \{\, p \,\}$ and $\models \{\, \mathbb{P}(x = 0) = 1 \,\}\ \texttt{x := x + 1}\ \{\, \mathbb{P}(x = 1) = 1 \,\}$. Also, we have $\models \{\, \texttt{i} = 5 \,\}\ s\ \{\, \texttt{i} = 5 \,\}$ for any program $s$, as `i` is an integer valued variable remaining unaffected by any program $s$.

To prove the validity of Hoare triples, a proof system called *pH* is introduced. The proof system consists of the axioms and rules as given below.

$$\{\, p \,\}\ \texttt{skip}\ \{\, p \,\} \quad \text{(Skip)} \qquad \frac{\{\, c?p \,\}\ s\ \{\, q \,\} \quad \{\, \neg c?p \,\}\ s'\ \{\, q' \,\}}{\{\, p \,\}\ \texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{fi}\ \{\, q + q' \,\}}\ \text{(If)}$$

$$\{\, p[e/x] \,\}\ x\ \texttt{:=}\ e\ \{\, p \,\} \quad \text{(Assign)} \qquad \frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \{\, p \,\}\ s'\ \{\, q' \,\}}{\{\, p \,\}\ s \oplus_\rho s'\ \{\, q \oplus_\rho q' \,\}}\ \text{(Prob)}$$

$$\frac{\{\, p \,\}\ s\ \{\, p' \,\} \quad \{\, p' \,\}\ s'\ \{\, q \,\}}{\{\, p \,\}\ s\ ;\ s'\ \{\, q \,\}}\ \text{(Seq)} \qquad \frac{p' \Rightarrow p \quad \{\, p \,\}\ s\ \{\, q \,\} \quad q \Rightarrow q'}{\{\, p' \,\}\ s\ \{\, q' \,\}}\ \text{(Cons)}$$

$$\frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \{\, p' \,\}\ s\ \{\, q \,\}}{\{\, p \vee p' \,\}\ s\ \{\, q \,\}}\ \text{(Or)} \qquad \frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \texttt{j}\ \text{not free in}\ q}{\{\, \exists j : p \,\}\ s\ \{\, q \,\}}\ \text{(Exists)}$$

$$\frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \{\, p \,\}\ s\ \{\, q' \,\}}{\{\, p \,\}\ s\ \{\, q \wedge q' \,\}}\ \text{(And)} \qquad \frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \texttt{j}\ \text{not free in}\ p}{\{\, p \,\}\ s\ \{\, \forall j : q \,\}}\ \text{(Forall)}$$

$$\frac{\{\,p\,\}\ s\ \{\,q\,\}}{\{\,\rho\cdot p\,\}\ s\ \{\,\rho\cdot q\,\}}\ (\text{Lin}\ \cdot) \qquad \frac{\{\,p\,\}\ s\ \{\,q\,\} \quad \{\,p'\,\}\ s\ \{\,q'\,\}}{\{\,p+p'\,\}\ s\ \{\,q+q'\,\}}\ (\text{Lin}\ +)$$

The rules (Skip), (Assign), (Seq) and (Cons) are as for standard Hoare logic but now dealing with probabilistic predicates. The rule (If) has changed and the rules (Prob), (Or), (And), (Exists), (Forall), (Lin +) and (Lin ·) are new.

For a predicate $p$ to hold after the execution of `skip`, it should hold before the execution since `skip` does nothing. The predicate $p$ holds after an assignment `x := e` exactly when $p$ with $e$ substituted for x holds before the assignment, as the effect of the assignment is exactly replacing x with the value of $e$. The rule (Seq) states that $p$ is a sufficient precondition for $q$ to hold after execution of $s\,\texttt{;}\,s'$ if there exists an intermediate predicate $p'$ which holds after the execution of $s$ and which implies that $q$ holds after the execution of $s'$. The rule (Cons) states that the precondition may be strengthened and the postcondition may be weakened. The premise $p \Rightarrow p'$ holds if for states $\theta$ and all interpretations $J$ for which $(\theta, J) \models p$ holds we also have $(\theta, J) \models p'$. For each pair of predicates $p, p'$ for which this condition holds $p \Rightarrow p'$ is assumed to be included as an axiom in the proof system *pH*. One would prefer to have a deduction system to obtain $p \Rightarrow p'$. Although some rules that can be included in such a deduction system are clear, a complete deduction system is not available and may not even exist. The completeness result presented in the next section, therefore is a 'relative completeness' result [62]. See section 6.8 for a further discussion of this issue.

The rule (Prob) states that the result of executing $s \oplus_\rho s'$ is obtained by combining the results obtained by executing $s$ and $s'$ with the appropriate probabilities. The necessity for the (Or), (And), (Exists), (Forall) and (Lin) rules becomes clear when one recalls that $p \oplus_\rho p$ does not imply $p$. Proving correctness of e.g. $\{\,p \vee q\,\}\ \texttt{skip} \oplus_\rho \texttt{skip}\ \{\,p \vee q\,\}$ is, in general, not possible without the (Or)-rule. Similar examples show the need for the other rules. The soundness of these rules is intuitively clear as they closely resemble the natural deduction rules for $\vee$ and $\exists$ elimination and $\wedge$ and $\forall$ introduction.

The rule (If) has changed with respect to the (If) rule of standard Hoare logic. In a probabilistic state the value of the boolean condition $c$ may not be determined. Therefore, the probabilistic state is split into two parts, a part in which $c$ is true and a part in which $c$ is false. After splitting the state, the effect of the corresponding program, either $s$ or $s'$, can be found after which the parts are recombined using the + operator.

A Hoare triple $\{\,p\,\}\ s\ \{\,q\,\}$ is said to be *deducible* from the system *pH*, denoted by

$$\vdash\ \{\,p\,\}\ s\ \{\,q\,\}$$

if there exists a proof tree for $\{\,p\,\}\ s\ \{\,q\,\}$ in *pH*.

**Example 6.4.11** *The picture below gives an example of a proof tree in the system pH. The proof tree shows how the Hoare triple* $\{\,\mathbb{P}(\mathrm{x}=1)=1\,\}\ (\texttt{x := x + 1}) \oplus_{\frac{1}{2}} (\texttt{x := x + 2})\ \{\,\mathbb{P}(\mathrm{x}=2)=\frac{1}{2} \wedge \mathbb{P}(\mathrm{x}=3)=\frac{1}{2}\,\}$ *can be deduced using the rules in pH. For several deterministic predicates dp the shorthand* $[dp]$ *is used for the predicate* $\mathbb{P}(dp)=1$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\{\,[x+1=2]\,\}\ \mathtt{x := x + 1}\ \{\,[x=2]\,\}}\,\textit{(Assign)}}{\{\,[x=1]\,\}\ \mathtt{x := x + 1}\ \{\,[x=2]\,\}}\,\textit{(Cons)} \qquad \dfrac{\dfrac{}{\{\,[x+2=3]\,\}\ \mathtt{x := x + 2}\ \{\,[x=3]\,\}}\,\textit{(Assign)}}{\{\,[x=1]\,\}\ \mathtt{x := x + 2}\ \{\,[x=3]\,\}}\,\textit{(Cons)}}{\{\,[x=1]\,\}\ (\mathtt{x := x + 1}) \oplus_{\frac{1}{2}} (\mathtt{x := x + 2})\ \{\,[x=2] \oplus_{\frac{1}{2}} [x=3]\,\}}\,\textit{(Prob)}}{\{\,[x=1]\,\}\ (\mathtt{x := x + 1}) \oplus_{\frac{1}{2}} (\mathtt{x := x + 2})\ \{\,\mathbb{P}(x=2)=\frac{1}{2} \wedge \mathbb{P}(x=3)=\frac{1}{2}\,\}}\,\textit{(Cons)}$$

As a basic result we have soundness of the proof system *pH*.

**Theorem 6.4.12** *The proof system pH is sound, i.e. for all predicates $p$ and $q$ and programs $s$, $\vdash\ \{\,p\,\}\ s\ \{\,q\,\}$ implies $\models\ \{\,p\,\}\ s\ \{\,q\,\}$.*

**Proof** It is sufficient to show that if $(\theta, J) \models p$ and $\vdash\ \{\,p\,\}\ s\ \{\,q\,\}$ then $(\mathcal{D}(s)(\theta), J) \models q$, for all predicates $p$, $q$, states $\theta$, and interpretations $J$. This is shown by induction on the depth of the proof tree for $\{\,p\,\}\ s\ \{\,q\,\}$, by looking at the last rule used. The proof follows the standard approach. Only a few typical cases are given.

- As in non-probabilistic Hoare logic one can show that $(\sigma[\mathcal{V}(e)(\sigma)/x], I) \models dp$ exactly when $(\sigma, I) \models dp[e/x]$. By induction on the structure of the probabilistic predicate $p$ this extends to $(\theta[\mathcal{V}(e)/x], J) \models p$ exactly when $(\theta, J) \models p[e/x]$. Soundness of the (Assign) rule follows directly.

- If rule (Prob) is used to deduce $\vdash\ \{\,p\,\}\ s \oplus_\rho s'\ \{\,q \oplus_\rho q'\,\}$ from $\vdash\ \{\,p\,\}\ s\ \{\,q\,\}$ and $\vdash\ \{\,p\,\}\ s'\ \{\,q'\,\}$ then by induction $\models\ \{\,p\,\}\ s\ \{\,q\,\}$ and $\models\ \{\,p\,\}\ s'\ \{\,q'\,\}$. This means that if $(\theta, J) \models p$ then $(\mathcal{D}(s)(\theta), J) \models q$ and $(\mathcal{D}(s')(\theta), J) \models q'$. But then $(\mathcal{D}(s \oplus_\rho s')(\theta), J) = (\mathcal{D}(s)(\theta) \oplus_\rho \mathcal{D}(s')(\theta), J) \models q \oplus_\rho q'$.

## 6.5 Weakest preconditions and completeness

Theorem 6.4.12 shows that the proof system *pH* is sound. This means that any Hoare triple deduced with the proof system is valid. The next question is whether the proof system is complete: Can any Hoare triple that is valid be deduced with the proof system? In this section it is shown that all valid Hoare triples with a slight restriction on the postcondition can be deduced with the proof system.

Note that the language $\mathcal{L}_{\text{pif}}$ does not contain constructs for recursion or iteration. In a setting without probability the completeness proof is usually straightforward in absence of these constructs. The addition of probability, however, complicates the completeness proof even without these constructs.

Completeness of the proof system is shown by using weakest preconditions. For a postcondition $q$ and a program $s$ the weakest precondition $p$ that yields a valid Hoare triple $\{\,p\,\}\ s\ \{\,q\,\}$ is found. Next it is shown that the Hoare triple $\{\,p\,\}\ s\ \{\,q\,\}$ can be deduced in the proof system. Using the rule of consequence (Cons) this gives that any valid Hoare triple with $q$ as a postcondition can be deduced.

Before finding the weakest precondition, the form of the postconditions is restricted in two ways. The first restriction does not influence the expressiveness of the predicates.

For the second restriction this is not clear. The restricted probabilistic predicates used in the remainder of this section are defined by

$$\begin{array}{rcl} p & ::= & \texttt{true} \mid \texttt{false} \mid c \mid \mathbb{P}(dp) \texttt{=} r \mid e_r \texttt{=} e_r \mid e_r < e_r \mid \ldots \mid p \wedge p \mid p \vee p \\ & & \mid \forall j : p \mid \exists j : p \mid \rho \cdot p \mid p + p \mid p \oplus_\rho p \\ e_r & ::= & \rho \mid r \mid e_r + e_r \mid e_r - e_r \mid e_r * e_r \mid e_r / e_r \mid e_r^e \end{array}$$

where $c$ is a boolean condition on integer valued variables in $BC\langle IVar\rangle$ and $e$ is an (integer) expression on integer valued variables in $Exp\langle IVar\rangle$.

Compared to the predicates used in the previous section, this definition restricts the real expressions by disallowing the use of $\mathbb{P}(dp)$ as part of a real expression. Instead $\mathbb{P}(dp)$ may only be used in the construct $\mathbb{P}(dp) \texttt{=} r$. The predicates are further restricted by disallowing the use of the operator $c?p$.

The restriction of the real expression implies that the comparison of unrestricted real expressions using $\mathbb{P}(dp)$ is no longer possible in the restricted probabilistic predicates. The construct $\mathbb{P}(dp)$ may only be used in the form $\mathbb{P}(dp) = r$ where $r$ is a real valued logical variable. This restriction does not apply for real expressions not containing the construct $\mathbb{P}(dp)$. It is easy to adapt a predicate so that it satisfies this restriction by introducing additional real valued variables (see the example below). The restriction of the real expressions, therefore, does not influence the expressiveness of the probabilistic predicates.

**Example 6.5.1** *The predicate* $(\frac{1}{2}^i + \mathbb{P}(x = 1)) > \frac{1}{2}$ *is not of the restricted form used in this section, however the equivalent predicate* $\exists r : \mathbb{P}(x = 1) = r \wedge (\frac{1}{2}^i + r) > \frac{1}{2}$ *is.*

The second restriction that applies to the predicates allowed as postconditions is that the operation $c?$ may not be used. It is not clear whether for every predicate of the form $c?p$, an equivalent predicate without use of the $c?$ operator exists. The omission of the $c?$ operator may restrict the postconditions which can be checked. The operator $c?$ is more an auxiliary operator for use in the proof system than something that has a clear use in specification of properties. It is therefore not considered a severe restriction that the completeness result obtained in this section does not consider specifications of the postcondition that use the operator $c?$.

The restrictions on the probabilistic predicates make it possible to find the weakest precondition. Formally a weakest precondition is defined as follows: A predicate $p$ is a weakest precondition for postcondition $q$ after program $s$ when

1. $p$ is a sufficient precondition, i.e. for all states $\theta$ and interpretations $J$ $(\theta, J) \models p$ implies that $(\mathcal{D}(s)(\theta), J) \models q$

   and

2. if $p'$ satisfies condition 1 then $p'$ implies $p$.

If a predicate $p$ can be found which for all interpretations $J$ satisfies $(\theta, J) \models p \iff (\mathcal{D}(s)(\theta), J) \models q$, then the predicate $p$ captures exactly those states which will satisfy $q$ after executing the program $s$. Clearly $p$ is the weakest precondition and will stay the weakest precondition even if additional predicates are added. (In general it may be possible to add a weaker predicate that is still a precondition.) In particular if a predicate $p$ in the set of restricted predicates used in this section, satisfies the property above then it is also the weakest precondition among all predicates in *Pred*.

Below a predicate $\wp(s)(q)$ is defined for each program $s$ and predicate $q$. The proof system is shown to be general enough to be able to deduce the Hoare triple $\{\, \wp(s)(q) \,\}\; s \;\{\, q \,\}$. By soundness of the proof system this also means that $\{\, \wp(s)(q) \,\}\; s \;\{\, q \,\}$ is a valid Hoare triple. Next it is shown that $\wp(s)(q)$ satisfies the stronger property mentioned above: $\theta \models \wp(s)(q) \iff \mathcal{D}(s)(\theta) \models q$.

**Definition 6.5.2** *For a program $s$ and predicate $q$, the predicate $\wp(s)(q)$ is defined by induction on the structure of the program $s$ and the predicate $q$ as follows*

$$\begin{array}{rcl}
\wp(\texttt{skip})(q) & = & q \\
\wp(x \;\texttt{:=}\; e)(q) & = & q[e/x] \\
\wp(s \;\texttt{;}\; s')(q) & = & \wp(s)(\wp(s')(q))
\end{array}$$

$$\begin{array}{rcl}
\wp(s \oplus_\rho s')(q \;\texttt{op}\; q') & = & \wp(s \oplus_\rho s')(q) \;\texttt{op}\; \wp(s \oplus_\rho s')(q') \quad \texttt{op} \in \{\vee, \wedge, +, \oplus_{\rho'}\} \\
\wp(s \oplus_\rho s')(\texttt{op}\;\; q) & = & \texttt{op}\; \wp(s \oplus_\rho s')(q) \quad \texttt{op} \in \{\exists j, \forall j, \rho\,\cdot\,\} \\
\wp(s \oplus_\rho s')(\mathbb{P}(dp) = r) & = & \exists r_1, r_2 : \big(\rho * r_1 + (1-\rho) * r_2 = r\big) \wedge \\
 & & \qquad \wp(s)(\mathbb{P}(dp) = r_1) \wedge \wp(s')(\mathbb{P}(dp) = r_2)\big) \\
\wp(s \oplus_\rho s')(q) & = & q \quad \textit{for all other predicates } q
\end{array}$$

$$\begin{array}{l}
\wp(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi})(q \;\texttt{op}\; q') \;=\; \wp(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi})(q) \;\texttt{op} \\
\qquad\qquad\qquad\qquad \wp(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi})(q') \quad \texttt{op} \in \{\vee, \wedge, +, \oplus_{\rho'}\} \\
\wp(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi})(\texttt{op}\;\; q) \;=\; \texttt{op}\; \wp(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi})(q) \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \texttt{op} \in \{\exists j, \forall j, \rho\,\cdot\,\} \\
\wp(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi})(\mathbb{P}(dp) = r) \;=\; \exists r_1, r_2 : \big((r_1 + r_2 = r) \wedge \\
\qquad ((\wp(s)(\mathbb{P}(dp) = r_1) \wedge \mathbb{P}(\neg c) = 0) + (\wp(s')(\mathbb{P}(dp) = r_2) \wedge \mathbb{P}(c) = 0))\big) \\
\wp(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi})(q) \;=\; q \quad \textit{for all other predicates } q
\end{array}$$

The predicate $\wp(s)(q)$ is meant to be the weakest precondition for postcondition $q$ after program $s$. For `skip`, the weakest precondition is the postcondition itself. The weakest precondition for assignment can be found by substituting the expression for the variable in the postcondition. To find the weakest precondition for a sequential composition $s \,;\, s'$, the weakest precondition for $s'$ is found and used as postcondition to be satisfied after $s$.

The weakest precondition for the probabilistic choice is harder to find. The rule (Prob) does not give a clear precondition for every postcondition, only for postconditions which are of the form $q \oplus_\rho q'$ and even for this postcondition, $\wp(s)(q) \wedge \wp(s')(q')$ is a precondition for $s \oplus_\rho s'$, but in general not the weakest precondition.

**Example 6.5.3** *The weakest precondition for the predicate $q = (\mathbb{P}(y = 1) = 1)$ after the program $s = \texttt{x := 1}$ is given by $\wp(s)(q) = (\mathbb{P}(y = 1) = 1)$ as the execution of $s$ does not change the value of the variable $y$. Similarly the weakest precondition for the predicate $q' = (\mathbb{P}(y = 2) = 1)$ after the program $s' = \texttt{x := 2}$ is $q'$ itself, $\wp(s')(q') = (\mathbb{P}(y = 2) = 1)$.*

*The weakest precondition for the predicate $q \oplus_{\frac{1}{2}} q'$ and the program $s \oplus_{\frac{1}{2}} s'$ is $q \oplus_{\frac{1}{2}} q'$ which is weaker than $\wp(s)(q) \wedge \wp(s')(q') = (\mathbb{P}(y = 1) = 1) \wedge (\mathbb{P}(y = 2) = 1) = \texttt{false}$*

A different approach is needed. Instead of trying to give the weakest precondition for general postconditions directly, the weakest precondition is built by combining the weakest preconditions for the parts of the predicate.

For a simple predicate of the form $\mathbb{P}(dp)$ = $r$ the weakest precondition can be found. If the chance that $dp$ holds must be $\mathtt{r}$ after executing $s \oplus_\rho s'$, then combining the chance that $dp$ holds after executing $s$, say $\mathtt{r}_1$, and the chance that $dp$ holds after executing $s'$, say $\mathtt{r}_2$, should yield exactly $\mathtt{r}$, thus $\mathtt{r} = \rho * \mathtt{r}_1 + (1-\rho) * \mathtt{r}_2$. Basic predicates that do not use the $\mathbb{P}(dp)$ construct are not effected by the execution of a program; the weakest precondition is the predicate itself. The weakest precondition for combined predicates can be found by combining the weakest preconditions for the basic predicates.

To find the weakest precondition for postcondition $\mathbb{P}(dp)$ = $r$ after a conditional choice, one can reason as follows: To execute $\mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s'\ \mathtt{fi}$ the state is split into a part satisfying $c$ and a part satisfying $\neg c$. That a state can be split into two parts can be described in a predicate by using the operator $+$. A predicate $(p \wedge \mathbb{P}(\neg c) = 0) + (q \wedge \mathbb{P}(c) = 0)$ holds in a state $\theta$ exactly when $p$ holds in $c?\theta$ and $q$ holds in $\neg c?\theta$. Using this similar reasoning as for the probabilistic choice leads to the weakest precondition as given in definition 6.5.2.

**Example 6.5.4** *For $r_1$ and $r_2$ real valued variables in RVar we have that $\wp(\mathtt{x} := \mathtt{x}+1)(\mathbb{P}(\mathtt{x}=3) = r_1)$ is equal to $\mathbb{P}(\mathtt{x}+1=3) = r_1$ which is equivalent with $\mathbb{P}(\mathtt{x}=2) = r_1$ and similarly that $\wp(\mathtt{x} := \mathtt{x}+2)(\mathbb{P}(\mathtt{x}=3) = r_2)$ is equivalent to $\mathbb{P}(\mathtt{x}=1) = r_2$. Using this gives*

$$\begin{aligned}&\wp(\mathtt{x} := \mathtt{x}+1 \oplus_\rho \mathtt{x} := \mathtt{x}+2)(\mathbb{P}(\mathtt{x}=3) = \tfrac{1}{2}) \quad = \\ &\qquad \exists r_1, r_2 : (\rho * r_1 + (1-\rho) * r_2 = \tfrac{1}{2}) \wedge \wp(\mathtt{x} := \mathtt{x}+1)(\mathbb{P}(\mathtt{x}=3) = r_1) \wedge \\ &\qquad\qquad \wp(\mathtt{x} := \mathtt{x}+2)(\mathbb{P}(\mathtt{x}=3) = r_2) \quad = \\ &\qquad \exists r_1, r_2 : (\rho * r_1 + (1-\rho) * r_2 = \tfrac{1}{2}) \wedge \mathbb{P}(\mathtt{x}=2) = r_1 \wedge \mathbb{P}(\mathtt{x}=1) = r_2\end{aligned}$$

*This predicate can be simplified to $\rho * \mathbb{P}(\mathtt{x}=2) + (1-\rho) * \mathbb{P}(\mathtt{x}=1) = \frac{1}{2}$. For the simplified predicate the restrictions imposed in this section are no longer satisfied so if further calculation of weakest preconditions is required, as e.g. for the sequential composition $s\ ;\ (\mathtt{x} := \mathtt{x}+1 \oplus_\rho \mathtt{x} := \mathtt{x}+2)$, the simplification cannot be used.*

The first part in proving that $\wp$, as given by definition 6.5.2, does indeed give the weakest precondition is to show that the postcondition $q$ can be deduced from $\wp(s)(q)$ by the proof system *pH*. By soundness of the proof system (theorem 6.4.12) this gives that $\wp(s)(q)$ is a sufficient precondition.

**Lemma 6.5.5** *For any program $s$ and predicate $q$ (of the restricted form used in this section), $\vdash \{\, \wp(s)(q) \,\}\ s\ \{\, q \,\}$.*

**Proof** As with the definition of $\wp$ this proof uses induction on the structure of the program $s$ and subinduction on the structure of the predicate $p$ if the program is a probabilistic choice or conditional choice. Each case uses the corresponding rule from the proof system *pH*. Only the case for probabilistic choice with the predicate $\mathbb{P}(dp) = r$ is given.

- Using the induction assumption $\vdash \{\, \wp(s)(p) \,\}\ s\ \{\, p \,\}$ for $p = (\mathbb{P}(dp) = r_1)$ and $\vdash \{\, \wp(s')(p') \,\}\ s'\ \{\, p' \,\}$ for $p' = (\mathbb{P}(dp) = r_2)$ and rules (Cons) and (Prob) gives that $\vdash \{\, \wp(s)(\mathbb{P}(dp) = r_1) \wedge \wp(s')(\mathbb{P}(dp) = r_2) \,\}\ s \oplus_\rho s'\ \{\, (\mathbb{P}(dp) = r_1) \oplus_\rho (\mathbb{P}(dp) = r_2) \,\}$. The postcondition implies $\mathbb{P}(dp) = \rho * r_1 + (1-\rho) * r_2$.

As $c_r \oplus_\rho c_r$ is equivalent with $c_r$ for any condition that does not use the $\mathbb{P}(dp)$ construct, we have $\vdash \{ r = \rho * r_1 + (1-\rho) * r_2 \} \; s \oplus_\rho s' \; \{ r = \rho * r_1 + (1-\rho) * r_2 \}$. Using rules (And) and (Cons) gives $\vdash \{ \wp(s)(\mathbb{P}(dp) = r_1) \wedge \wp(s')(\mathbb{P}(dp) = r_2) \wedge r = \rho * r_1 + (1-\rho) * r_2 \} \; s \oplus_\rho s' \; \{ \mathbb{P}(dp) = r \}$. Applying rule (Exists) for $r_1$ and $r_2$ yields the desired result. □

Having shown that $\wp$ gives a sufficient precondition, it remains to be shown that $\wp$ gives the weakest precondition. To show that $\wp$ yields the weakest precondition the following stronger property is shown: For any program $s$ and predicate $q$, $(\mathcal{D}(s)(\theta), J) \models q \iff (\theta, J) \models \wp(s)(q)$. The implication $(\theta, J) \models \wp(s)(q) \Rightarrow (\mathcal{D}(s)(\theta), J) \models q$ is direct from the property $\models \{ \wp(s)(q) \} \; s \; \{ q \}$ shown above. The following lemma shows the reverse implication.

**Lemma 6.5.6** *For any program $s \in \mathcal{L}_{\text{pif}}$ and predicate $q$ of the restricted form used in this section, $(\mathcal{D}(s)(\theta), J) \models q$ implies $(\theta, J) \models \wp(s)(q)$.*

**Proof** The proof again uses induction on the structure of $s$ and subinduction on the structure of $q$ if $s$ is a probabilistic choice or conditional choice. Only the case for probabilistic choice with the predicate $\mathbb{P}(dp) = r$ is given.

- Assume that $(\mathcal{D}(s \oplus_\rho s')(\theta), J) \models \mathbb{P}(dp) = r$. By writing out the definition, it is easy to check that this is exactly the case when there exist real numbers $\rho_1$ and $\rho_2$ such that: $\rho\rho_1 + (1-\rho)\rho_2$ is equal to $J(r)$, $(\mathcal{D}(s)(\theta), J[r_1/\rho_1]) \models \mathbb{P}(dp) = r_1$ and $(\mathcal{D}(s')(\theta), J[r_2/\rho_2]) \models \mathbb{P}(dp) = r_2$. By induction this gives that $(\theta, J[r_1/\rho_1]) \models \wp(s)(\mathbb{P}(dp) = r_1)$ and $(\theta, J[r_2/\rho_2]) \models \wp(s')(\mathbb{P}(dp) = r_2)$. Combining this with $\rho\rho_1 + (1-\rho)\rho_2 = J(r)$ immediately gives $(\theta, J) \models \exists r_1, r_2 : \rho * r_1 + (1-\rho) * r_2 = r \wedge \wp(s)(\mathbb{P}(dp) = r_1) \wedge \wp(s')(\mathbb{P}(dp) = r_2)$, i.e. $(\theta, J) \models \wp(s \oplus_\rho s')(\mathbb{P}(dp) = r)$. □

This lemma together with lemma 6.5.5 above gives all the ingredients needed for the completeness result.

**Theorem 6.5.7** *Let $s$ be any program in $\mathcal{L}_{\text{pif}}$, $p$ any predicate and $q$ a predicate that does not contain the $c?$ operator then*

$$\models \{ p \} \; s \; \{ q \} \;\; \textit{if and only if} \;\; \vdash \{ p \} \; s \; \{ q \}$$

**Proof** Soundness of the proof system (theorem 6.4.12) already gives the 'if' part. For the 'only if' part assume that $\models \{ p \} \; s \; \{ q \}$. For a predicate $q$ which does not contain any $c?$ operator, an equivalent predicate $q'$ of the restricted form used in this section can be found as in e.g. example 6.5.1. Lemma 6.5.5 shows that for such a predicate $q'$ the property $\vdash \{ \wp(s)(q') \} \; s \; \{ q' \}$ holds. It is therefore sufficient to show that $p$ implies $\wp(s)(q')$ because then $\vdash \{ p \} \; s \; \{ q \}$ follows by rule (Cons).

For any state $\theta$ satisfying $p$, i.e. $(\theta, J) \models p$, $\mathcal{D}(s)(\theta)$ satisfies $q'$ because $\models \{ p \} \; s \; \{ q \}$ and because $q'$ is equivalent with $q$. But then by lemma 6.5.6, $(\theta, J) \models \wp(s)(q')$ holds. So any state satisfying $p$ also satisfies $\wp(s)(q')$, i.e. $p$ implies $\wp(s)(q')$. □

## 6.6 Extending $\mathcal{L}_{\mathrm{pif}}$: Adding iteration

The language $\mathcal{L}_{\mathrm{pif}}$, discussed above, lacks a construct for iteration. In this section we extend $\mathcal{L}_{\mathrm{pif}}$, obtaining a new language $\mathcal{L}_{\mathrm{pw}}$ in which a `while`-construct is present. We add a proof rule for `while` to our Hoare logic and establish its soundness. In contrast to the setting of the language $\mathcal{L}_{\mathrm{pif}}$, for which a completeness result with minor restrictions has been obtained, the completeness of the Hoare logic for $\mathcal{L}_{\mathrm{pw}}$ is an open issue. In subsection 6.6.1 the usage of the proof rule is illustrated in the context of the example of an erratic sequence summer and of a geometrically distributed program variable.

The syntax of the language $\mathcal{L}_{\mathrm{pw}}$ is a straightforward extension of the syntax of the language $\mathcal{L}_{\mathrm{pif}}$. The constructs remain the same except for the addition of a clause for the `while` construct.

**Definition 6.6.1** *Let PVar be a set of program variables and let* x *range over PVar. The programs in* $\mathcal{L}_{\mathrm{pw}}$*, ranged over by* $s$*, are given by:*

$$s ::= \mathtt{skip} \mid x \;\mathtt{:=}\; e \mid s \;\mathtt{;}\; s \mid s \oplus_\rho s \mid \mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s\ \mathtt{fi} \mid \mathtt{while}\ c\ \mathtt{do}\ s\ \mathtt{od}$$

*with* $e \in Exp\langle PVar\rangle$, $c \in BC\langle PVar\rangle$ *and* $\rho \in (0,1)$.

The program `while` $c$ `do` $s$ `od` is interpreted as "repeatedly execute $s$ as long as the condition $c$ holds". The `while` construct introduces the possibility of arbitrarily many repetitions as well as the possibility of non-termination; the program `while true do` $s$ `od`, for example, will never finish.

The denotational semantics $\mathcal{D}$ for $\mathcal{L}_{\mathrm{pw}}$ gives, for each program $s$, and state $\theta$, the state $\mathcal{D}(s)(\theta)$ resulting from executing $s$ starting in state $\theta$. The denotational semantics is also an extension of the denotational semantics for the language $\mathcal{L}_{\mathrm{pif}}$ given in section 6.3. A clause is added for programs built with `while`. This clause uses a least fixed point construction. The cpo structure on $\Theta$ (See section 6.2) guarantees that this least fixed point exists.

**Definition 6.6.2**

*(a) For a condition* $c$ *in* $BC\langle PVar\rangle$ *and a program* $s$ *in* $\mathcal{L}_{\mathrm{pw}}$*, the higher-order operator* $\Psi_{\langle c,s\rangle} : (\Theta \to \Theta) \to (\Theta \to \Theta)$ *is given by*

$$\Psi_{\langle c,s\rangle}(\psi)(\theta) \quad = \quad \psi(\mathcal{D}(s)(c?\theta)) + \neg c?\theta$$

*This operator is monotone and therefore has a least fixed point* $\mathit{fix}(\Psi_{\langle c,s\rangle})$.

*(b) The denotational semantics* $\mathcal{D} : \mathcal{L}_{\mathrm{pw}} \to (\Theta \to \Theta)$ *is given by*

$$\begin{array}{rcl}
\mathcal{D}(\mathtt{skip})(\theta) & = & \theta \\
\mathcal{D}(\mathrm{x} \;\mathtt{:=}\; e)(\theta) & = & \theta[\mathcal{V}(e)/\mathrm{x}] \\
\mathcal{D}(s \;\mathtt{;}\; s')(\theta) & = & \mathcal{D}(s')(\mathcal{D}(s)(\theta)) \\
\mathcal{D}(s \oplus_\rho s')(\theta) & = & \mathcal{D}(s)(\theta) \oplus_\rho \mathcal{D}(s')(\theta) \\
\mathcal{D}(\mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s'\ \mathtt{fi})(\theta) & = & \mathcal{D}(s)(c?\theta) + \mathcal{D}(s')(\neg c?\theta) \\
\mathcal{D}(\mathtt{while}\ c\ \mathtt{do}\ s\ \mathtt{od})(\theta) & = & \big(\mathit{fix}(\Psi_{\langle c,s\rangle})\big)(\theta)
\end{array}$$

For a while program `while` $c$ `do` $s$ `od`, one would like to use the familiar unfolding to `if` $c$ `then` $s$ ; `while` $c$ `do` $s$ `od else skip fi`. In the present setting this cannot be done directly, as the second program is more complex than the first. Instead, the fact that $\mathcal{D}(\texttt{while } c \texttt{ do } s \texttt{ od})$ is a fixed point of the higher-order operator $\Psi_{\langle c,s\rangle}$ can be used

$$\begin{array}{rl} \multicolumn{2}{l}{\mathcal{D}(\texttt{while } c \texttt{ do } s \texttt{ od})(\theta)} \\ = & \Psi_{\langle c,s\rangle}(\mathcal{D}(\texttt{while } c \texttt{ do } s \texttt{ od}))(\theta) \\ = & \mathcal{D}(\texttt{while } c \texttt{ do } s \texttt{ od})(\mathcal{D}(s)(c?\theta)) + \neg c?\theta \\ = & \mathcal{D}(s\,;\texttt{while } c \texttt{ do } s \texttt{ od})(c?\theta) + \mathcal{D}(\texttt{skip})(\neg c?\theta) \\ = & \mathcal{D}(\texttt{if } c \texttt{ then } s\,;\texttt{while } c \texttt{ do } s \texttt{ od else skip fi})(\theta) \end{array}$$

The first equality uses the fact that $\mathcal{D}(\texttt{while } c \texttt{ do } s \texttt{ od})$ is a fixed point of $\Psi_{\langle c,s\rangle}$ and the second equality is direct from the definition of $\Psi_{\langle c,s\rangle}$. The last two equalities are direct from the definition of the semantics $\mathcal{D}$ for sequential composition, skip and conditional choice.

The total probability of $\mathcal{D}(\texttt{while } c \texttt{ do } s \texttt{ od})(\theta)$ may be less than that of $\theta$. The 'missing' probability is caused by non-termination; if a possible computation does not terminate, it does not contribute to the probabilities in the final state.

The semantics of the `while` construction is given as the least fixed point of the higher-order operator $\Psi_{\langle c,s\rangle}$. The set of probabilistic states $\Theta$ consists of the distributions over the set of deterministic states and as such it forms a cpo. Given this it is not difficult to check that $\Psi_{\langle c,s\rangle}$ indeed has a least fixed point. (One only needs to check monotonicity which is straightforward.) A more constructive description of the state $\mathcal{D}(\texttt{while } c \texttt{ do } s \texttt{ od})(\theta)$, however, is also useful. Below the least fixed point of $\Psi_{\langle c,s\rangle}$ is constructed by giving a chain which converges to this least fixed point $\mathit{fix}(\Psi_{\langle c,s\rangle})$.

**Definition 6.6.3** *For a program $s$ define $s^0 = \texttt{skip}$ and $s^{n+1} = s\ ;\ s^n$. The functions $\mathit{if}^n_{\langle c,s\rangle}$ and $L_{\langle c,s\rangle}$ from probabilistic states to probabilistic states are given by*

$$\begin{array}{rcl} \mathit{if}^n_{\langle c,s\rangle}(\theta) & = & \mathcal{D}((\texttt{if } c \texttt{ then } s \texttt{ else skip fi})^n)(\theta) \\ L_{\langle c,s\rangle}(\theta) & = & \lim\limits_{n\to\infty} \neg c?\mathit{if}^n_{\langle c,s\rangle}(\theta) \end{array}$$

The function $\mathit{if}^n_{\langle c,s\rangle}$ is merely a shorthand notation. The function $L_{\langle c,s\rangle}$ characterizes the least fixed point of $\Psi_{\langle c,s\rangle}$ and is thus equal to $\mathcal{D}(\texttt{while } c \texttt{ do } s \texttt{ od})$.

**Lemma 6.6.4** *The least fixed point of $\Psi_{\langle c,s\rangle}$ is given by $L_{\langle c,s\rangle}$.*

**Proof** Note that $\mathit{if}^n_{\langle c,s\rangle}(\neg c?\theta) = \neg c?\theta$ for any $n$ so also $L_{\langle c,s\rangle}(\neg c?\theta) = \neg c?\theta$. Using this straightforward calculation shows that $L_{\langle c,s\rangle}$ is a fixed point of $\Psi_{\langle c,s\rangle}$. Any fixed point, say $\xi$, not larger or equal to $L_{\langle c,s\rangle}$ would have to be smaller on a certain $\theta$, $\xi(\theta) < L_{\langle c,s\rangle}(\theta)$. But then $\xi(\theta) < \mathit{if}^n_{\langle c,s\rangle}(\theta)$ for some $n$. By using that $\xi$ is a fixed point and working out $\Psi^n_{\langle c,s\rangle}(\xi)(\theta)$ one gets: $\xi(\theta) = \Psi^n_{\langle c,s\rangle}(\xi)(\theta) = \mathit{if}^n_{\langle c,s\rangle}(\theta) + \xi(\ldots) \geq \mathit{if}^n_{\langle c,s\rangle}(\theta)$ which gives a contradiction.

Note that this lemma can also be obtained from the easily checked fact that $\Psi_{\langle c,s\rangle}$ is continuous ($\Psi_{\langle c,s\rangle}$ applied to the least upper bound of any chain $(\psi_i)_{i\in\mathbb{N}}$ is the same as the least upper bound of the sequence $(\Psi_{\langle c,s\rangle}(\psi_i))_{i\in\mathbb{N}})$ and applying more general results available for cpo's.

To reason about programs in $\mathcal{L}_{\text{pw}}$, an extension of the proof system *pH* given in section 6.4 is used. The semantics of $\mathcal{L}_{\text{pw}}$ is a consistent extension of the semantics of $\mathcal{L}_{\text{pif}}$. It is, therefore, not surprising that the existing rules of the proof system *pH* remain valid. To deal with programs containing the `while` construct, the following rule is added.

$$\frac{p \text{ invariant for } \langle c, s\rangle}{\{\, p \,\}\ \texttt{while}\ c\ \texttt{do}\ s\ \texttt{od}\ \{\, p \wedge \mathbb{P}(c) = 0 \,\}}\ \text{(While)}$$

The extended system, including the rule (While), is also referred to as *pH*. The rule (While) has the same form as the rule used in (non-probabilistic) Hoare logic, however the notion of invariant is more complicated. Simply replacing $p$ invariant for $\langle c, s\rangle$ by $\{\, p \wedge c \,\}\ s\ \{\, p \,\}$ does not work here.

To use the (While) rule, an invariant $p$ should be found. In non-probabilistic Hoare logic, a predicate $p$ is said to be an invariant if the Hoare triple $\{\, p \wedge c \,\}\ s\ \{\, p \,\}$ is valid or, equivalently, when $\{\, p \,\}$ `if` $c$ `then` $s$ `else skip fi` $\{\, p \,\}$ is valid. This condition is also included here: For an assertion $p$ to be an invariant, it should satisfy $\{\, p \,\}$ `if` $c$ `then` $s$ `else skip fi` $\{\, p \,\}$. As in Hoare logic, this condition is sufficient to obtain partial correctness. If the program $s$ is terminating and $\{\, p \,\}\ s\ \{\, q \,\}$ can be deduced from *pH*, then $\models\ \{\, p \,\}\ s\ \{\, q \,\}$ holds. A probabilistic program is terminating, if the program terminates for all possible outcomes of the probabilistic choices. Formally, a while loop is said to terminate for start state $\theta$ when $\mathcal{D}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od})(\theta) = \mathit{if}^{\,n}_{\langle c,s\rangle}(\theta)$ holds for some $n$. The loop is said to be terminating if it terminates for all states $\theta$. A program $s$ is said to be terminating if all while loops in $s$ are terminating.

Partial correctness, however, is not sufficient for probabilistic programs. Many probabilistic programs do not satisfy the termination condition, they may for instance only terminate with a certain probability. To deduce valid Hoare triples for programs that need not terminate, a form of unconditional correctness is required. This requires somehow adding termination conditions to the rules. To obtain this unconditional correctness the notion of invariant is strengthened by imposing the extra condition of so called $\langle c, s\rangle$-*closedness*. The main idea of $\langle c, s\rangle$-closedness for a predicate $p$ is that any sequence of states satisfying the predicate $p$ that could correspond to the states obtained from repeated iterations of the while loop must have a limit also satisfying $p$. Before this can be made precise, some auxiliary definitions are needed.

**Definition 6.6.5** *Assume some fixed interpretation $J$.*

*(a) For a predicate $p$ the $n$-step termination ratio, denoted by $r^n_{\langle c,s\rangle}$, is the minimum probability that, starting from a state satisfying $p$, the while loop "`while` $c$ `do` $s$ `od`" terminates within $n$ steps.*

$$\begin{aligned} r(\theta)^n_{\langle c,s\rangle} &= \textstyle\sum(\neg c ? \mathit{if}^{\,n}_{\langle c,s\rangle}(\theta))[\mathcal{S}] \\ r^n_{\langle c,s\rangle} &= \inf\{\, r(\theta)^n_{\langle c,s\rangle} \mid (\theta, J) \models p \,\} \end{aligned}$$

*(b) A sequence of states $(\theta_n)_{n\in\mathbb{N}}$ is called a $\langle c, s\rangle$-sequence within $p$ if $(\neg c?\theta_n)_{n\in\mathbb{N}}$ is an ascending chain and for each $n$ in $\mathbb{N}$ the state $\theta_n$ satisfies $p$ and $\sum(\neg c?\theta_n)[\mathcal{S}] \geq r^n_{\langle c,s\rangle}$ holds.*

*(c) A $\langle c, s\rangle$-sequence $(\theta_n)_{n\in\mathbb{N}}$ within $p$ is said to terminate within $p$ if the least upper bound of the sequence $(\neg c?\theta_n)_{n\in\mathbb{N}}$ satisfies $p$.*

Recall from the preliminaries of this chapter that $\sum\theta[\mathcal{S}]$ denotes the total probability of the probabilistic state $\theta$. For a given state $\theta$, the ratio $r(\theta)^n_{\langle c,s\rangle}$ expresses the probability that the while loop "`while` $c$ `do` $s$ `od`" terminates within $n$ steps. The state after $n$-iterations of the loop is $\mathit{if}^n_{\langle c,s\rangle}(\theta)$. The part of this state $\neg c?\mathit{if}^n_{\langle c,s\rangle}(\theta)$ that does not satisfy the loop condition $c$ has terminated. By taking the infimum over all states that satisfy the predicate $p$, the $n$-step termination ratio is obtained.

Consider a sequence of states that starts from a state satisfying the predicate $p$ and in which each following state is obtained by executing another iteration of the while loop "`while` $c$ `do` $s$ `od`". Clearly such a sequence must satisfy the following properties: In the first place the terminated part of the state, i.e. the part of the state not satisfying the condition $c$, does not change. Secondly, the total probability of this terminated part of the state must be at least the $n$-step termination ratio. Any sequence satisfying these two properties is referred to as a $\langle c, s\rangle$-sequence.

In the formulation of the main idea of the notion of $\langle c, s\rangle$-closedness given above definition 6.6.5 we referred to "sequences of states that could correspond to the states obtained from repeated iterations of the while loop". Using the observation above that any such a sequence must be a $\langle c, s\rangle$-sequence, we can strengthen this to "all $\langle c, s\rangle$-sequences".

**Definition 6.6.6**

*(a) A predicate $p$ is called $\langle c, s\rangle$-closed if each $\langle c, s\rangle$-sequence within $p$ terminates within $p$.*

*(b) A predicate $p$ is called an invariant for $\langle c, s\rangle$ when $p$ is $\langle c, s\rangle$-closed and $\{p\}$ `if` $c$ `then` $s$ `else skip fi` $\{p\}$.*

The notion of $\langle c, s\rangle$-closedness is meant to express that a sequence that can be obtained by repeated iterations of the loop `while` $c$ `do` $s$ `od` will have a least upper bound satisfying $p$. However, one does not want to have to find and check exactly these sequences. Needing to do this makes the checking of $\langle c, s\rangle$-closedness too difficult. Requiring arbitrary sequences to have a least upper bound satisfying $p$, however, is clearly to strong a property making it impossible to find useful invariants. As a compromise, we require this property only for $\langle c, s\rangle$-sequences. In this way, we obtain a notion of $\langle c, s\rangle$-closedness that can be shown to hold, gives a valid while rule (see theorem 6.6.7 below) and does not restrict the predicates that can be used as invariants too much.

Combining the definition of invariant with the rule (While) given above gives

$$\frac{\{p\}\ \texttt{if}\ c\ \texttt{then}\ s\ \texttt{else skip fi}\ \{p\} \qquad p \text{ is } \langle c,s\rangle\text{-closed}}{\{p\}\ \texttt{while}\ c\ \texttt{do}\ s\ \texttt{od}\ \{p \wedge \mathbb{P}(c) = 0\}}\ \text{(While)}$$

Note that, for a loop `while` $c$ `do` $s$ `od` that is terminating, every predicate $p$ automatically satisfies $\langle c,s\rangle$-closedness. Therefore, for a terminating program, there is no need to check any $\langle c,s\rangle$-closedness conditions. Two examples of programs using the while construct are given in subsection 6.6.1. The first program is terminating and no closedness conditions need to be checked. The second program shows a situation where the closedness condition is essential to guarantee soundness.

With addition of the rule (While) the proof system still remains sound, i.e. only valid Hoare triples can be deduced from *pH*.

**Theorem 6.6.7** *The proof system pH is sound, i.e. for all predicates p and q and programs s,* $\vdash \{\,p\,\}\ s\ \{\,q\,\}$ *implies* $\models \{\,p\,\}\ s\ \{\,q\,\}$ *.*

**Proof** A fixed interpretation $J$ is assumed and $\theta$ satisfies $p$ is written for $(\theta, J) \models p$. It is sufficient to show that if $\theta$ satisfies $p$ and $\vdash \{\,p\,\}\ s\ \{\,q\,\}$ then $\mathcal{D}(s)(\theta)$ satisfies $q$. This is shown by induction on the depth of the proof tree for $\{\,p\,\}\ s\ \{\,q\,\}$, by looking at the last rule used. The only new case, compared to theorem 6.4.12, is that of rule (While).

- Assume rule (While) is used with program $s$, condition $c$ and invariant $p$. We need to show that $\mathcal{D}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od})(\theta)$ satisfies $p \wedge \mathbb{P}(c) = 0$.

  We have $\mathcal{D}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od})(\theta) = L_{\langle c,s\rangle}(\theta) = \lim_{n\to\infty} \neg c?\mathit{if}^n_{\langle c,s\rangle}(\theta)$. Clearly, for each $n$, the state $\neg c?\mathit{if}^n_{\langle c,s\rangle}(\theta)$ assigns probability 0 to any deterministic state which satisfies $c$. The limit $L_{\langle c,s\rangle}(\theta)$ therefore also assigns probability 0 to any state which satisfies $c$. In other words we have $\mathcal{D}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od})(\theta) \models \mathbb{P}(c) = 0$.

  The induction assumption gives that $\models \{\,p\,\}$ `if` $c$ `then` $s$ `else skip fi` $\{\,p\,\}$ holds. For each state $\theta$ that satisfies $p$ it follows by induction on $n$ that $\mathit{if}^n_{\langle c,s\rangle}(\theta)$ also satisfies $p$ and, as $\sum \neg c?\mathit{if}^n_{\langle c,s\rangle}(\theta)[\mathcal{S}] = r(\theta)^n_{\langle c,s\rangle} \le r^n_{\langle c,s\rangle}$, the sequence $(\mathit{if}^n_{\langle c,s\rangle}(\theta))_{n\in\mathbb{N}}$ is a $\langle c,s\rangle$-sequence within $p$. By $\langle c,s\rangle$-closedness of $p$ the least upper bound of this sequence, i.e. the state $\mathcal{D}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od})(\theta)$ also satisfies $p$. $\Box$

In this section a way has been given to check properties of programs containing `while` loops by using invariants. The two example programs in the next subsection illustrates this technique. For these two programs the invariant is clear but as for non-probabilistic programs, and maybe even more so, finding an invariant for a loop for probabilistic programs can, in general, be far from trivial.

In the non-probabilistic setting, finding a weakest precondition formula for the `while`-construct and showing completeness of the extended proof system is significantly more involved than for other constructs like assignment, sequential composition and alternative composition, see e.g. [34]. Although some preliminary results are available on the issue of finding a complete logic for $\mathcal{L}_{\text{pw}}$, a completeness result for $pH_w$ is not pursued in this thesis and would likely require extensive further work, including a study of the expressiveness of probabilistic predicates.

## 6.6.1 Two example programs

In this subsection two programs are studied. The first program adds an array of numbers, but some elements may inadvertently get skipped. This program is an adapted version of the 'erratic sequence accumulator' example from [162].

```
Int $ss[1 \dots N],\ k,\ t$;
$t$ := 0;      $k$ := 1;
while $(k \leq N)$ do $t$ := $t + ss[k] \oplus_\rho$ skip; $k$ := $k + 1$ od
```

A lower bound on the probability that the answer will still be correct is deduced. Instead of giving complete proof trees a proof outline is given. In a proof outline the rules (Cons) and (Seq) are implicitly used by writing predicates within the program and some basic steps are skipped. A predicate inserted at some point in the program gives a condition that the intermediate states at this point in the computation must satisfy. The shorthand $\exists \mathtt{i} \leq e : p$ is used for $\exists \mathtt{i} : (\mathtt{i} \leq e) \wedge p$. Similarly $\forall \mathtt{i} \leq e : p$ is short for $\forall \mathtt{i} : (\mathtt{i} > e) \vee p$.

```
Int $ss[1 \dots N],\ k,\ t$;
$\{\, \mathbb{P}(\mathtt{true}) = 1 \,\} \Rightarrow \{\, \mathbb{P}(0 = 0 \wedge 1 = 1) = 1 \,\}$
$t$ := 0;      $k$ := 1;
$\{\, \mathbb{P}(t = 0, k = 1) = 1 \,\} \Rightarrow$
$\{\, \mathbb{P}(k = N + 1 \wedge t = \sum_{i=1}^{N} ss[i]) \geq \rho^N \ \vee\ \exists n \leq N : \mathbb{P}(k = n, t = \sum_{i=1}^{k-1} ss[i]) \geq \rho^{n-1} \}$

while $(k \leq N)$ do
    $\{\, \exists n \leq N : \mathbb{P}(k = n, t = \sum_{i=1}^{k-1} ss[i]) \geq \rho^{n-1} \,\} \Rightarrow$
    $\{\, \exists n \leq N : \mathbb{P}(k = n, t + ss[k] = \sum_{i=1}^{k} ss[i]) \geq \rho^{n-1} \,\}$
    $t$ := $t + ss[k] \oplus_\rho$ skip;
    $\{\, \exists n \leq N : \mathbb{P}(k = n, t + ss[k] = \sum_{i=1}^{k} ss[i]) \geq \rho^{n-1} \oplus_\rho \mathtt{true} \,\} \Rightarrow$
    $\{\, \exists m \leq N + 1 : \mathbb{P}(k + 1 = m, t = \sum_{i=1}^{k-1} ss[i]) \geq \rho^{m-1} \,\}$
    $k := k + 1$
    $\{\, \exists m \leq N + 1 : \mathbb{P}(k = m, t = \sum_{i=1}^{k-1} ss[i]) \geq \rho^{m-1} \,\}$
od
$\{\, \mathbb{P}(k \leq N) = 0 \wedge \exists n \leq N + 1 : \mathbb{P}(k = n, t = \sum_{i=1}^{k-1} ss[i]) \geq \rho^{n-1} \,\} \Rightarrow$
$\{\, \mathbb{P}(t = \sum_{i=1}^{N} ss[i]) \geq \rho^N \,\}$
```

The second implication used in this proof outline, $\big(\mathbb{P}(t = 0, k = 1) = 1\big) \Rightarrow \big(\mathbb{P}(k = N + 1 \wedge t = \sum_{i=1}^{N} ss[i]) \geq \rho^N \ \vee\ \exists n \leq N : \mathbb{P}(k = n, t = \sum_{i=1}^{k-1} ss[i]) \geq \rho^{n-1}\big)$, is clear by taking $n = 1$, for which value the right part of the conclusion becomes $\mathbb{P}(k = 1, t = 0) = 1$. For the first program statement within the `while` loop the rule (Assign) is used for the program $t$ `:=` $t + ss[k]$. The result of this rule is combined with the conclusion `true` for the program `skip` by using the rule (Prob). The derivation step for the second program statement in the loop is obtained by using the rule (Assign). The final implication is clear from the fact that for any value $n < N + 1$ the predicate $\mathbb{P}(k = n, t = \sum_{i=1}^{k-1} ss[i]) \geq \rho^{n-1}$ implies a positive probability for $k \leq N$ Note that there is no need to check a closedness condition for the invariant as the program is terminating.

In the second program, a coin is tossed until heads is thrown.

```
Bool $done$;    Int $x$;
$done$ := false;    $x$ := 0;
while $\neg done$ do $x := x + 1; done :=$ true $\oplus_{\frac{1}{2}}$ skip od
```

The number of required throws is shown to be geometrically distributed. For ease of notation the following shorthands are used.

$$\begin{aligned} p &= q_\infty \vee \exists i : q \\ q &= \mathbb{P}(x = \mathtt{i}, done = false) = \tfrac{1}{2}^{\mathtt{i}} \wedge \forall \mathtt{j} \in \{1, \ldots, \mathtt{i}\} : \mathbb{P}(x = \mathtt{j}, done = true) = \tfrac{1}{2}^{\mathtt{j}} \\ q_\infty &= \forall \mathtt{j} > 0 : \mathbb{P}(x = \mathtt{j}, done = true) = \tfrac{1}{2}^{\mathtt{j}} \end{aligned}$$

Assuming $p$ is an invariant and using rule (While) gives

```
$\{\, \mathbb{P}(\mathtt{true}) = 1 \,\}$
$done$ := false;    $x$ := 0;
$\{\, \mathbb{P}(x = 0, done\ = false) = 1 \,\} \Rightarrow \{\, p \,\}$
while $\neg done$ do $x := x + 1; done :=$ true $\oplus_{\frac{1}{2}}$ skip od
$\{\, p \wedge \mathbb{P}(\neg done) = 0 \,\} \Rightarrow \{\, \forall n > 0 : \mathbb{P}(x = n) = \frac{1}{2}^{n} \,\}$
```

To show that $p$ is an invariant the rule (Or) is used to split the proof into two parts, the first of which is trivial. For the second part the rule (Exists) is used to give:

```
$\{\, q \,\}$
while $\neg done$ do
    $\{\, (\neg done)?q \,\} \Rightarrow$
    $\{\, \mathbb{P}(x = i, done\ = false) = \frac{1}{2}^{i} \,\}$
    $x := x + 1$;
    $\{\, \mathbb{P}(x = i + 1, done\ = false) = \frac{1}{2}^{i} \,\}$
    $done$ := true  $\oplus_{\frac{1}{2}}$  skip
    $\{\, \mathbb{P}(x = i + 1, done\ = false) = \frac{1}{2}^{i+1} \wedge \mathbb{P}(x = i + 1, done\ = true) = \frac{1}{2}^{i+1} \,\}$
od
$\{\, \mathbb{P}(x = i + 1, done\ = false) = \frac{1}{2}^{i+1} \wedge \mathbb{P}(x = i + 1, done\ = true) = \frac{1}{2}^{i+1} \wedge$
  $\forall j \in \{\, 1, \ldots, i \,\}$: $\mathbb{P}(x = j, done\ = true) = \frac{1}{2}^{j} \}\ \Rightarrow \{\, q[\mathtt{i + 1/i}] \,\} \Rightarrow \{\, p \,\}$
```

What remains to be shown is that the predicate $p$ is $\langle \neg done, x$ := $x + 1; done$ := true $\oplus_{\frac{1}{2}}$ skip$\rangle$-closed.

Assume some fixed interpretation $J$. First we find the $n$-step termination ratio. For states satisfying $q_\infty$, the termination ratio is clearly 1. For a state satisfying $q$ we use the the Hoare triple $\{\, q \,\}$ if $c$ then $s$ else skip fi $\{\, q[\mathtt{i + 1/i}] \,\}$ deduced above. This gives that if $(\theta, J) \models q$ then $(if^{n}_{\langle c,s\rangle}(\theta), J) \models q[\mathtt{i} + n/\mathtt{i}]$ for all $n$. As $q[\mathtt{i} + n/\mathtt{i}]$ implies that $\mathbb{P}(done) \geq 1 - \frac{1}{2}^{n}$ we have $(\theta, J) \models q \Rightarrow r(\theta)^{n}_{\langle c,s\rangle} \geq 1 - \frac{1}{2}^{n}$. So for all states that satisfy the predicate $p$, the $n$-step termination ratio is at least $1 - \frac{1}{2}^{n}$, $r^{n}_{\langle c,s\rangle} \geq 1 - \frac{1}{2}^{n}$.

Next we show that any $\langle c, s\rangle$-sequence within $p$ terminates in a state satisfying $q_\infty$. The predicate $p$ is divided in $p_n = q_\infty \vee \exists i \geq n : q$ and $\exists i < n : q$. Clearly $p$ is equivalent with $p_n \vee (\exists i < n : q)$. A state satisfying $\exists i < n : q$ cannot have a termination ratio of $1 - \frac{1}{2}^n$. This means that if $(\theta_n)_{n\in\mathbb{N}}$ is a $\langle c, s\rangle$-sequence within $p$ then $\theta_n$ satisfies $p_n$. All states $\theta_n$ with $n \geq \mathtt{j}$ satisfy $\mathbb{P}(x = \mathtt{j}, \mathit{done} = \mathit{true}) = \frac{1}{2}^{\mathtt{j}}$. The probability $\mathbb{P}(x = \mathtt{j}, \mathit{done} = \mathit{true})$ cannot change in the least upper bound point of the sequence. The sequence must therefore terminate in a state satisfying $\mathbb{P}(x = \mathtt{j}, \mathit{done} = \mathit{true}) = \frac{1}{2}^{\mathtt{j}}$. This holds for all possible values of $\mathtt{j}$.

Note that the predicate $p$ would not be closed without the term $q_\infty$.

## 6.7 Another extension of $\mathcal{L}_{\text{pif}}$: Adding nondeterminism

In chapter 4 it is shown that not all choices can be adequately described by probabilistic choices. A choice can be a probabilistic choice for which the probability is not known. It is also possible that a choice does not satisfy the properties of a probabilistic choice at all.

One type of choice which cannot be described by probabilistic choice is the choice of an opponent in a game, or the choices of a user of a system. In this section an operator $\Box$, used to denote nondeterministic choice, is added to the language $\mathcal{L}_{\text{pif}}$ to be able to describe this type of choice. The resulting language is called $\mathcal{L}_{\text{pnif}}$. The program $s \Box s'$ can be interpreted as a choice for the user to either execute $s$ or to execute $s'$. The user can also be seen as an opponent from the point of the verification of the algorithm. The algorithm has to work correctly, no matter which choices the user makes.

The verification of a single program is seen as a single player game. It is also possible to analyze a game between two players. In a two player game a probabilistic strategy for one player can be described by a program. The nondeterministic choices in such a program describe the choices for the other player.

In the situations given above the question one wants to answer is which properties are sure to hold, for all possible choices of the opponent. In other words one is interested in the worst case behavior for the nondeterministic choice. It is also possible to look at a best case behavior for the nondeterminism. In such a case one looks for properties which can be made to hold by making the nondeterministic choices in the right way. This section will mainly deal with investigating the worst case behavior of an algorithm. The best case behavior, however, can be dealt with in the same framework.

In chapter 4 the choice of an opponent is referred to as one type of 'information-oriented' nondeterminism. The choice has to be made with the right amount of information. As the probabilistic choice is unpredictable, the nondeterministic choice cannot depend on the outcome of probabilistic choices which have not yet been made. This is not a restriction on the way the nondeterminism is decided but a consequence of the assumption of unpredictability of the probabilistic choice. If the result of a choice can somehow be known beforehand, the choice is not probabilistic.

The nondeterministic choice can also not depend on information which is not available for the opponent, for example the value of a variable that the opponent cannot access. The opponent is assumed not to be able to see the value of any of the variables. Any observation that the user can make based on the value of the variables is made explicit in the program.

In subsection 6.7.1 the language $\mathcal{L}_{\mathrm{pnif}}$ is defined. To describe the meaning of a program in $\mathcal{L}_{\mathrm{pnif}}$ a denotational semantics $\mathcal{D}$ is also given in this subsection. The denotational semantics gives the set of all possible meanings, depending on how the nondeterminism is resolved. The elements of the set are distributions as used in section 6.3.

In subsection 6.7.2 the Hoare-style logic *pH* introduced in section 6.4 is extended to deal with nondeterministic programs from $\mathcal{L}_{\mathrm{pnif}}$. The extended proof system, denoted by $pH_{nd}$, is shown to be sound. In subsection 6.7.3 weakest preconditions are introduced. The conjecture that the logic $pH_{nd}$ is complete is also discussed in this subsection.

In subsection 6.7.4 the "3 doors problem", introduced in section 4.2 is again considered. The problem is translated into programs in $\mathcal{L}_{\mathrm{pnif}}$ and the logic $pH_{nd}$ is used to derive some properties of these programs.

### 6.7.1 The syntax and semantics of the language $\mathcal{L}_{\mathrm{pnif}}$

The language $\mathcal{L}_{\mathrm{pnif}}$ extends the language $\mathcal{L}_{\mathrm{pif}}$ introduced in definition 6.3.6 by adding the operator $\Box$ describing nondeterministic choice. Expressions and boolean conditions are as defined in section 6.3. Note that the language $\mathcal{L}_{\mathrm{pnif}}$ extends $\mathcal{L}_{\mathrm{pif}}$ and not $\mathcal{L}_{\mathrm{pw}}$. The language $\mathcal{L}_{\mathrm{pnif}}$ does not contain the `while`-construct. Further study, not reported upon in this thesis, is required before a Hoare-style logic can be given for the language which adds a `while`-construct to the language $\mathcal{L}_{\mathrm{pnif}}$. The complications in giving the semantics for such a language seem only to be of a technical nature. It is, however, not clear how to extend the notion of invariant used in the (While) rule of the Hoare-style logic introduced in section 6.6 to a nondeterministic setting. (Recall that the notion of invariant in the probabilistic setting is more involved than in the non-probabilistic setting because probabilistic predicates can also express claims about the probability of termination.)

**Definition 6.7.1** *Let PVar be a set of program variables and let x range over PVar. The language $\mathcal{L}_{\mathrm{pnif}}$, ranged over by s, is given by*

$$s ::= \texttt{skip} \mid x \texttt{ := } e \mid s \texttt{ ; } s \mid s \oplus_\rho s \mid \texttt{if } c \texttt{ then } s \texttt{ else } s \texttt{ fi} \mid s \Box s$$

*where $e \in Exp\langle PVar\rangle$, $c \in BC\langle PVar\rangle$ and $\rho$ is a ratio in the open interval $(0, 1)$.*

The program $s \Box s'$ is new compared to the setup of the language $\mathcal{L}_{\mathrm{pif}}$, the other programs are interpreted as before. The program $s \Box s'$ makes a nondeterministic choice. The only thing known is that at this point a choice will be made and the program will execute either $s$ or $s'$. There is no probabilistic information available about the choice. It is not even known whether the choice is determined in a probabilistic fashion.

To describe the state of computation of a deterministic program the values of the program variables have to be given. A deterministic state in $\mathcal{S}$ gives the value for each program variable. It is not known what the value of the variables will be after executing, e.g. the probabilistic program ($x$ `:= 1` $\oplus_{\frac{1}{2}}$ $x$ `:= 3`), but a distribution in $\Theta$ gives the probabilities for each value. What the value of the variable $x$ will be after executing the program ($x$ `:= 1` $\oplus_{\frac{1}{2}}$ $x$ `:= 3`) $\Box$ $x$ `:= 2` is not known, it can be 1, 2 or 3. Different than for the purely probabilistic program $(x \texttt{ := } 1 \oplus_{\frac{1}{2}} x \texttt{ := } 3)$, however, the probability for each of these values is also not known. The probability that $x$ becomes 1 is $\frac{1}{2}$ if the program $(x \texttt{ := } 1 \oplus_{\frac{1}{2}} x \texttt{ := } 3)$ is chosen and zero if $x \texttt{ := } 2$ is chosen. The state of a program in $\mathcal{L}_{\text{pnif}}$ is not a distribution over deterministic states as for programs in $\mathcal{L}_{\text{pif}}$. Instead both nondeterminism and probability have to be resolved before the actual value of the variables can be found.

In chapter 4 the combination of nondeterminism and probabilistic choice has been studied and several interpretations for the nondeterminism have been proposed. In this section the approach of sections 4.2 and 4.4 is followed. The situation in these sections is as follows: The nondeterminism described can be interpreted as a user's or opponent's choice (as opposed to the resource oriented view of nondeterminism used in section 4.3). Delaying a probabilistic choice until after a nondeterministic choice has been made gives the same outcome as not delaying the choice. This fact, illustrated in example 6.7.2 below, is used to give a meaning to programs with both nondeterminism and probabilistic choice. Such programs are treated as a nondeterministic choice between probabilistic programs. The meaning of a nondeterministic program is given by a process consisting of a set of probabilistic subprocesses.

For a program in $\mathcal{L}_{\text{pnif}}$ it is also possible to delay a probabilistic choice and interpret the program as a nondeterministic choice between probabilistic programs. A nondeterministic choice, on the other hand, cannot be delayed. As the choice is made at this point in the execution, it cannot depend on the outcome of future probabilistic choices. The probabilistic choice is assumed to be unpredictable; the outcome of the choice cannot be known before it is made.

**Example 6.7.2** *The programs in this example are interpreted as games. The nondeterministic choices in a program describe the choices of the user who plays the game. The value of the program variable* x *at the end of the program is the amount the user won by playing the game.*

*The games described by the programs* $(x \texttt{ := } 1 \,\square\, x \texttt{ := } 2) \oplus_{\frac{1}{2}} x \texttt{ := } 3$ *and* $(x \texttt{ := } 1 \oplus_{\frac{1}{2}} x \texttt{ := } 3) \,\square\, (x \texttt{ := } 2 \oplus_{\frac{1}{2}} x \texttt{ := } 3)$ *are equivalent. In the first program the user may get to choose between* 1 *and* 2 *for* x*. In the second program the user has to decide for* 1 *or* 2 *before the probabilistic choice is made. This choice will not influence the probabilistic choice, so exactly the same results can be obtained. Delaying the probabilistic choice until after the nondeterministic choice does not change the behavior.*

*In the game described by the program* $(x \texttt{ := } 1 \oplus_{\frac{1}{2}} x \texttt{ := } 3) \,\square\, x \texttt{ := } 2$ *the user can choose for a certain value of two or gamble between the values one and three. This is clearly a different situation than for the program* $(x \texttt{ := } 1 \,\square\, x \texttt{ := } 2) \oplus_{\frac{1}{2}} (x \texttt{ := } 3 \,\square\, x \texttt{ := } 2)$ *where the nondeterministic choice is made after the probabilistic choice. In the second program the user always has an easy choice as the exact value obtained for either option is known. The user can simply select the higher of the two values. As a result the user can obtain higher expected winnings* ($2\frac{1}{2}$ *versus* 2) *in the game described by the second program. Allowing the nondeterministic choice to be made when the result of the probabilistic choice is known results in a different behavior.*

These examples are the same as the examples in the introduction of chapter 4 except that the uninterpreted atomic actions used there are replaced by assignment statements. The examples show that the interpretation of nondeterminism corresponds to an 'information-oriented' view of nondeterminism. A nondeterministic choice has to be made with the correct amount of information.

The programs in $\mathcal{L}_{\mathrm{pif}}$ are interpreted as transformers of probabilistic states. The denotational semantics of a program gives, for each probabilistic state, the result of executing the program starting in this state. The programs in $\mathcal{L}_{\mathrm{pnif}}$ are interpreted as transformers of *nondeterministic states*. A program in $\mathcal{L}_{\mathrm{pnif}}$ is seen as a nondeterministic choice between probabilistic programs. This means that the state after running a program can be one of several probabilistic states. A nondeterministic state, therefore, is a set of probabilistic states. The set of all nondeterministic states is denoted by $\Pi$.

**Definition 6.7.3** *The set of nondeterministic states* $\Pi$*, ranged over by* $\pi$*, is given by*

$$\Pi = \mathcal{P}_f(\Theta)$$

A nondeterministic state $\pi$ is a finite set containing probabilistic states in $\Theta$. The probabilistic states in a nondeterministic state are referred to as (probabilistic) substates. Each probabilistic substate gives the probabilities associated with one of the nondeterministic alternatives of the state.

**Example 6.7.4** *Assuming that the only program variable is* $\mathrm{x}$ *the following sets are nondeterministic states in* $\Pi$, $\{\, 1\langle \mathrm{x} = 0\rangle \,\}$, $\{\, \frac{1}{2}\langle \mathrm{x} = 1\rangle + \frac{1}{2}\langle \mathrm{x} = 3\rangle, 1\langle \mathrm{x} = 2\rangle \,\}$. *By running the program* $(\mathtt{x} \mathrel{\mathtt{:=}} \mathtt{x} + \mathtt{1} \oplus_{\frac{1}{2}} \mathtt{x} \mathrel{\mathtt{:=}} \mathtt{3}) \,\Box\, \mathtt{x} \mathrel{\mathtt{:=}} \mathtt{2}$ *in the first state, the second state is obtained.*

The example shows the state resulting from executing a program starting in another state. To give the effects of the execution of a program in general, a denotational semantics is defined. The denotational semantics can be used to give a formal justification for the claim in the example above. To be able to define the denotational semantics, the meaning of a program consisting of a single assignment is given and the operations $\oplus_\rho$ and $+$ introduced below are used to find the meaning of programs starting with a probabilistic choice and a conditional choice respectively. Taking the union of nondeterministic states is used to give the meaning of a program built with $\Box$.

In section 6.3 the variant of a probabilistic state is introduced and used to give the effect on a probabilistic state of the execution of an assignment. The notion of a variant of a nondeterministic state is introduced here to find the effect of the execution of an assignment on a nondeterministic state. The value of a variable is not necessarily determined in a nondeterministic state. As with probabilistic states, an expression cannot be evaluated in a nondeterministic state. Instead, the notion of variant of a probabilistic state is lifted to nondeterministic states.

**Definition 6.7.5** *The variant* $\pi[f/\mathrm{x}]$ *of a state* $\pi$ *is given by*

$$\pi[f/\mathrm{x}] \quad = \quad \{\, \theta[f/\mathrm{x}] \mid \theta \in \pi \,\}$$

*where* $\mathrm{x} \in PVar$ *and* $f : \mathcal{S} \to Int_\perp$.

The variant of a nondeterministic state is obtained by using the variant of a probabilistic state as introduced in section 6.3. The variant of a state $\pi$ is the set consisting of the variants of the probabilistic states in $\pi$.

**Example 6.7.6** *For the following probabilistic states taking the variant where* 3 *is substituted for* x *gives*

$$\begin{array}{rcl}(1\langle x=1,y=1\rangle)[\mathtt{3}/x] &=& 1\langle x=3,y=1\rangle\\ (\frac{1}{3}\langle x=1,y=1\rangle+\frac{2}{3}\langle x=1,y=2\rangle)[\mathtt{3}/x] &=& \frac{1}{3}\langle x=3,y=1\rangle+\frac{2}{3}\langle x=3,y=2\rangle\\ (\frac{1}{3}\langle x=2,y=1\rangle+\frac{2}{3}\langle x=2,y=2\rangle)[\mathtt{3}/x] &=& \frac{1}{3}\langle x=3,y=1\rangle+\frac{2}{3}\langle x=3,y=2\rangle\end{array}$$

*As will be seen below, the effect of the assignment* $\mathtt{x := 3}$ *on a state corresponds to taking the variant* $[\mathtt{3}/x]$ *of the state. Taking the variant* $[\mathtt{3}/x]$ *in the state* $\pi$ *given by*

$$\begin{array}{rcl}\pi &=& \{\,1\langle x=1,y=1\rangle,\ \frac{1}{3}\langle x=1,y=1\rangle+\frac{2}{3}\langle x=1,y=2\rangle,\\ && \frac{1}{3}\langle x=2,y=1\rangle+\frac{2}{3}\langle x=2,y=2\rangle\,\}\end{array}$$

*results in the state*

$$\pi[3/x]=\{\,1\langle x=3,y=1\rangle,\ \tfrac{1}{3}\langle x=3,y=1\rangle+\tfrac{2}{3}\langle x=3,y=2\rangle\,\}$$

The effect of an assignment can be easily found by taking the variant of a state. In section 6.3 the operations $\oplus_\rho$ and $+$ are used to find the effect of executing a probabilistic choice or a conditional choice. The operations $\oplus_\rho$ and $+$ can be lifted to work on $\Pi$ in the same way as we lifted the notion of a variant of a state in definition 6.7.5 above.

**Definition 6.7.7** *The probabilistic combination* $\oplus_\rho$ *and the sum* $+$ *of two nondeterministic states are given by* $\oplus_\rho, + \,:\, \Pi\times\Pi\to\Pi$

$$\begin{array}{rcl}\pi\oplus_\rho\pi' &=& \{\,\theta\oplus_\rho\theta' \mid \theta\in\pi,\theta'\in\pi'\,\}\\ \pi+\pi' &=& \{\,\theta+\theta' \mid \theta\in\pi,\theta'\in\pi'\,\}\end{array}$$

The substates in the probabilistic combination of two nondeterministic states are obtained by combining each substate of the first nondeterministic state with each substate of the second nondeterministic state. Similarly for the sum of two nondeterministic states.

**Example 6.7.8** *The lifting of the operations* $\oplus_\rho$ *and* $+$ *to sets is obtained by taking all possible combinations:* $\{\,1\langle x=1\rangle,1\langle x=2\rangle\,\}\oplus_{\frac{1}{2}}\{\,1\langle x=2\rangle,1\langle x=3\rangle\,\}=\{\,\frac{1}{2}\langle x=1\rangle+\frac{1}{2}\langle x=2\rangle,\frac{1}{2}\langle x=1\rangle+\frac{1}{2}\langle x=3\rangle,1\langle x=2\rangle,\frac{1}{2}\langle x=2\rangle+\frac{1}{2}\langle x=3\rangle\,\}$ *and* $\{\,\frac{1}{3}\langle x=1\rangle,\frac{1}{3}\langle x=2\rangle\,\}+\{\,\frac{2}{3}\langle x=2\rangle\,\}=\{\,\frac{1}{3}\langle x=1\rangle+\frac{2}{3}\langle x=2\rangle,1\langle x=2\rangle\,\}$.

All substates in the sum of $\{\,\frac{1}{3}\langle x=1\rangle,\frac{1}{3}\langle x=2\rangle\,\}$ and $\{\,\frac{2}{3}\langle x=2\rangle\,\}$ have a total probability of 1. As programs in $\mathcal{L}_{\text{pnif}}$ always terminate, no 'missing' probability will be introduced as in section 6.6. If the total probability of each substate before the execution of the program is 1 then the total probability of each substate after the execution of a program will also be 1.

The denotational semantics $\mathcal{D}$ for $\mathcal{L}_{\text{pnif}}$ gives, for each program $s$ and state $\pi$, the state $\mathcal{D}(s)(\pi)$ resulting from executing the program $s$ starting in state $\pi$.

**Definition 6.7.9** *The denotational semantics* $\mathcal{D} : \mathcal{L}_{\text{pnif}} \to (\Pi \to \Pi)$ *is given by*

$$\begin{array}{rcl}
\mathcal{D}(\texttt{skip})(\pi) & = & \pi \\
\mathcal{D}(x \texttt{ := } e)(\pi) & = & \pi[\mathcal{V}(e)/x] \\
\mathcal{D}(s \texttt{ ; } s')(\pi) & = & \mathcal{D}(s')(\mathcal{D}(s)(\pi)) \\
\mathcal{D}(s \oplus_\rho s')(\pi) & = & \cup \{ \mathcal{D}(s)(\{ \theta \}) \oplus_\rho \mathcal{D}(s')(\{ \theta \}) \mid \theta \in \pi \} \\
\mathcal{D}(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi})(\pi) & & \\
 & = & \cup \{ \mathcal{D}(s)(\{ c?\theta \}) + \mathcal{D}(s')(\{ \neg c?\theta \}) \mid \theta \in \pi \} \\
\mathcal{D}(s \,\Box\, s')(\pi) & = & \mathcal{D}(s)(\pi) \cup \mathcal{D}(s')(\pi)
\end{array}$$

The first three clauses have changed little compared to definition 6.3.12. The program `skip` does nothing, so the start state is also the end state. The result of a single assignment $x \texttt{ := } e$ in a state $\pi$ is given by the variant $\pi[\mathcal{V}(e)/x]$ of the state $\pi$. To execute the program $s;s'$ the program $s$ is executed and in the resulting state $\mathcal{D}(s)(\pi)$ the program $s'$ is executed.

To find the meaning of the programs $s \oplus_\rho s'$ and `if` $c$ `then` $s$ `else` $s'$ `fi` the operations $\oplus_\rho$ and $+$ are used respectively. At first sight it may seem sufficient to use $\mathcal{D}(s)(\pi) \oplus_\rho \mathcal{D}(s')(\pi)$ as the definition of $\mathcal{D}(s \oplus_\rho s')(\pi)$. This definition, however, would not give the correct result. The start state $\pi$ is a nondeterministic state. Each element of this nondeterministic state belongs to one way of resolving the nondeterminism of the execution up till now. A nondeterministic choice cannot depend on the outcome of a probabilistic choice which has not yet been made. In $\pi$ the nondeterministic choices made in the past are collected. These choices have therefore been fixed at this point. The same probabilistic substate from $\pi$ has to be used for both of the two possible outcomes of the probabilistic choice. In $\mathcal{D}(s)(\pi) \oplus_\rho \mathcal{D}(s')(\pi)$ different probabilistic substates from $\pi$ can be selected for $s$ and $s'$. This means that it would be possible to take different options for previously made nondeterministic choices depending on the result of the current probabilistic choice.

For a conditional choice a similar remark can be made as for the probabilistic choice. As the nondeterministic choices collected in the state $\pi$ have already been made, the same probabilistic substate of $\pi$ has to be used for both alternatives of the conditional choice.

The nondeterministic alternatives available after execution of $s \,\Box\, s'$ are the alternatives available after the execution of $s$ together with the alternatives available after the execution of $s'$.

**Example 6.7.10** *In the program* $(x \texttt{ := 1 } \Box\ x \texttt{ := 2})\texttt{ ; }(y \texttt{ := 1 } \oplus_{\frac{1}{2}} y \texttt{ := 2})$ *the user has to choose between* 1 *and* 2 *for* $x$ *first, after which a probabilistic choice is made between* 1 *and* 2 *for* $y$. *As the user has no way of knowing the outcome of the probabilistic choice, the probability that* $x$ *equals* $y$ *should be* $\frac{1}{2}$ *no matter what choices the user makes. This can be checked by looking at the denotational semantics of the program. Clearly the values of* $x$ *and* $y$ *at the start of execution are irrelevant. Without loss of generality both* $x$ *and* $y$ *are assumed to be* 0. *The denotational semantics can be found as follows.*

$$\begin{array}{l}
\mathcal{D}((x \texttt{ := 1 } \Box\ x \texttt{ := 2})\texttt{ ; }(y \texttt{ := 1 } \oplus_{\frac{1}{2}} y \texttt{ := 2}))(\{ 1\langle x = 0, y = 0\rangle \}) \\
\quad = \ \mathcal{D}(y \texttt{ := 1 } \oplus_{\frac{1}{2}} y \texttt{ := 2})(\mathcal{D}(x \texttt{ := 1 } \Box\ x \texttt{ := 2})(\{ 1\langle x = 0, y = 0\rangle \})) \\
\quad = \ \mathcal{D}(y \texttt{ := 1 } \oplus_{\frac{1}{2}} y \texttt{ := 2})(\{ 1\langle x = 1, y = 0\rangle, 1\langle x = 2, y = 0\rangle \})
\end{array}$$

$$\begin{aligned}
&= \mathcal{D}(y \texttt{ := 1})(\{\, 1\langle x=1, y=0\rangle \,\}) \oplus_{\frac{1}{2}} \mathcal{D}(y \texttt{ := 2})(\{\, 1\langle x=1, y=0\rangle \,\}) \\
&\quad \cup \mathcal{D}(y \texttt{ := 1})(\{\, 1\langle x=2, y=0\rangle \,\}) \oplus_{\frac{1}{2}} \mathcal{D}(y \texttt{ := 2})(\{\, 1\langle x=2, y=0\rangle \,\}) \\
&= (\{\, 1\langle x=1, y=1\rangle \,\} \oplus_{\frac{1}{2}} \{\, 1\langle x=1, y=2\rangle \,\}) \\
&\quad \cup (\{\, 1\langle x=2, y=1\rangle \,\} \oplus_{\frac{1}{2}} \{\, 1\langle x=2, y=1\rangle \,\}) \\
&= \{\, \tfrac{1}{2}\langle x=1, y=1\rangle + \tfrac{1}{2}\langle x=1, y=2\rangle, \tfrac{1}{2}\langle x=2, y=1\rangle + \tfrac{1}{2}\langle x=2, y=2\rangle \,\}
\end{aligned}$$

*For both possible choices the probability that* $x$ *equals* $y$ *is indeed* $\frac{1}{2}$.

*In the program* $(y \texttt{ := 1 ; } (x \texttt{ := 1} \,\Box\, x \texttt{ := 2})) \oplus_{\frac{1}{2}} (y \texttt{ := 2 ; } (x \texttt{ := 1} \,\Box\, x \texttt{ := 2}))$ *the user can choose different options for the two different possible outcomes of the probabilistic choice. As a result, in one of the nondeterministic options the probability that* $x$ *equals* $y$ *is* 1*. Assuming again that both* $x$ *and* $y$ *are* 0 *at the start of execution the denotational semantics can be found as follows.*

$$\begin{aligned}
&\mathcal{D}((y \texttt{ := 1 ; } (x \texttt{ := 1} \,\Box\, x \texttt{ := 2})) \\
&\quad \oplus_{\frac{1}{2}} (y \texttt{ := 2 ; } (x \texttt{ := 1} \,\Box\, x \texttt{ := 2})))(\{\, 1\langle x=0, y=0\rangle \,\}) \\
&\quad = \mathcal{D}(y \texttt{ := 1 ; } (x \texttt{ := 1} \,\Box\, x \texttt{ := 2}))(\{\, 1\langle x=0, y=0\rangle \,\}) \\
&\qquad \oplus_{\frac{1}{2}} \mathcal{D}(y \texttt{ := 2 ; } (x \texttt{ := 1} \,\Box\, x \texttt{ := 2}))(\{\, 1\langle x=0, y=0\rangle \,\}) \\
&\quad = \{\, 1\langle x=1, y=1\rangle, 1\langle x=2, y=1\rangle \,\} \\
&\qquad \oplus_{\frac{1}{2}} \{\, 1\langle x=1, y=2\rangle, 1\langle x=2, y=2\rangle \,\} \\
&\quad = \{\, \tfrac{1}{2}\langle x=1, y=1\rangle + \tfrac{1}{2}\langle x=1, y=2\rangle, \tfrac{1}{2}\langle x=1, y=1\rangle + \tfrac{1}{2}\langle x=2, y=2\rangle, \\
&\qquad\ \tfrac{1}{2}\langle x=2, y=1\rangle + \tfrac{1}{2}\langle x=1, y=2\rangle, \tfrac{1}{2}\langle x=2, y=1\rangle + \tfrac{1}{2}\langle x=2, y=2\rangle \,\}
\end{aligned}$$

*For the option* $\frac{1}{2}\langle x=1, y=1\rangle + \frac{1}{2}\langle x=2, y=2\rangle$ *the probability that* $x$ *equals* $y$ *is* 1.

In the second example the user can select the value 1 for $x$ when $y$ equals 1 and select value 2 for $x$ when $y$ is 2. This allows the user to make a selection in which the probability that $x$ equals $y$ is 1. As mentioned in the introduction to this section, the user is assumed not to be able to observe the state, which means that the user cannot see the value of $y$. The reason the user can select different values for the two possible values of $y$ is because there are two different nondeterministic choices. The first choice is always made with a value of 1 for $y$, while the other is always made with a value of 2 for $y$. Because the user does know which choice is being made, the value of $y$ can be deduced.

In the program $(y \texttt{ := 1} \oplus_{\frac{1}{2}} y \texttt{ := 2}) \texttt{ ; } (x \texttt{ := 1} \,\Box\, x \texttt{ := 2})$ the probabilistic choice is made before the nondeterministic choice. However, as there is no way for the user to observe the outcome of the choice, the value of $y$ is not known to the user and the nondeterministic choice cannot depend on this value.

**Example 6.7.11** *For the program* $(y \texttt{ := 1} \oplus_{\frac{1}{2}} y \texttt{ := 2}) \texttt{ ; } (x \texttt{ := 1} \,\Box\, x \texttt{ := 2})$ *there are only two nondeterministic alternatives. For both alternatives the probability that* $x$ *equals* $y$ *is* $\frac{1}{2}$.

$$\begin{aligned}
&\mathcal{D}((y \texttt{ := 1} \oplus_{\frac{1}{2}} y \texttt{ := 2}) \texttt{ ; } (x \texttt{ := 1} \,\Box\, x \texttt{ := 2}))(\{\, 1\langle x=0, y=0\rangle \,\}) \\
&\quad = \mathcal{D}(x \texttt{ := 1} \,\Box\, x \texttt{ := 2})(\mathcal{D}(y \texttt{ := 1} \oplus_{\frac{1}{2}} y \texttt{ := 2})(\{\, 1\langle x=0, y=0\rangle \,\})) \\
&\quad = \mathcal{D}(x \texttt{ := 1} \,\Box\, x \texttt{ := 2})(\{\, \tfrac{1}{2}\langle x=0, y=1\rangle + \tfrac{1}{2}\langle x=0, y=2\rangle \,\}) \\
&\quad = \mathcal{D}(x \texttt{ := 1})(\{\, \tfrac{1}{2}\langle x=0, y=1\rangle + \tfrac{1}{2}\langle x=0, y=2\rangle \,\}) \\
&\qquad \cup \mathcal{D}(x \texttt{ := 2})(\{\, \tfrac{1}{2}\langle x=0, y=1\rangle + \tfrac{1}{2}\langle x=0, y=2\rangle \,\}) \\
&\quad = \{\, \tfrac{1}{2}\langle x=1, y=1\rangle + \tfrac{1}{2}\langle x=1, y=2\rangle, \tfrac{1}{2}\langle x=2, y=1\rangle + \tfrac{1}{2}\langle x=2, y=2\rangle \,\}
\end{aligned}$$

If the user can observe something about the state, this has to be made explicit in the program. This can be done by introducing a conditional choice.

**Example 6.7.12** *The situation where the user is able to observe the value of $y$ after running the program ($y$ `:=` $1 \oplus_{\frac{1}{2}} y$ `:=` $2$) and use this value in the decision for the value of* x *can be described by the program*

$$s_{\mathtt{if}} \quad = \quad \mathtt{if}\ y = 1\ \mathtt{then}\ \mathrm{x} := 1 \,\square\, \mathrm{x} := 2\ \mathtt{else}\ \mathrm{x} := 1 \,\square\, \mathrm{x} := 2\ \mathtt{fi}$$

*The choices which are available to the user are the same for both branches of the conditional choice but the fact that the choice in the left branch or the choice in the right branch is being made provides extra information to the user.*

$$\begin{aligned}
&\mathcal{D}((y := 1 \oplus_{\frac{1}{2}} y := 2)\ ;\ s_{\mathtt{if}})(\{\, 1\langle \mathrm{x} = 0, y = 0\rangle \,\}) \\
&\quad = \mathcal{D}(s_{\mathtt{if}})(\{\, \tfrac{1}{2}\langle \mathrm{x} = 0, y = 1\rangle + \tfrac{1}{2}\langle \mathrm{x} = 0, y = 2\rangle \,\}) \\
&\quad = \mathcal{D}(\mathrm{x} := 1 \,\square\, \mathrm{x} := 2)(\{\, \tfrac{1}{2}\langle \mathrm{x} = 0, y = 1\rangle \,\}) \\
&\qquad + \mathcal{D}(\mathrm{x} := 1 \,\square\, \mathrm{x} := 2)(\{\, \tfrac{1}{2}\langle \mathrm{x} = 0, y = 2\rangle \,\}) \\
&\quad = \{\, \tfrac{1}{2}\langle \mathrm{x} = 1, y = 1\rangle, \tfrac{1}{2}\langle \mathrm{x} = 2, y = 1\rangle \,\} + \{\, \tfrac{1}{2}\langle \mathrm{x} = 1, y = 2\rangle, \tfrac{1}{2}\langle \mathrm{x} = 2, y = 2\rangle \,\} \\
&\quad = \{\, \tfrac{1}{2}\langle \mathrm{x} = 1, y = 1\rangle + \tfrac{1}{2}\langle \mathrm{x} = 1, y = 2\rangle, \tfrac{1}{2}\langle \mathrm{x} = 1, y = 1\rangle + \tfrac{1}{2}\langle \mathrm{x} = 2, y = 2\rangle, \\
&\qquad \tfrac{1}{2}\langle \mathrm{x} = 2, y = 1\rangle + \tfrac{1}{2}\langle \mathrm{x} = 1, y = 2\rangle, \tfrac{1}{2}\langle \mathrm{x} = 2, y = 1\rangle + \tfrac{1}{2}\langle \mathrm{x} = 2, y = 2\rangle \,\}
\end{aligned}$$

The program $(y := 1\ ;\ (\mathrm{x} := 1 \,\square\, \mathrm{x} := 2)) \oplus_{\frac{1}{2}} (y := 2\ ;\ (\mathrm{x} := 1 \,\square\, \mathrm{x} := 2))$ is not equivalent to $(y := 1 \oplus_{\frac{1}{2}} y := 2)\ ;\ (\mathrm{x} := 1 \,\square\, \mathrm{x} := 2)$ but is equivalent to $(y := 1 \oplus_{\frac{1}{2}} y := 2)\,;\,\mathtt{if}\ y = 1\ \mathtt{then}\ \mathrm{x} := 1 \,\square\, \mathrm{x} := 2\ \mathtt{else}\ \mathrm{x} := 1 \,\square\, \mathrm{x} := 2\ \mathtt{fi}$. This again shows that an 'information-oriented' view of nondeterminism is used: The amount of information available to the user is important for the possible outcomes of the choice.

The language $\mathcal{L}_{\mathrm{pnif}}$ is an extension of the language $\mathcal{L}_{\mathrm{pif}}$. The meaning of a program in $\mathcal{L}_{\mathrm{pnif}}$ is given as a function from nondeterministic states to nondeterministic states whereas the meaning of a program in $\mathcal{L}_{\mathrm{pif}}$ is given as a function from probabilistic states to probabilistic states. The semantics of $\mathcal{L}_{\mathrm{pnif}}$, therefore, cannot be an direct extension of the semantics of $\mathcal{L}_{\mathrm{pif}}$. A probabilistic state, however, can be interpreted as a nondeterministic state containing only a single option. Using this interpretation, the semantics of $\mathcal{L}_{\mathrm{pnif}}$ is a conservative extension of the semantics for $\mathcal{L}_{\mathrm{pif}}$. Part (a) of the following lemma makes this statement precise.

**Lemma 6.7.13**

*(a) Using $\mathcal{D}_{pif}$ to denote the denotational semantics introduced in definition 6.3.12 we have*

$$\mathcal{D}(s)(\{\,\theta\,\}) \quad = \quad \{\, \mathcal{D}_{pif}(s)(\theta) \,\}$$

*for all programs $s$ in $\mathcal{L}_{\mathrm{pif}}$ and probabilistic states $\theta$ in $\Theta$.*

*(b) The denotational semantics is 'linear' in the nondeterministic state, i.e.*

$$\mathcal{D}(s)(\pi \cup \pi') \quad = \quad \mathcal{D}(s)(\pi) \cup \mathcal{D}(s)(\pi')$$

*for all programs $s$ in $\mathcal{L}_{\mathrm{pnif}}$ and states $\pi, \pi'$ in $\Pi$.*

*(c) For a program $s$ and a state $\pi$ the denotational semantics $\mathcal{D}(s)(\pi)$ can be obtained for each probabilistic substate $\theta$ in $\pi$ separately, meaning*

$$\mathcal{D}(s)(\pi) \quad = \quad \cup \{ \mathcal{D}(s)(\{ \theta \}) \mid \theta \in \pi \}$$

**Proof**

(a) The proof proceeds by structural induction on the program $s$ from $\mathcal{L}_{\text{pif}}$. Only two typical cases are given.

- $\mathcal{D}(x := e)(\{ \theta \}) = \{ \theta \}[\mathcal{V}(e)/x] = \{ \theta[\mathcal{V}(e)/x] \} = \{ \mathcal{D}_{\text{pif}}(x := e)(\theta) \}$
- $$\begin{aligned} \mathcal{D}(s_1 \ ; \ s_2)(\{ \theta \}) &= [\text{def. 6.7.9}] \quad \mathcal{D}(s_2)(\mathcal{D}(s_1)(\{ \theta \})) \\ &= [\text{ind. hyp.}] \quad \mathcal{D}(s_2)(\{ \mathcal{D}_{\text{pif}}(s_1)(\theta) \}) \\ &= [\text{ind. hyp.}] \quad \{ \mathcal{D}_{\text{pif}}(s_2)(\mathcal{D}_{\text{pif}}(s_1)(\theta)) \} \\ &= [\text{def. 6.3.12}] \quad \{ \mathcal{D}_{\text{pif}}(s_1 \ ; \ s_2)(\theta) \} \end{aligned}$$

(b) This proof also proceeds by induction on the structure of the program $s$, but now for programs $s$ in $\mathcal{L}_{\text{pnif}}$. Again only a few typical cases are treated.

- $$\begin{aligned} \mathcal{D}(x := e)(\pi \cup \pi') &= [\text{def. 6.7.9}] \quad (\pi \cup \pi')[\mathcal{V}(e)/x] \\ &= [\text{def. 6.7.5}] \quad \pi[\mathcal{V}(e)/x] \cup \pi'[\mathcal{V}(e)/x] \\ &= [\text{def. 6.7.9}] \quad \mathcal{D}(x := e)(\pi) \cup \mathcal{D}(x := e)(\pi') \end{aligned}$$

- $$\begin{aligned} &\mathcal{D}(s_1 \ ; \ s_2)(\pi \cup \pi') \\ &= [\text{def. 6.7.9}] \quad \mathcal{D}(s_2)(\mathcal{D}(s_1)(\pi \cup \pi')) \\ &= [\text{ind. hyp.}] \quad \mathcal{D}(s_2)(\mathcal{D}(s_1)(\pi) \cup \mathcal{D}(s_1)(\pi')) \\ &= [\text{ind. hyp.}] \quad \mathcal{D}(s_2)(\mathcal{D}(s_1)(\pi)) \cup \mathcal{D}(s_2)(\mathcal{D}(s_1)(\pi')) \\ &= [\text{def. 6.7.9}] \quad \mathcal{D}(s_1 \ ; \ s_2)(\pi) \cup \mathcal{D}(s_1 \ ; \ s_2)(\pi') \end{aligned}$$

- $$\begin{aligned} &\mathcal{D}(s_1 \square s_2)(\pi \cup \pi') \\ &= [\text{def. 6.7.9}] \quad \mathcal{D}(s_1)(\pi \cup \pi') \cup \mathcal{D}(s_2)(\pi \cup \pi') \\ &= [\text{ind. hyp.}] \quad \mathcal{D}(s_1)(\pi) \cup \mathcal{D}(s_1)(\pi') \cup \mathcal{D}(s_2)(\pi) \cup \mathcal{D}(s_2)(\pi') \\ &= [\text{def. 6.7.9}] \quad \mathcal{D}(s_1 \square s_2)(\pi) \cup \mathcal{D}(s_1 \square s_2)(\pi') \end{aligned}$$

(c) As there are only finitely many options in a nondeterministic state this is a direct consequence of part (b). □

The first part of this lemma states that the semantics of $\mathcal{L}_{\text{pnif}}$ is a conservative extension of the semantics for $\mathcal{L}_{\text{pif}}$. The second part states that the denotational semantics is 'linear' in the nondeterministic state. If the nondeterministic start state is the union of the other states $\pi$ and $\pi'$, the end state given by the semantics can be split into the end state for $\pi$ and the end state for $\pi'$. The last part of the lemma states that only the currently selected option is relevant for the semantics of the program to be executed. Any other nondeterministic options that could have been nondeterministically selected in the execution so far do not influence the computation for the currently selected option.

### 6.7.2 A Hoare-style logic for $\mathcal{L}_{\mathrm{pnif}}$

The goal of this subsection is to develop a Hoare-style logic $pH_{nd}$ to reason about the nondeterministic programs in $\mathcal{L}_{\mathrm{pnif}}$, similar to the logic *pH* introduced in section 6.4. The logic allows verification of programs in $\mathcal{L}_{\mathrm{pnif}}$. The verification addresses the question: "Given that property $p$ holds before executing the program, does property $q$ hold after the execution of a program, no matter how the nondeterministic choices are made ?". This is the usual interpretation of nondeterminism in Hoare logic (see e.g. [126, 34]). This corresponds to looking at the worst case behavior of a program. It is also possible to study the best case behavior of a program by looking at questions of the form "can the nondeterministic choices be made in a way such that property $p$ holds after the execution of the program ?". These questions could also be treated in this framework but require a different definition of validity of the predicates, and a straightforward adaptation of the rule for nondeterminism in the logic.

The properties that are to be checked in the verification of the nondeterministic programs in $\mathcal{L}_{\mathrm{pnif}}$ are the same properties as for the programs without nondeterminism in $\mathcal{L}_{\mathrm{pif}}$ and $\mathcal{L}_{\mathrm{pw}}$. The same predicates can therefore be used to express these properties. Recall that the set of predicates, denoted by *Pred* and ranged over by $p$ and $q$, is given by

$$p \quad = \quad c_r \mid p \wedge p \mid p \vee p \mid \exists j : p \mid \forall j : p \mid \rho \cdot p \mid p + p \mid p \oplus_\rho p \mid c?p$$

with $c_r$ a real condition in $RC$, $\mathtt{j}$ an integer valued or real valued variable in $IVar \cup RVar$, $\rho$ a ratio in $(0,1)$ and $c$ a boolean condition in $BC\langle PVar\rangle$.

As we are interested in properties which hold for all possible ways of making the nondeterministic choices, a property is said to hold in some nondeterministic state exactly when the property holds for all the probabilistic substates in the nondeterministic state.

**Definition 6.7.14** *A nondeterministic state $\pi$ with an interpretation $J$ is said to satisfy the predicate $p$, denoted $(\pi, J) \models p$, when all substates of $\pi$ (with interpretation $J$) satisfy $p$, i.e.*

$$(\pi, J) \models p \quad \textit{when} \quad (\theta, J) \models p \textit{ for all } \theta \in \pi$$

When one is interested in the best case behavior of a program, this definition should be adapted. In that case one should check that the predicate $p$ holds for some substate instead of for all substates.

Note that the claim "$(\pi, J) \models p \vee q$" is weaker than the claim "$(\pi, J) \models p$ or $(\pi, J) \models q$". In the second case, all elements of $\pi$ must satisfy $p$ or all elements must satisfy $q$. For the first case it is also possible that some states satisfy $p$ but not $q$ and some satisfy $q$ but not $p$.

**Example 6.7.15** *Let $J$ be any interpretation. The probabilistic state $1\langle \mathrm{x} = 1\rangle$ satisfies the predicate $\mathbb{P}(\mathrm{x} = 1) = 1$ and therefore also the predicate $\mathbb{P}(\mathrm{x} = 1) = 1 \vee \mathbb{P}(\mathrm{x} = 2) = 1$. Thus we have*

$$(1\langle \mathrm{x} = 1\rangle, J) \;\models\; \mathbb{P}(\mathrm{x} = 1) = 1 \vee \mathbb{P}(\mathrm{x} = 2) = 1$$

*The probabilistic state* $1\langle \mathrm{x} = 2\rangle$ *satisfies the predicate* $\mathbb{P}(\mathrm{x} = 2) = 1$ *and therefore also the predicate* $\mathbb{P}(\mathrm{x} = 1) = 1 \vee \mathbb{P}(\mathrm{x} = 2) = 1$*, i.e.*

$$(1\langle \mathrm{x} = 2\rangle, J) \models \mathbb{P}(\mathrm{x} = 1) = 1 \vee \mathbb{P}(\mathrm{x} = 2) = 1$$

*The nondeterministic state* $\pi = \{\, 1\langle \mathrm{x} = 1\rangle, 1\langle \mathrm{x} = 2\rangle \,\}$ *satisfies* $\mathbb{P}(\mathrm{x} = 1) = 1 \vee \mathbb{P}(\mathrm{x} = 2) = 1$ *as all substates of* $\pi$ *satisfy this predicate, therefore*

$$(\{\, 1\langle \mathrm{x} = 1\rangle, 1\langle \mathrm{x} = 2\rangle \,\}, J) \models \mathbb{P}(\mathrm{x} = 1) = 1 \vee \mathbb{P}(\mathrm{x} = 2) = 1$$

*The state* $\pi = \{\, 1\langle \mathrm{x} = 1\rangle, 1\langle \mathrm{x} = 2\rangle \,\}$ *does not satisfy* $\mathbb{P}(\mathrm{x} = 1) = 1$ *as the substate* $1\langle \mathrm{x} = 2\rangle$ *does not satisfy this predicate. The state* $\pi$ *also does not satisfy* $\mathbb{P}(\mathrm{x} = 2) = 1$ *as the substate* $1\langle \mathrm{x} = 1\rangle$ *does not satisfy this predicate:*

$$\begin{aligned}(\{\, 1\langle \mathrm{x} = 1\rangle, 1\langle \mathrm{x} = 2\rangle \,\}, J) &\not\models \mathbb{P}(\mathrm{x} = 1) = 1\\ (\{\, 1\langle \mathrm{x} = 1\rangle, 1\langle \mathrm{x} = 2\rangle \,\}, J) &\not\models \mathbb{P}(\mathrm{x} = 2) = 1\end{aligned}$$

Recall from section 6.4 that a Hoare triple $\{\, p \,\}\ s\ \{\, q \,\}$ states that $p$ is a precondition and $q$ is a postcondition for the program $s$. A Hoare triple is valid if execution of the program starting from any state that satisfies the precondition leads to a state satisfying the postcondition. The formal definition of validity of a Hoare triple is as in section 6.4, but now using nondeterministic states in $\Pi$.

**Definition 6.7.16** *A Hoare triple,* $\{\, p \,\}\, s\, \{\, q \,\}$*, is said to be valid, denoted* $\models \{\, p \,\}\, s\, \{\, q \,\}$*, when*

$$(\pi, J) \models p \Rightarrow (\mathcal{D}(s)(\pi), J) \models q$$

*holds for all states* $\pi$ *in* $\Pi$ *and interpretations* $J$ *in* $\mathcal{J}$*.*

The observation that $(\pi, J) \models p \vee q$ is not equivalent with $(\pi, J) \models p$ or $(\pi, J) \models q$ shows that one needs to be careful when reasoning about predicates over nondeterministic states. For instance the rule (Or), which is obviously true in the purely probabilistic setting, requires more attention in the nondeterministic setting. That the rule remains valid is because the computation can be done for each probabilistic substate separately. This also means that for validity of a Hoare triple it is possible to restrict the start state to a set containing a single probabilistic substate.

**Lemma 6.7.17** *A Hoare triple* $\{\, p \,\}\ s\ \{\, q \,\}$ *is valid if and only if for all probabilistic states* $\theta$ *in* $\Theta$ *and interpretations* $J$ *in* $\mathcal{J}$*:* $(\theta, J) \models p \Rightarrow (\mathcal{D}(s)(\{\, \theta \,\}), J) \models q$*.*

**Proof** As $(\theta, J) \models p$ holds exactly when $(\{\, \theta \,\}, J) \models p$ holds and $\{\, \theta \,\}$ is a nondeterministic state, the 'only if' part of this lemma is immediately clear.

For the 'if' part of this lemma assume that $(\theta, J) \models p \Rightarrow (\mathcal{D}(s)(\{\, \theta \,\}), J) \models q$ holds for all $\theta$ in $\Theta$. If $(\pi, J) \models p$ then $(\theta, J) \models p$ for all $\theta$ in $\pi$. But then $(\mathcal{D}(s)(\{\, \theta \,\}), J) \models q$ for all $\theta$ in $\pi$. As $\mathcal{D}(s)(\pi) = \cup \{\, \mathcal{D}(s)(\{\, \theta \,\}) \mid \theta \in \pi \,\}$ (see lemma 6.7.13) this means that also $(\mathcal{D}(s)(\pi), J) \models q$. □

This lemma is used in theorem 6.7.20 below to show the soundness of the Hoare-style logic $pH_{nd}$ for $\mathcal{L}_{\text{pnif}}$. This logic is obtained from the logic $pH$ given in section 6.4, by adding a rule to deal with programs built with the nondeterministic operator □.

**Definition 6.7.18** *The proof system pH$_{nd}$ contains the rules of the proof system pH given in section 6.4 and additionally the rule*

$$\frac{\{\,p\,\}\ s\ \{\,q\,\}\qquad \{\,p\,\}\ s'\ \{\,q\,\}}{\{\,p\,\}\ s \,\Box\, s'\ \{\,q\,\}}\ \text{(Nd)}$$

The rule (Nd) has the same form as the rule for nondeterminism in standard Hoare logic. If a start state satisfying $p$ leads to an end state satisfying $q$ for both the execution of the program $s$ and the program $s'$ then it will also lead to an end state satisfying $q$ after executing the program $s \,\Box\, s'$ because the execution of $s \,\Box\, s'$ will either be the execution of $s$ or the execution of $s'$.

**Example 6.7.19** *Figure 6.7.19 below shows a proof tree in the system pH. This proof tree deduces the Hoare triple* $\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ \texttt{x := 1} \,\Box\, (\texttt{x := 2} \,\Box\, \texttt{x := 3})\ \{\,(\mathbb{P}(\texttt{x} = 1) = 1) \vee (\mathbb{P}(\texttt{x} = 2) = 1) \vee (\mathbb{P}(\texttt{x} = 3) = 1)\,\}$.

*As another example we consider the program* $s = (\texttt{x := 1} \oplus_{\frac{1}{2}} \texttt{x := 2}) \,\Box\, (\texttt{x := 2} \oplus_{\frac{1}{2}} \texttt{x := 3})$. *Using rules (Assign) and (Cons) (as in figure6.7.19) gives*

$$\vdash \{\,\mathbb{P}(\texttt{true}) = 1\,\}\ \texttt{x := } n\ \{\,\mathbb{P}(\texttt{x} = n) = 1\,\}$$

*for* $n = 1, 2, 3$. *By applying rules (Prob) and rule (Cons) next one obtains*

$$\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ \texttt{x := 1} \oplus_{\frac{1}{2}} \texttt{x := 2}\ \{\,(\mathbb{P}(\texttt{x} = 1) = \tfrac{1}{2}) \wedge (\mathbb{P}(\texttt{x} = 2) = \tfrac{1}{2})\,\}$$

*and*

$$\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ \texttt{x := 2} \oplus_{\frac{1}{2}} \texttt{x := 3}\ \{\,(\mathbb{P}(\texttt{x} = 2) = \tfrac{1}{2}) \wedge (\mathbb{P}(\texttt{x} = 3) = \tfrac{1}{2})\,\}$$

*Again using rule (Cons) gives both*

$$\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ \texttt{x := 1} \oplus_{\frac{1}{2}} \texttt{x := 2}\ \{\,\mathbb{P}(\texttt{x} = 2) = \tfrac{1}{2}\,\}$$

*and*

$$\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ \texttt{x := 2} \oplus_{\frac{1}{2}} \texttt{x := 3}\ \{\,\mathbb{P}(\texttt{x} = 2) = \tfrac{1}{2}\,\}$$

*From these Hoare triples we obtain for the program s the triple*

$$\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ s\ \{\,\mathbb{P}(\texttt{x} = 2) = \tfrac{1}{2}\,\}$$

*by using rule (Nd).*

*By similar reasoning we can deduce the triples* $\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ s\ \{\,\mathbb{P}(\texttt{x} \geq 2) \geq \tfrac{1}{2}\,\}$ *and* $\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ s\ \{\,\mathbb{P}(\texttt{x} \geq 1) = 1\,\}$ *using the proof system pH$_{nd}$.*

It is also possible to give the rule (Nd) more in the style of rule (Prob). It is not necessary to extend the predicates with an operator $\Box$ similar to the operator $\oplus_\rho$. Such an operator $\Box$ is equivalent with the operator $\vee$ which is already present. Replacing the rule (Nd) with the rule (Nd') below gives an equivalent logic.

$$\frac{\{\,p\,\}\ s\ \{\,q\,\}\qquad \{\,p\,\}\ s'\ \{\,q'\,\}}{\{\,p\,\}\ s \,\Box\, s'\ \{\,q \vee q'\,\}}\ \text{(Nd')}$$

$$\dfrac{\dfrac{\dfrac{}{\{\,\mathbb{P}(1 = 1) = 1\,\}\ x := 1\ \{\,\mathbb{P}(x = 1) = 1\,\}}(\text{Assign})}{\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ x := 1\ \{\,p\,\}}(\text{Cons}) \qquad \dfrac{\dfrac{\dfrac{}{\{\,\mathbb{P}(2 = 2) = 1\,\}\ x := 2\ \{\,\mathbb{P}(x = 2) = 1\,\}}(\text{Assign})}{\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ x := 1\ \{\,p\,\}}(\text{Cons}) \quad \dfrac{\dfrac{}{\{\,\mathbb{P}(3 = 3) = 1\,\}\ x := 3\ \{\,\mathbb{P}(3 = 3) = 1\,\}}(\text{Assign})}{\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ x := 2\ \{\,p\,\}}(\text{Cons})}{\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ x := 2 \,\Box\, x := 3\ \{\,p\,\}}(\text{Nd})}{\{\,\mathbb{P}(\texttt{true}) = 1\,\}\ x := 1 \,\Box\, (x := 2 \,\Box\, x := 3)\ \{\,p\,\}}(\text{Nd})$$

with $p = \Big(\mathbb{P}(x = 1) = 1\Big) \vee \Big(\mathbb{P}(x = 2) = 1\Big) \vee \Big(\mathbb{P}(x = 3) = 1\Big).$

Figure 6.7.19, a proof tree in $pH_{nd}$

This rule is similar to the rule (Prob), with the only difference being that the operator $\vee$ is used instead of the operator $\oplus_\rho$. Not only is the form of the rules the same, but the operator $\vee$ can be linked to the operation for nondeterministic choice in the semantics (union) in the same way the operator $\oplus_\rho$ in the logic is linked to the semantical operation $\oplus_\rho$: A probabilistic state satisfies the predicate $p \oplus_\rho p'$ when the state can be written as the probabilistic sum of two states, the first satisfying $p$ and the second satisfying $p'$. A nondeterministic state satisfies the predicate $p \vee p'$ when the state can be written as the union of two states the first satisfying $p$ and the second satisfying $p'$.

It is easy to check that the the logic $pH_{nd}$ and the logic $pH'_{nd}$, where rule (Nd) is replaced by rule (Nd'), are equivalent. By taking $q' = q$ in rule (Nd') the rule (Nd) is obtained, so clearly anything deduced with rule (Nd) can be deduced with rule (Nd'). Any Hoare triple deduced with rule (Nd') can be deduced by using $q \vee q'$ as the postcondition in rule (Nd) and using rule (Cons).

When looking at the best case behavior of programs, the rule (Nd) needs to be split into two rules. Both rules derive $\{\,p\,\}\ s \,\Box\, s'\ \{\,q\,\}$. The first rule from the validity of $\{\,p\,\}\ s\ \{\,q\,\}$, the second from the assertion $\{\,p\,\}\ s'\ \{\,q\,\}$. For the alternative rule (Nd') it is sufficient to change the operator used to represent nondeterministic choice from $\vee$ to $\wedge$.

The notation $\vdash\ \{\,p\,\}\ s\ \{\,q\,\}$ is again used to denote that the Hoare triple $\{\,p\,\}\ s\ \{\,q\,\}$ can be deduced in the logic. Although only the rule (Nd) is added and all other rules remain the same, the validity of the other rules needs to be rechecked as they are now used for predicates on nondeterministic states instead of predicates on probabilistic states. The following theorem states that the extended proof system $pH_{nd}$ is sound, i.e. that only valid Hoare triples can be obtained from this system.

**Theorem 6.7.20** *The proof system $pH_{nd}$ is sound, i.e. for each program $s$ in $\mathcal{L}_{\mathrm{pnif}}$ and for all predicates $p$ and $q$, $\vdash\ \{\,p\,\}\ s\ \{\,q\,\}$ implies $\models\ \{\,p\,\}\ s\ \{\,q\,\}$.*

**Proof** The proof uses induction on the depth of the proof tree for the Hoare triple $\{\,p\,\}\ s\ \{\,q\,\}$. A few typical cases are given below. In the reasoning below a fixed interpretation $J$ is assumed and a state will be said to satisfy a predicate instead of saying that the state together with the interpretation $J$ satisfies the predicate.

- If the rule (Nd) is used to derive $\vdash\ \{\,p\,\}\ s\Box s'\ \{\,q\,\}$ from $\vdash\ \{\,p\,\}\ s\ \{\,q\,\}$ and $\vdash\ \{\,p\,\}\ s'\ \{\,q\,\}$ then $\models\ \{\,p\,\}\ s\ \{\,q\,\}$ and $\models\ \{\,p\,\}\ s'\ \{\,q\,\}$ by induction.

  If $\pi$ satisfies $p$ then both $\mathcal{D}(s)(\pi)$ and $\mathcal{D}(s')(\pi)$ satisfy $q$. But then $\mathcal{D}(s \,\Box\, s')(\pi) = \mathcal{D}(s)(\pi) \cup \mathcal{D}(s')(\pi)$ also satisfies $q$.

- If the rule (Assign) is used to derive $\vdash\ \{\,p[e/x]\,\}\ x := e\ \{\,p\,\}$ and $\theta$ satisfies $p[e/x]$ then the state $\theta[\mathcal{V}(e)/x]$ satisfies $p$ (see lemma 6.4.12). But then also $\mathcal{D}(x := e)(\{\,\theta\,\}) = \{\,\theta[\mathcal{V}(e)/x]\,\}$ satisfies $p$. Using lemma 6.7.17 this is sufficient for the validity of the Hoare triple $\{\,p[e/x]\,\}\ x := e\ \{\,p\,\}$.

- If the rule (Or) is used to derive $\vdash\ \{\,p \vee p'\,\}\ s\ \{\,q\,\}$ from $\vdash\ \{\,p\,\}\ s\ \{\,q\,\}$ and $\vdash\ \{\,p'\,\}\ s\ \{\,q\,\}$ then $\models\ \{\,p\,\}\ s\ \{\,q\,\}$ and $\models\ \{\,p'\,\}\ s\ \{\,q\,\}$ by induction.

  If a probabilistic state $\theta$ satisfies $p \vee p'$ then it satisfies $p$ or it satisfies $p'$. (Note that this is not always the case for a nondeterministic state.) As $\models\ \{\,p\,\}\ s\ \{\,q\,\}$ and

$\models \{ p' \} \; s \; \{ q \}$, in both cases $\mathcal{D}(s)(\{ \theta \})$ will satisfy $q$. The validity of the Hoare triple $\{ p \vee p' \} \; s \; \{ q \}$ now follows by lemma 6.7.17.

- If rule (Prob) is used to derive $\vdash \; \{ p \} \; s \oplus_\rho s' \; \{ q \oplus_\rho q' \}$ from $\vdash \; \{ p \} \; s \; \{ q \}$ and $\vdash \; \{ p \} \; s' \; \{ q' \}$ then by induction $\models \; \{ p \} \; s \; \{ q \}$ and $\models \; \{ p \} \; s' \; \{ q' \}$.

  If $\theta$ satisfies $p$ then $\mathcal{D}(s)(\{ \theta \})$ satisfies $q$ and $\mathcal{D}(s')(\{ \theta \})$ satisfies $q'$. But then $\mathcal{D}(s \oplus_\rho s')(\{ \theta \}) = \mathcal{D}(s)(\{ \theta \}) \oplus_\rho \mathcal{D}(s')(\{ \theta \})$ satisfies $q \oplus_\rho q'$ as clearly each probabilistic state in $\mathcal{D}(s)(\{ \theta \}) \oplus_\rho \mathcal{D}(s')(\{ \theta \})$ can be written as the probabilistic combination of a probabilistic state satisfying $q$ and a probabilistic state satisfying $q'$. Using lemma 6.7.17 this is sufficient for the validity of the Hoare triple $\{ p \} \; s \oplus_\rho s' \; \{ q \oplus_\rho q' \}$. □

As the proof system is sound, Hoare triples deduced with the system are valid. Without soundness the proof system is, of course, useless. Soundness gives that the proof system is at least useful for Hoare triples which can be deduced. The next question is whether the proof system is also complete. Can all valid Hoare triples be deduced using the proof system? The following subsection addresses this question.

### 6.7.3 Weakest preconditions and completeness

In this subsection the conjecture that all valid Hoare triples, with a slight restriction on the postcondition, can be obtained using the proof system $pH_{nd}$ is discussed. The restriction is the same as the restriction introduced in section 6.5.

The completeness of the proof system is shown by using weakest preconditions. For a postcondition $q$ and a program $s$ the weakest precondition $p$ that yields a valid Hoare triple $\{ p \} \; s \; \{ q \}$ is found. Next it is shown that the Hoare triple $\{ p \} \; s \; \{ q \}$ can be deduced in the proof system. Using the rule of consequence (Cons) this gives that any valid Hoare triple with $q$ as a postcondition can be deduced.

It is not clear how to define the weakest precondition for a general nondeterministic program in $\mathcal{L}_{\text{pnif}}$ directly. Instead, the weakest precondition is only given directly for a subset of $\mathcal{L}_{\text{pnif}}$ and for each program an equivalent program in this subset is given. Recall that two programs are called equivalent if they have the same denotational semantics. Clearly the weakest precondition for two equivalent programs is the same.

In section 6.5 the function $\wp$ was introduced and shown to give the weakest precondition for programs in $\mathcal{L}_{\text{pif}}$. To be able to give the weakest precondition of a program of the form $s = s_1 \oplus_\rho s_2$, the equivalence $\wp(s)(q \vee q') = \wp(s)(q) \vee \wp(s)(q')$ was used in the definition of $\wp$ in section 6.5. For predicates over a nondeterministic state, however, the weakest precondition does not satisfy this equation in general. The weakest precondition for $q$ after $s$ or the weakest precondition for $q'$ after $s$ is a precondition for $q \vee q'$ after $s$, but not the weakest.

**Example 6.7.21** *Let the program $s$ be given by* `x := 1` $\Box$ `x := 2` *and the predicates $p$ and $p'$ by $\mathbb{P}(x = 1) = 1$ and $\mathbb{P}(x = 2) = 1$ respectively, then the weakest precondition for $p \vee p'$ after $s$ is $\mathbb{P}(\texttt{true}) = 1$. (See definition 6.7.9 and example 6.7.19.) The weakest precondition for $p$ after $s$ is* `false` *as the option* `x := 2` *can be selected. Similarly the weakest precondition for $p'$ after $s$ is also* `false` *as the option* `x := 1` *can be selected. The predicate* `false` $\vee$ `false` *is not the weakest precondition for $p \vee p'$ after $s$.*

Not being able to use the same construction as in section 6.5, it is not clear how to give a weakest precondition for all nondeterministic programs directly. Instead a structural definition of the weakest precondition is only given for nondeterministic programs in a specific form. Other programs are reduced to an equivalent program of this form. The equivalence with the reduced program is then used to give the weakest precondition indirectly.

In sections 4.2 and 4.4 linear models are given where all nondeterministic choices are made at the start of the execution, leaving only purely probabilistic programs, i.e. programs in which all nondeterminism is caused by probabilistic choices. The same idea of moving nondeterministic choices to the start of the program is used here. By moving all nondeterministic choices to the start of the program, the execution will consist of first making all nondeterministic choices, referred to as resolving the nondeterminism, and then executing a purely probabilistic program. The weakest precondition for a purely probabilistic program can be found using the approach of section 6.5. The weakest precondition for the nondeterministic choice between several programs $s$ is simply the disjunction of the weakest preconditions for all of these programs: A property holds for a nondeterministic program if it holds for all ways of making the nondeterministic choices in the program. For a program $s$ which is the nondeterministic choice between several purely probabilistic programs $s_k$ $(k = 1, \dots n)$ this means that the weakest precondition $p$ for $s$ must imply the weakest precondition $p_k$ of each of the programs $s_k$. The weakest predicate that satisfies this property is the disjunction of the predicates $p_k$.

The approach to giving the weakest precondition of a nondeterministic program is, thus, as follows. First find an equivalent program with all nondeterministic choices at the start of the program. Find the weakest precondition for all purely probabilistic alternatives and take the disjunction of these preconditions.

The function *red* returns for every program an equivalent program with all nondeterministic choices at the start of the program. A program $s$ in $\mathcal{L}_{\mathrm{pnif}}$ is called purely probabilistic when there are no nondeterministic choices in $s$, i.e. when $s$ is also a program in $\mathcal{L}_{\mathrm{pif}}$. Note that if $s$ is not purely probabilistic then $red(s) = s_1 \,\Box\, s_2$ for some programs $s_1$, $s_2$.

**Definition 6.7.22** *The reduction function* $red : \mathcal{L}_{\mathrm{pnif}} \to \mathcal{L}_{\mathrm{pnif}}$ *is given by*

$$\begin{array}{rcl} red(\texttt{skip}) & = & \texttt{skip} \\ red(x \texttt{ := } e) & = & x \texttt{ := } e \\ red(s \,\Box\, s') & = & red(s) \,\Box\, red(s') \end{array}$$

*when* $red(s) = s_1 \,\Box\, s_2$ *then*

$$\begin{array}{rcl} red(s \texttt{ ; } s') & = & red(s_1 \texttt{ ; } s') \,\Box\, red(s_2 \texttt{ ; } s') \\ red(s \oplus_\rho s') & = & red(s_1 \oplus_\rho s') \,\Box\, red(s_2 \oplus_\rho s') \\ red(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi}) & = & red(\texttt{if } c \texttt{ then } s_1 \texttt{ else } s' \texttt{ fi}) \\ & & \Box\; red\,(\texttt{if } c \texttt{ then } s_2 \texttt{ else } s' \texttt{ fi}) \end{array}$$

*when $s$ is purely probabilistic and $red(s') = s_1 \,\Box\, s_2$ then*

$$\begin{array}{rcl} red(s\ ;\ s') &=& red(s\ ;\ s_1) \,\Box\, red(s\ ;\ s_2) \\ red(s \oplus_\rho s') &=& red(s \oplus_\rho s_1) \,\Box\, red(s \oplus_\rho s_2) \\ red(\texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{fi}) &=& red(\texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s_1\ \texttt{fi}) \\ && \Box\ red\,(\texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s_2\ \texttt{fi}) \end{array}$$

*finally when both $s$ and $s'$ are purely probabilistic*

$$\begin{array}{rcl} red(s\ ;\ s') &=& s\ ;\ s' \\ red(s \oplus_\rho s') &=& s \oplus_\rho s' \\ red(\texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{fi}) &=& \texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{fi} \end{array}$$

The execution of a program $red(s)$ starts with a number of nondeterministic choices. After the nondeterministic choices at the start of the program have been made, the remaining program is a purely probabilistic program in $\mathcal{L}_{\text{pif}}$.

**Lemma 6.7.23** *The function red is well-defined.*

**Proof** By induction on the weight function $wgt$ given by

$$\begin{array}{rcl} wgt(\texttt{skip}) &=& 2 \\ wgt(x := e) &=& 2 \\ wgt(s\ ;\ s') &=& wgt(s) \cdot wgt(s') \\ wgt(s \oplus_\rho s') &=& wgt(s) \cdot wgt(s') \\ wgt(\texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{fi}) &=& wgt(s) \cdot wgt(s') \\ wgt(s \,\Box\, s') &=& wgt(s) + wgt(s') + 1 \end{array}$$

one shows that

- $red(s)$ is well-defined,
- $red(s) = s_1 \,\Box\, s_2$ or $s$ is in $\mathcal{L}_{\text{pif}}$ and
- $wgt(red(s)) < wgt(s)$ whenever $red(s) \neq s$.

We only consider two representative cases

[$s \,\Box\, s'$] As $wgt(s)$ and $wgt(s')$ are less than $wgt(s \,\Box\, s')$ we have that $red(s)$ and $red(s')$ and therefore also $red(s \,\Box\, s')$ are well-defined. Clearly the second point mentioned above holds as $red(s \,\Box\, s') = red(s) \,\Box\, red(s')$ Finally we check the weight of $red(s)$. We have $wgt(red(s \Box s')) = wgt(red(s) \Box red(s')) = wgt(red(s)) + wgt(red(s')) + 1$. As $wgt(s \,\Box\, s') = wgt(s) + wgt(s') + 1$ we are done by applying the induction hypothesis for $s$ and $s'$.

[$s\ ;\ s'$] We only treat the case that $s$ is purely probabilistic, i.e. $s \in \mathcal{L}_{\text{pif}}$, and $red(s') = s_1 \,\Box\, s_2$. We have that $wgt(s\ ;\ s_1) = wgt(s) \cdot wgt(s_1) < wgt(s) \cdot wgt(s_1\ ;\ s_2) \leq wgt(s) \cdot wgt(s') = wgt(s\ ;\ s')$ by applying the third property in the induction hypothesis for $s'$ and similarly $wgt(s\ ;\ s_2) < wgt(s\ ;\ s')$. From this well-definedness is clear. Note that the fact that each weight is at least 2 is essential here. The second property is clear leaving only the checking of the weight of $red(s\ ;\ s')$.

$$
\begin{array}{ll}
& \mathit{wgt}(s\ ;\ s') \\
= & \mathit{wgt}(s) \cdot \mathit{wgt}(s') \\
\geq & [\text{ind. hyp. } s'] \quad \mathit{wgt}(s) \cdot \mathit{wgt}(\mathit{red}(s')) \\
= & \mathit{wgt}(s) \cdot \mathit{wgt}(s_1 \,\square\, s_2) \\
= & \mathit{wgt}(s) \cdot (\mathit{wgt}(s_1) + \mathit{wgt}(s_2) + 1) \\
> & \mathit{wgt}(s) \cdot \mathit{wgt}(s_1) + \mathit{wgt}(s) \cdot \mathit{wgt}(s_2) + 1 \\
= & \mathit{wgt}(s\ ;\ s_1) + \mathit{wgt}(s\ ;\ s_2) + 1 \\
\geq & [\text{ind. hyp. } s\ ;\ s_i] \quad \mathit{wgt}(\mathit{red}(s\ ;\ s_1)) + \mathit{wgt}(\mathit{red}(s\ ;\ s_2)) + 1 \\
= & \mathit{wgt}(\mathit{red}(s\ ;\ s_1) \,\square\, \mathit{red}(s\ ;\ s_2)) \\
= & \mathit{wgt}(\mathit{red}(s\ ;\ s')) \qquad \square
\end{array}
$$

The program $\mathit{red}(s)$ describes a situation where the user or opponent, who makes the nondeterministic choices, has to choose a strategy ahead of time. The choices available for the user are the same as in $s$, only the moment at which they are made are different. As probabilistic choices in the program are not affected by any other choices in the program, the different moments of nondeterministic choice in $\mathit{red}(s)$ and $s$ will not affect the probabilistic choices. For the end state, the moment a nondeterministic choice is made is also irrelevant. The programs $s$ and $\mathit{red}(s)$ are, therefore, equivalent.

**Lemma 6.7.24** *For each program $s$, the denotational semantics of $s$ is equal to the denotational semantics of* $\mathrm{red}(s)$*.*

**Proof** This property is shown by induction on structure of $\mathit{red}(s)$. The analysis is split into the different cases possible for $s$. The cases `skip` and $x$ `:=` $e$ are directly clear and the case $s_1 \,\square\, s_2$ is immediate from the induction assumption. The remaining cases for sequential composition, probabilistic choice and conditional choice are similar. Only the case $s_1 \oplus_\rho s_2$ is given.

- If $\mathit{red}(s_1) = s_1' \,\square\, s_1''$ then

$$
\begin{array}{rcl}
\mathcal{D}(\mathit{red}(s_1)) & = & \mathcal{D}(\mathit{red}(s_1' \oplus_\rho s_2) \,\square\, \mathit{red}(s_1'' \oplus_\rho s_2)) \\
& = & \mathcal{D}(\mathit{red}(s_1' \oplus_\rho s_2)) \cup \mathcal{D}(\mathit{red}(s_1'' \oplus_\rho s_2)) \\
\text{[ind. hyp.]} & = & \mathcal{D}(s_1' \oplus_\rho s_2) \cup \mathcal{D}(s_1'' \oplus_\rho s_2) \\
& = & (\mathcal{D}(s_1') \oplus_\rho \mathcal{D}(s_2)) \cup (\mathcal{D}(s_1'') \oplus_\rho \mathcal{D}(s_2)) \\
\text{[straigthforw. calc.]} & = & (\mathcal{D}(s_1') \cup \mathcal{D}(s_1'')) \oplus_\rho \mathcal{D}(s_2) \\
& = & (\mathcal{D}(s_1' \,\square\, s_1'')) \oplus_\rho \mathcal{D}(s_2) \\
& = & \mathcal{D}(\mathit{red}(s_1)) \oplus_\rho \mathcal{D}(s_2) \\
\text{[ind. hyp.]} & = & \mathcal{D}(s_1) \oplus_\rho \mathcal{D}(s_2) \\
& = & \mathcal{D}(s_1 \oplus_\rho s_2)
\end{array}
$$

If $s_1$ is purely probabilistic and $\mathit{red}(s_2) = s_2' \,\square\, s_2''$ then symmetrically $\mathcal{D}(\mathit{red}(s_1 \oplus_\rho s_2)) = \mathcal{D}(s_1 \oplus_\rho s_2)$, and if both $s_1$ and $s_2$ are purely probabilistic then $\mathit{red}(s_1 \oplus_\rho s_2) = s_1 \oplus_\rho s_2$ so clearly $\mathcal{D}(\mathit{red}(s_1 \oplus_\rho s_2))(\pi) = \mathcal{D}(s_1 \oplus_\rho s_2)(\pi)$. $\square$

As an immediate consequence of this lemma we have that, by the definition of validity, $s$ and $\mathit{red}(s)$ form valid Hoare triples with the same predicates: For all programs $s$ in $\mathcal{L}_{\text{pnif}}$ and all predicates $p$, $q$ in $\mathit{Pred}$

$$\models \{\, p \,\}\ s\ \{\, q \,\} \quad \text{if and only if} \quad \models \{\, p \,\}\ \mathit{red}(s)\ \{\, q \,\}$$

This also means that the weakest precondition for program $s$ and postcondition $q$ is the same as the weakest precondition for the program $\mathit{red}(s)$ and postcondition $q$. Hence, the weakest precondition needs to be defined only for programs which are the nondeterministic choice between programs in $\mathcal{L}_{\text{pif}}$. For a program $s$ not of this form, the weakest precondition is defined as the weakest precondition of $\mathit{red}(s)$.

**Definition 6.7.25** *The weakest precondition $\wp_{nd}(s)(q)$ for a postcondition $q$ and a program $s$ in $\mathcal{L}_{\text{pnif}}$ is given by*

$$\begin{array}{rcl} \wp_{nd}(s)(q) & = & \wp'_{nd}(\mathit{red}(s))(q) \\ \wp'_{nd}(s \,\Box\, s')(q) & = & \wp'_{nd}(s)(q) \wedge \wp'_{nd}(s')(q) \\ \wp'_{nd}(s)(q) & = & \wp(s)(q) \quad \textit{if } s \textit{ purely probabilistic} \end{array}$$

*with $\wp$ as introduced in definition 6.5.2.*

The function $\wp_{nd}$ gives the weakest precondition for any program $s$ and postcondition $q$. For a program which is a nondeterministic choice between purely probabilistic programs in $\mathcal{L}_{\text{pif}}$, the function $\wp'_{nd}$ gives the weakest precondition. To be certain that $q$ holds after executing $s \,\Box\, s'$ one must know that it holds after executing $s$ and after executing $s'$ as either of these two programs may be selected. For programs in $\mathcal{L}_{\text{pif}}$ the weakest precondition has already been given in section 6.5.

**Lemma 6.7.26** *For each program $s$ and predicate $q$, the predicate $p = \wp_{nd}(s)(q)$ is the weakest predicate for which $\{\, p \,\}\ s\ \{\, q \,\}$ is a valid Hoare triple.*

**Proof** For each program $s$, state $\pi$, interpretation $J$ and predicate $q$ the following property is shown: $(\pi, J) \models \wp_{nd}(s)(q) \iff (\mathcal{D}(s)(\pi), J) \models q$. This implies that $\wp_{nd}$ gives the weakest precondition.

By virtue of lemma 6.7.13 part (c) and the equivalence of $\mathit{red}(s)$ and $s$ it is sufficient to show the property $(\theta, J) \models \wp'_{nd}(s)(q) \iff (\mathcal{D}(s)(\{\,\theta\,\}), J) \models q$. for $s$ of the form $s_1 \,\Box\, s_2$ and $s \in \mathcal{L}_{\text{pif}}$. Using the results from section 6.5 for $\wp$ (lemma 6.5.6) together with part (a) of lemma 6.7.13 gives this property for all programs in $\mathcal{L}_{\text{pif}}$. If the property holds for $s$ and $s'$ then

$$\begin{array}{l} (\{\,\theta\,\}, J) \models \wp'_{nd}(s \,\Box\, s')(q) \\ \iff (\{\,\theta\,\}, J) \models \wp'_{nd}(s)(q) \wedge \wp'_{nd}(s')(q) \\ \iff (\theta, J) \models \wp'_{nd}(s)(q) \wedge \wp'_{nd}(s')(q) \\ \iff (\theta, J) \models \wp'_{nd}(s)(q) \text{ and } (\theta, J) \models \wp'_{nd}(s')(q) \\ \iff (\{\,\theta\,\}, J) \models \wp'_{nd}(s)(q) \text{ and } (\{\,\theta\,\}, J) \models \wp'_{nd}(s')(q) \\ \iff (\mathcal{D}(s)(\{\,\theta\,\}), J) \models q \text{ and } (\mathcal{D}(s')(\{\,\theta\,\}), J) \models q \\ \iff (\mathcal{D}(s)(\{\,\theta\,\}) \cup \mathcal{D}(s')(\{\,\theta\,\}), J) \models q \\ \iff (\mathcal{D}(s \,\Box\, s')(\{\,\theta\,\}), J) \models q \end{array}$$

so the property also holds for $s \Box s'$. $\Box$

The function $\wp_{nd}$,thus, gives the weakest precondition for any program and postcondition. To show completeness of the logic $pH_{nd}$ it is sufficient to show that the weakest precondition can be deduced from $pH_{nd}$.

**Lemma 6.7.27** *For each program $s$ in $\mathcal{L}_{\text{pnif}}$, predicate $p$ and restricted predicate $q$*

$$\models \{ p \}\ s\ \{ q \} \ \Rightarrow \vdash \{ p \}\ red(s)\ \{ q \}$$

**Proof** As $\wp_{nd}$ gives the weakest precondition, it is sufficient to show that we have that $\vdash \{ \wp_{nd}(s)(q) \}\ red(s)\ \{ q \}$ holds. By using rule (Cons) this gives that any valid Hoare triple can be deduced. For a purely probabilistic program this follows from lemma 6.5.5. For programs of the form $s \Box s'$ the rule (Nd) and the definition of weakest precondition can be used to reduce the problem to $\vdash \{ \wp_{nd}(s)(q) \}\ red(s)\ \{ q \}$ and $\vdash \{ \wp_{nd}(s')(q) \}\ red(s')\ \{ q \}$ which will lead to only purely probabilistic programs in finitely many steps. $\Box$

This lemma shows that any valid Hoare triple can be deduced by first reducing the program using the function *red*. Although we believe that all valid Hoare triples can also be deduced without reducing the program, no proof of this fact is available. An approach that can be followed is to try to show that $\vdash \{ p \}\ red(s)\ \{ q \}$ implies that $\vdash \{ p \}\ s\ \{ q \}$. The main problem is showing this property for programs $s$ of the form $s = s_1 \oplus_\rho s_2$.

**Conjecture 6.7.28** *The logic $pH_{nd}$ is complete, i.e. for each program $s$ in $\mathcal{L}_{\text{pnif}}$, predicate $p$ and restricted predicate $q$*

$$\models \{ p \}\ s\ \{ q \} \ \Rightarrow \vdash \{ p \}\ s\ \{ q \}$$

Note that, due to property derived in the lemma 6.7.27, we do know that the proof system $pH_{nd}$ extended with the rule

$$\frac{\{ p \}\ red(s)\ \{ q \}}{\{ p \}\ s\ \{ q \}}\ \text{(Reduce)}$$

is complete. Rules that change the structure of the program step by step, such as a rule that derives $\{ p \}\ (s_1 \Box s_2) \oplus_\rho s_3\ \{ q \}$ from $\{ p \}\ (s_1 \oplus_\rho s_3) \Box (s_2 \oplus_\rho s_3)\ \{ q \}$, can also be added to obtain completeness should the conjecture above be false.

## 6.7.4 An example: Three doors revisited

In this subsection the 'three doors problem', introduced in section 4.2, is studied as an example of an algorithm with nondeterminism and probability. Several possible strategies are analyzed using the system $pH_{nd}$.

In a quiz show a prize is hidden behind one of three doors. The contestant may pick one of the doors. After selecting a door, the quiz master will open one of the other two doors. (To show that the prize is not located there.) After this the contestant is again allowed to pick a door. The question is what is the best strategy for the contestant, stay with the first choice or switch doors.

The algorithm which describes the first part of the three doors problem is as follows. A variable *prize* describes the location of the prize. No information is available about where the prize is placed, so nondeterminism is used to describe the selection of the value of *prize*. The variable *door* describes the door selected by the contestant. After this the quiz master opens one of the other two doors. The doors that the quiz master can choose from depend on the door opened by the contestant. As the contestant does not know how the quiz master selects this door, this choice is also described by a nondeterministic choice.

```
Int$[1\ldots3]$ $prize, door, open$;
($prize$ := 1 $\square$ ($prize$ := 2 $\square$ $prize$ := 3));
($door$ := 1 $\oplus_{\frac{1}{3}}$ ($door$ := 2 $\oplus_{\frac{1}{2}}$ $door$ := 3));
if $door = 1$ then $open$ := 2 $\square$ $open$ := 3
else if $door = 2$ then $open$ := 1 $\square$ $open$ := 3
else $open$ := 1 $\square$ $open$ := 2 fi
fi;
```

The next step depends on the strategy the contestant uses. The easiest to describe is the strategy to always stay with the door selected at the start.

$$door := door$$

The strategy where the other closed door is always selected, can be described by the following program.

$$door := 6 - open - door$$

The algorithm is different from that given in section 4.2. The most obvious difference is that the use of variables and values replaces the use of atomic actions. There is also a more subtle difference which will show up during the analysis. The first part of the program is the same for all strategies.

As we have

$$\begin{array}{l} \{\, \mathbb{P}(\mathtt{true}) = 1 \,\} \Rightarrow \\ \{\, \exists \mathtt{i} : \mathbb{P}(3 = \mathtt{i}) = 1 \,\} \\ prize := 3 \\ \{\, \exists \mathtt{i} : \mathbb{P}(prize = \mathtt{i}) = 1 \,\} \end{array}$$

and similarly $\vdash \{\, \mathbb{P}(\mathtt{true}) = 1 \,\}\ prize := 2\ \{\, \exists \mathtt{i} : \mathbb{P}(prize = \mathtt{i}) = 1 \,\}$, using rule (Nd) gives $\vdash \{\, \mathbb{P}(\mathtt{true}) = 1 \,\}\ prize := 2 \,\square\, prize := 3\ \{\, \exists \mathtt{i} : \mathbb{P}(prize = \mathtt{i}) = 1 \,\}$. In the proof outline below, the rule (Nd) is used implicitly. For $j, k \in \{\, 1, 2, 3 \,\}$ the following shorthands are also used

$$\begin{array}{rcl} p_j & = & \mathbb{P}(prize = \mathtt{i} \wedge door = \mathtt{j}) = 1 \\ p_{j,k} & = & \mathbb{P}(prize = \mathtt{i} \wedge door = \mathtt{j} \wedge open = \mathtt{k}) = 1 \\ q & = & \frac{1}{3} \cdot (p_{1,2} \vee p_{1,3}) + \frac{1}{3} \cdot (p_{2,1} \vee p_{2,3}) + \frac{1}{3} \cdot (p_{3,1} \vee p_{3,2}) \end{array}$$

The analysis below shows that the predicate $\exists \mathtt{i} : q$ holds after execution of the first part of the program. Note that the predicate $\exists \mathtt{i} : q$ is equivalent with the predicate

$\exists \mathtt{i} : \forall \mathtt{j} : \exists \mathtt{k} \neq \mathtt{j} : \mathbb{P}(prize = \mathtt{i}, door = \mathtt{j}, open = \mathtt{k}) = \frac{1}{3}$. This second predicate describes the situation more intuitively: The prize is behind a door `i`, each door `j` can be selected by the contestant with equal probability and a door `k` that has not been selected by the contestant, $\mathtt{k} \neq \mathtt{j}$, is open. The predicate $\exists \mathtt{i} : q$ is used below as it is easier to calculate with this predicate,

```
Int[1...3] $prize, door, open$;
{ $\mathbb{P}(\mathtt{true}) = 1$ }
$(prize$ := 1 $\Box$ $(prize$ := 2 $\Box$ $prize$ := 3));
{ $\exists \mathtt{i} : \mathbb{P}(prize = \mathtt{i}) = 1$ }
```

By using the rule (Exists) the analysis can be restricted to some unknown but fixed value for `i`.

```
{ $\mathbb{P}(prize = \mathtt{i}) = 1$ }
$(door$ := 1 $\oplus_{\frac{1}{3}}$ $(door$ := 2 $\oplus_{\frac{1}{2}}$ $door$ := 3));
{ $\frac{1}{3} \cdot p_1 + \frac{1}{3} \cdot p_2 + \frac{1}{3} \cdot p_3$ }
if $door = 1$ then
    { $\frac{1}{3} \cdot (\mathbb{P}(prize = \mathtt{i} \wedge door = 2) = 1)$ }
    $open$ := 2 $\Box$ $open$ := 3
    { $\frac{1}{3} \cdot (\mathbb{P}(prize = \mathtt{i} \wedge door = 2 \wedge open = 1) = 1 \vee$
          $\mathbb{P}(prize = \mathtt{i} \wedge door = 2 \wedge open = 3) = 1)$ }
else if $door = 2$ then
    { $\frac{1}{3} \cdot p_2$ }
    $open$ := 1 $\Box$ $open$ := 3
    { $\frac{1}{3} \cdot (p_{2,1} \vee p_{2,3})$ }
else
    { $\frac{1}{3} \cdot p_3$ }
    $open$ := 1 $\Box$ $open$ := 2
    { $\frac{1}{3} \cdot (p_{3,1} \vee p_{3,2})$ }
fi;
{ $q$ }
```

After the first part of the program the property $\exists \mathtt{i} : q$ is satisfied. Clearly this property will still hold after executing *door* := *door*. Simplifying this property by forgetting the value of *open* gives that the predicate $\exists \mathtt{i} : \frac{1}{3} \cdot (\mathbb{P}(prize = \mathtt{i}, door = 1) = 1) + \frac{1}{3} \cdot (\mathbb{P}(prize = \mathtt{i}, door = 2) = 1) + \frac{1}{3} \cdot (\mathbb{P}(prize = \mathtt{i}, door = 3) = 1)$ holds. (Note that simply forgetting the value *open* in $\mathbb{P}(prize = \mathtt{i}, door = 1, open = 2) = 1$ only gives that $\mathbb{P}(prize = \mathtt{i}, door = 1) \geq 1$ holds, but clearly the probability cannot be greater than one.) As exactly one of the three parts will have that $door = \mathtt{i}$ this predicate implies $\mathbb{P}(prize = door) = \frac{1}{3}$. In other words, the probability of winning the prize with the first strategy is $\frac{1}{3}$.

After executing the program *door* := 6 $-$ *open* $-$ *door*, the predicate $\exists \mathtt{i} : \frac{1}{3} \cdot (p_{3,2} \vee p_{2,3}) + \frac{1}{3} \cdot (p_{3,1} \vee p_{1,3}) + \frac{1}{3} \cdot (p_{2,1} \vee p_{1,2})$ holds. This predicate implies $\exists.i : \frac{1}{3} \cdot (p_3 \vee p_2) + \frac{1}{3} \cdot (p_3 \vee p_1) + \frac{1}{3} \cdot (p_2 \vee p_1)$. For each possible value of `i`, it is always possible to have the value of *door* different from `i`, meaning that the door with the prize is not selected. The worst case probability for winning the prize is 0.

This result is different from result obtained in section 4.2. The result is different is because the assumption that the quiz master will not open the door with the prize behind it is not enforced here. By opening the door with the prize behind it (unless the player chooses this door) the prize will always be behind the initially selected door or the opened door, never behind the other closed door. This would of course not work in reality. If the quiz master opens the door with the prize, the contestant can see where the prize is and select this door. The strategy for the contestant, however, does not reflect this. There are several ways to deal with this problem. The assumption can be added as a predicate at this point. It is also possible to change the program such that it is impossible for the quiz master to open a door with a prize behind it. The solution that seems most natural, however, is to correct the strategy of the contestant. A correction of the strategy uses the fact that, although the contestant can in general not see the prize, i.e. cannot observe *prize*, the contestant can observe whether the prize is behind the opened door, i.e. can observe $\mathit{prize} = \mathit{open}$. A strategy described by program $s$ can be corrected to include this possibility by changing the program to `if` $\mathit{prize} = \mathit{open}$ `then` $\mathit{door}$ `:=` $\mathit{open}$ `else` $s$ `fi`.

For the first strategy, "stay with your door", the adjustment does not change the worst case probability. For the second strategy, "switch doors", however, it does. Because of the rules (Lin +), (Lin $\cdot$) and (And) the analysis can be restricted to predicates of the form $p_{j,k}$ instead of the more complicated predicate $q$.

```
{ $p_{1,2}$ }
if $\mathit{prize} = \mathit{open}$ then
    { $(\mathtt{i} = 2) \Rightarrow p_{1,2}$ }
    $\mathit{prize} = \mathit{open}$
    { $(\mathtt{i} = 2) \Rightarrow p_{2,2}$ }
    $\Rightarrow$ { $(\mathtt{i} = 2) \Rightarrow \mathbb{P}(\mathit{door} = \mathit{prize}) = 1$ }
else
    { $(\mathtt{i} \neq 2) \Rightarrow p_{1,2}$ }
    $\mathit{door}$ := 6 $- \mathit{open} - \mathit{door}$
    { $(\mathtt{i} \neq 2) \Rightarrow p_{3,2}$ }
fi
{ $((\mathtt{i} = 2) \Rightarrow \mathbb{P}(\mathit{door} = \mathit{prize}) = 1) + ((\mathtt{i} \neq 2) \Rightarrow p_{3,2})$ }
$\Rightarrow$ { $\mathbb{P}(\mathit{door} = \mathit{prize}) = 1 \vee (\mathtt{i} \neq 2 \wedge p_{3,2})$ }
$\Rightarrow$ { $\mathbb{P}(\mathit{door} = \mathit{prize}) = 1 \vee (\mathtt{i} \neq 2 \wedge p_{3})$ }
```

In the same way one obtains $\mathbb{P}(\mathit{door} = \mathit{prize}) = 1 \vee (\mathtt{i} \neq 3 \wedge p_2)$ starting from $p_{1,3}$. Recombining these two predicates using rule (Or) gives $\mathbb{P}(\mathit{door} = \mathit{prize}) = 1 \vee (\mathtt{i} \neq 2 \wedge p_3) \vee (\mathtt{i} \neq 3 \wedge p_2)$ when starting from from $p_{1,2} \vee p_{1,3}$. Doing the same for the other two parts of $q$ and using rules (Lin $\cdot$) and (Lin +) to recombine the results gives that $\frac{1}{3} \cdot (\mathbb{P}(\mathit{door} = \mathit{prize}) = 1 \vee (\mathtt{i} \neq 2 \wedge p_3) \vee (\mathtt{i} \neq 3 \wedge p_2)) + \frac{1}{3} \cdot (\mathbb{P}(\mathit{door} = \mathit{prize}) = 1 \vee (\mathtt{i} \neq 1 \wedge p_3) \vee (\mathtt{i} \neq 3 \wedge p_1)) + \frac{1}{3} \cdot (\mathbb{P}(\mathit{door} = \mathit{prize}) = 1 \vee (\mathtt{i} \neq 1 \wedge p_2) \vee (\mathtt{i} \neq 2 \wedge p_1))$ holds when starting from $q$. This predicate is similar to the postcondition deduced before, except for the added conditions $(\mathtt{i} \neq n)$ and the added possibility $\mathbb{P}(\mathit{door} = \mathit{prize}) = 1$. The extra conditions make it impossible to choose *door* different from *prize* in more than one of the three parts. In the other two parts the correct door is chosen giving a (worst case) probability of $\frac{2}{3}$. For example if $\mathtt{i} = 3$ then the first and second part must satisfy

$\mathbb{P}(door = prize) = 1$. The probability of winning the prize with the corrected "switch doors" strategy is $\frac{2}{3}$.

We have again solved the "three doors problem", and come to the same conclusion: Switching doors is the best strategy.

## 6.8 Conclusions and bibliographical remarks

The main result of this chapter is the introduction of a Hoare style logic, called *pH*, for reasoning about probabilistic programs. The programs are written in a language $\mathcal{L}_{\text{pif}}$ and their meaning is given by the denotational semantics $\mathcal{D}$. Extensions of $\mathcal{L}_{\text{pif}}$ with iteration, $\mathcal{L}_{\text{pw}}$, or with nondeterminism, $\mathcal{L}_{\text{pnif}}$, are also given together with denotational semantics and extensions of the Hoare style logic $pH_w$ and $pH_{nd}$. The Hoare-style logic for $\mathcal{L}_{\text{pw}}$ was first published in [108]. An adapted version of this work extended with a completeness result for $\mathcal{L}_{\text{pif}}$ can be found in [111]. The work on nondeterminism reported in section 6.7 has not been published previously.

In general, Hoare logic is a system to deduce (partial) correctness formulae, also known as Hoare triples (see [126, 34]). A Hoare triple links a precondition and a postcondition with a program, expressing that the given precondition is a sufficient precondition for the program to guarantee that the postcondition holds after termination. The Hoare logic provides a proof system to deduce the validity of such a triple. Extensive work exists in the area of Hoare-style logics. See e.g. [34, 12, 13] for an overview.

Earlier work on the proof theory for probabilistic programs that has inspired the present work, can be found in e.g. [141, 133]. In [141] Kozen proposes a probabilistic dynamic logic in which arithmetical laws govern the program analysis. The thesis work of Jones [133] considers a language similar to the language $\mathcal{L}_{\text{pw}}$ of section 6.6. A denotational semantics for the language is given using evaluations (see also [134]) which are a slight simplification of measures. (In the setting used in this thesis each evaluation has a unique extension to a measure, though not necessarily a compact support measure.) Logics for partial and total correctness are given where distribution functions over states are used rather than predicates. Both logics are shown to be consistent and complete with respect to the semantics. The work of Kozen and Jones concentrates on the semantics of programs. An important strand of research in the syntactic approach (cf. [162, 163]) is focused on predicate transformers. In [162], extending the predicate transformer work of [163], a calculus of greatest pre-expectations is given for a language with both probabilistic choice and nondeterminism.

The paper [174] by Pnueli and Zuck also deals with the verification of probabilistic processes but only looks at deterministic properties, expressed in a restricted temporal logic, that are satisfied with probability one. In [153] a deduction system is introduced which can be used to show a lower bound for the probability that a system satisfies a given condition.

To describe probabilistic properties, deterministic predicates can be extended to arithmetical functions yielding the probability that the predicate holds as done in e.g. [163, 141]. Other approaches like e.g. probabilistic modal logic and probabilistic dynamic logic [104, 182, 23] use a construct like e.g. $\diamond_\rho\, p$ to indicate either that the property $p$ holds with exactly probability $\rho$ or holds with at least probability $\rho$. The method of chapter 6 aims at combining both approaches; a deterministic predicate is seen as a function yielding the probability that the predicate holds. This function is introduced in general real valued expressions by using the notation $\mathbb{P}(dp)$ to refer to the chance that deterministic predicate $dp$ holds. The comparison of such expressions is the basis for the probabilistic predicates. Besides the usual logical operators the probabilistic predicates also contain arithmetical operators to be able to deal with probabilistic combination of predicates. Despite the presence of these arithmetical operators, the probabilistic predicates used in this thesis retain their usual truth value interpretation, i.e. they either hold or not. The main advantage of keeping the interpretation as truth values is that the logical operators do not have to be extended.

Each of the logics $pH$, $pH_w$ and $pH_{nd}$ is shown to be sound with respect to the corresponding semantics $\mathcal{D}$. Completeness of $pH$ (with some restriction) has been shown and $pH_{nd}$ is also conjectured to be complete. A completeness result (unpublished) also exists for a version of the logic $pH_w$ which uses infinite conjuctions. For the current logic the question of completeness is still open. Especially the expressiveness of the probabilistic predicates has to be studied further.

The proof system $pH$ assumes that all implications between probabilistic predicates are included as axioms, following the approach of [34, definition 2.39 and theorem 2.40]. The completeness result presented in section 6.5 relies on this assumption. In [62] standard Hoare logic is shown to be incomplete in a setting where a (finite) deduction system is used to derive the implications instead of including them as axioms. Still one would prefer to have a deduction system to check the implications. A deduction system for probabilistic predicates can use (the rules from) a standard deduction system (e.g. natural deduction for first order predicate logic) to deal with the logical aspects of the probabilistic predicates by seeing real conditions as logical primitives (i.e. primitive logical variables). The same deduction system can be used to deal with the deterministic predicates incorporated within the construct $\mathbb{P}(dp)$. To deal with the arithmetical aspects of the probabilistic predicates and equivalences of real conditions, however, a deduction system for probabilistic predicates needs to be extended. This extension of the deduction system needs to be able to deal with probabilities, and thus needs to support some calculation on real numbers.

To make the logic practically useful, the process of checking the deduction of a Hoare triple should be tool supported. Some work has been done to embed the logic in the proof verification system PVS. (See e.g. [128] on non-probabilistic Hoare logic in PVS.) The system PVS can then be used both to check the applications of the rules and to check the deduction of the implications between predicates required for the (Cons) rule. The development of a deduction system for the implications is also an important step in the development of tool support.

Two different extensions of the basic language $\mathcal{L}_{\text{pif}}$ have been considered in this chapter. In section 6.6 iteration is added to the language $\mathcal{L}_{\text{pif}}$ by introducing a `while` loop. In section 6.7 the operator $\square$ is introduced to add nondeterministic choice to the language $\mathcal{L}_{\text{pif}}$. Extending the language $\mathcal{L}_{\text{pif}}$ with both of these concepts at the same time creates several issues. One problem with giving a Hoare-style logic for this combination is that it is not clear how the notion of invariant, introduced for the language $\mathcal{L}_{\text{pw}}$ can be extended to deal with nondeterminism. Another problem is that with the presence of the `while` construct a fixed point construction is required to define its semantics. The usual method of defining a cpo structure on a nondeterministic domain consisting of sets of deterministic states (e.g. using the Egli-Milner ordering, see chapter 7 of [34]) does not extend to the domain consisting of sets of probabilistic states. This, however, seems to be only a technical complication that can be circumvented by using metric techniques to define the semantics.

To be able to describe distributed randomized algorithms, it would also be interesting to extend the language and the logic with parallelism. However, verification of concurrent systems in general and more specifically extending Hoare logic to concurrent systems (see e.g. [14, 84]) is already difficult in the non-probabilistic case.

Another way of checking if systems satisfy properties that are expressed in logical formulas is to use model checking. If the system can be represented as a finite state machine then a formula can be checked by exhaustive search of the state space. An advantage of this approach is that if the formula fails to hold, a counter example showing why this is the case is found. The state space explosion is one of the main problems; the size of the state space can easily become too large. To reduce the amount of work needed, several techniques are used such as reductions to smaller bisimilar models using process algebras. An extensive body of work exists in the field of model checking. The thesis work of Hartonas-Garmhausen [114] reports on applications of model checking for probabilistic systems. In [129], Huth and Kwiatkowska treat model checking of quantitative properties of a probabilistic interpretation of model $\mu$-calculus. Probabilistic model checking is applied in [146] to obtain properties of the probabilistic root contention IEEE1394 FireWire protocol. See [185, 184] for another approach of showing correctness of root contention in IEEE1394. A lot of research has also been done in the area of model checking continuous time Markov chains. A discussion of some of this work is given below.

In [23] Baier treats fully probabilistic systems as well as nondeterministic and probabilistic systems, which are referred to as concurrent probabilistic systems. The process language PCCS with reactive probabilistic choice, nondeterministic choice and interleaving parallel composition, the language PSCCS with generative probabilistic choice and synchronize product and the language PSLCCS with a less restrictive form of synchronize product are studied. Transition systems are given which use finite probability distributions to describe the probabilistic steps and an alternating approach when nondeterminism is also present. Methods for calculating bisimilarity of systems are also given in [23] and a notion of weak bisimilarity is defined. Denotational domains specified by domain equations, using either a metric or a complete partial order approach, are investigated and shown to capture bisimulation. Finally [23] considers a probabilistic temporal logic and model checking of properties expressed in this logic.

In [59] preorders on probabilistic processes are used by I. Christoff to express that one process is ‘probability better’ i.e. has smaller change of error than another process. In [61] an alternative characterization of the probabilistic testing preorders is also given and proof techniques are derived from the alternative characterizations.

# Chapter 7

# Continuous Probability

## 7.1 Introduction

This chapter provides an overview of the possibilities of compact support measures and other metric techniques in modeling non-discrete probabilities. In the preceding chapters several languages containing a discrete form of probabilistic choice have been introduced. In these languages the number of alternatives for a single choice is always finite, i.e. the probability is finitely branching. There are, however, probabilistic processes for which this is insufficient as the number of alternatives is infinite, possibly even uncountable. One such process is the number of arrivals in a given time period which may, for example, be modeled by a geometric distribution. A geometric distribution assigns a positive probability to infinitely many options. In subsection 6.6.1 of the previous chapter a program resulting in a geometric distribution for a program variable was given. However, this program requires unbounded execution time to reach this distribution. The programs that are considered in this chapter can assign a geometric distribution in a single step. Another process that may consist of the probabilistic choice between infinitely many options is the selection of a word in a language. Such a probabilistic choice between words can be used to describe the execution of a probabilistic automaton. Other important examples of processes with infinitely many options are those based on continuous distributions. A distribution on the positive real numbers, such as an exponential distribution, can be used to model, for instance, the life time of a light bulb. Such 'infinitely branching' probabilistic processes cannot be modeled with the finite probabilistic choice used so far.

In this chapter the language $\mathcal{L}_c$ with a notion of random assignment is introduced. The value assigned in a random assignment is probabilistically selected from a possibly infinite set of possible values. The applicability to the modeling of infinitely branching probabilistic processes of compact support measures and other metric techniques such as definitions and proofs using fixed points and arguments based on distances is studied in the context of the language $\mathcal{L}_c$. The other constructs present in the language $\mathcal{L}_c$ are the familiar notions of skip, deterministic assignment, conditional choice and iteration.

A problem with modeling a language with infinite probabilistic choices is that the choices made at different points in the execution of a program have to be combined to obtain probabilities for possible behaviors of the whole program. If only the distribution for each step is known, this is, in general, not possible. The notion of stochastic kernels is used in [169, 140] to enable the composition of infinite probabilistic choices. A metric version of these kernels, called $\mathcal{O}_*$-kernels, is introduced in section 7.2 based on the compact support measures used throughout this thesis.

As with compact support measures, $\mathcal{O}_*$-kernels are based on complete ultrametric spaces. The space of all kernels is again a complete ultrametric space, allowing the use of the usual metric techniques. We will see that, using the correct interpretation, $\mathcal{O}_*$-kernels correspond to nonexpansive functions. This correspondence and several other properties of $\mathcal{O}_*$-kernels will also be discussed in section 7.2.

In section 7.3 the language $\mathcal{L}_c$ with random assignment mentioned above is introduced. In subsection 7.3.2 a transition system $\mathcal{T}_c$ is given for this language. To be able to deal with the infinite choices, this transition systems uses an extension of the notion of transition system introduced in chapter 3 in that it allows infinite probabilistic steps described by measures.

After deriving some properties of the transition system $\mathcal{T}_c$ in subsection 7.3.3, the operational model $\mathcal{O}$ is defined in subsection 7.3.4 based on $\mathcal{T}_c$. The usual approach of justifying the definition of the operational model $\mathcal{O}$ by giving a contractive higher-order operator that has $\mathcal{O}$ as its fixed point, is again used.

In section 7.4 the use of $\mathcal{O}_*$-kernels and the modeling of the language $\mathcal{L}_c$ is discussed further and some related work is mentioned.

## 7.2 Mathematical preliminaries

With a finite choice (or a countably infinite choice), one knows each possible state with its probability. Two finite choices can be combined by using multiplication and addition. For example the probability that $x$ equals 2 after running the program $(x \texttt{ := } 0 \oplus_{\frac{1}{3}} x \texttt{ := } 1); (x \texttt{ := } x + 2 \oplus_{\frac{1}{2}} x \texttt{ := } x + 1)$ is $\frac{1}{3} \cdot \frac{1}{2} + \frac{2}{3} \cdot \frac{1}{2}$. As mentioned in the introduction of this chapter, the composition of infinite probabilistic choices requires additional structure. For an (uncountably) infinite choice, the probabilities for each separate state are not sufficient to find the probabilities of all events (see example 3.3.12). The probability of events has to be given directly. A measure is used to describe this type of infinite probabilistic choice. To combine infinite choices, integration is used instead of addition. Integration (over the measure) is only possible for measurable functions. The use of *stochastic kernels* guarantees that the appropriate functions are indeed measurable.

The notion of kernels used in this chapter is adapted from the notion of stochastic kernels, given in [169], to fit into the framework of ultrametric spaces and compact support measures. See section 7.4 for a discussion of further work in the area of modeling continues probabilistic choice. First some auxiliary notions are defined after which the definition of an adapted version of a kernel, an $\mathcal{O}_*$-kernel, is given. Finally some properties of $\mathcal{O}_*$-kernels are shown.

For the remainder of this section let $X$, $Y$ denote complete ultrametric spaces.

**Definition 7.2.1** *The set $\mathcal{O}_\epsilon(X)$ consists of the open subsets $O$ of $X$ which satisfy the property $\forall x \in O, x' \in X : d(x, x') < \epsilon \Rightarrow x' \in O$. A set in $\mathcal{O}_\epsilon(X)$ is called an $\epsilon$-open set.*

*The set $\mathcal{O}_*(X)$ is the set of all $\epsilon$-open sets for $\epsilon > 0$, i.e.*

$$\mathcal{O}_*(X) \ = \ \cup_{\epsilon>0}\, \mathcal{O}_\epsilon(X)$$

*A finite $\mathcal{O}_\epsilon(X)$ step function is a function $f$ from $X$ to $\mathbb{R}$ of the form*

$$f \ = \ \sum_{i \in I} \rho_i\, \chi_{B_i}$$

*with $I$ a finite index set and $B_i \in \mathcal{O}_\epsilon(X)$ for all $i \in I$. Recall that $\chi_B$ is given by*

$$\chi_B(x) \ = \ \begin{cases} 1 & \textit{if } x \in B \\ 0 & \textit{otherwise} \end{cases}$$

*for all $x \in X$.*

*An $\mathcal{O}_\epsilon(X)$ approximation sequence for a function $f : X \to Z$ is a sequence of finite $\mathcal{O}_\epsilon(X)$ step functions converging pointwise to $f$, thus*

$$f = \lim_{n\to\infty} \sum_{i \in I_n} \rho_i\, \chi_{B_i}$$

*with $B_i \in \mathcal{O}_\epsilon(X)$ for all $i \in I_n$, $n \in \mathbb{N}$. Without loss of generality the index sets $I_n$ are assumed to be pairwise disjoint.*

*A function $f$ is called $\epsilon$-measurable if there exists an $\mathcal{O}_\epsilon(X)$ approximation sequence for $f$.*

The sets $\mathcal{O}_\epsilon(X)$ and $\mathcal{O}_*(X)$ have also been used in lemma 3.3.16. In this lemma we saw that for $\mu, \mu' \in \mathit{Meas}(X)$

$$d(\mu, \mu') \leq \epsilon \iff \forall B \in \mathcal{O}_\epsilon(X) : \mu(B) = \mu'(B) \tag{2.1}$$

This property will be used below (cf. 7.2.3). The notion of $\epsilon$-measurable function is related to the notion of a *measurable* function. A measurable function $f$ is the limit of a sequence of step functions based on Borel sets. An $\epsilon$-measurable function satisfies the stronger condition that the step functions are based on $\mathcal{O}_\epsilon(X)$ sets. In particular, an $\epsilon$-measurable function is measurable.

**Definition 7.2.2** *A function $\kappa : X \times \mathcal{B}(Y) \to [0,1]$ is called an $\mathcal{O}_*$-kernel (on $X$ and $Y$) when*

1. *$\kappa(x)(\bullet)$ is a measure on $Y$ for all $x \in X$.*
2. *The function $\kappa(\bullet)(B)$ with $B \in \mathcal{O}_\epsilon(Y)$ is $\epsilon$-measurable.*

*The space of all $\mathcal{O}_*$-kernels on $X$ and $Y$ is denoted by KER$(X, Y)$. The distance between two $\mathcal{O}_*$-kernels $\kappa, \kappa' \in$ KER$(X, Y)$ is given by*

$$d(\kappa, \kappa') \ = \ \sup\{\, d_{\mathit{Meas}}(\kappa(x)(\bullet), \kappa'(x)(\bullet)) \mid x \in X\,\}$$

*with $d_{\mathit{Meas}}$ the distance on measures as given in definition 3.3.14.*

For standard stochastic kernels requirement 1 is also used but requirement 2 is replaced by $\kappa(\bullet)(B)$ is measurable for all $B \in \mathcal{B}(Y)$. Compared to this condition the requirement for $\kappa(\bullet)(B)$ with $B \in \mathcal{O}_*(Y)$ is stronger but no restriction is made on $\kappa(\bullet)(B)$ for $B \notin \mathcal{O}_*(Y)$. As the first condition requires that $\kappa(x)(\bullet)$ is a measure for each $x$ in $X$ one can easily show that $\kappa(\bullet)(B)$ is measurable for any open or closed subset $B$ of $Y$. For general Borel sets $B$, however, it is not clear whether this must be the case. This means that an $\mathcal{O}_*$-kernel is not necessarily a standard stochastic kernel.

To find the distance of two $\mathcal{O}_*$-kernels the kernels are interpreted as functions from $X$ to $Meas(Y)$. The distance on $\mathcal{O}_*$-kernels also satisfies

$$d(\kappa, \kappa') \ = \ \inf\{\, \epsilon > 0 \mid \forall y \in Y : \kappa(\bullet)(B_\epsilon(y)) = \kappa'(\bullet)(B_\epsilon(y)) \,\}$$

where $=$ on the right hand side denotes equality of functions. This means that interpreting the kernels as measures on functions, i.e. elements of $Meas(X \to Y)$, gives the same distance (see definition 3.3.14).

The interpretation of $\mathcal{O}_*$-kernels as functions in $X \to Meas(Y)$ satisfies an important property: The $\mathcal{O}_*$-kernels correspond exactly to the nonexpansive functions.

**Lemma 7.2.3** *A function $f : X \to Meas(Y)$ is nonexpansive exactly when $\kappa_f : X \times \mathcal{B}(Y) \to [0,1]$, given by $\kappa_f(x)(B) = (f(x))(B)$, is an $\mathcal{O}_*$-kernel.*

**Proof** Let $f : X \to Meas(Y)$ be a nonexpansive function. That $\kappa_f(x)(\bullet)$ is a measure is clear from the definition of $\kappa_f$. Showing that the function $\kappa(\bullet)(B)$ has an $\mathcal{O}_\epsilon$-approximation sequence uses a similar approach as the proof of the fact that a measurable function has an approximation of step functions based on Borel sets. Let $\epsilon > 0$ and $B \in \mathcal{O}_\epsilon(Y)$ and define (for $n \in \mathbb{N}, i = 0, \ldots, 2^n$)

$$E_{n,i} = \{\, x \mid \kappa_f(x)(B) \in [i\tfrac{1}{2}^n, (i+1)\tfrac{1}{2}^n) \,\}$$

then $\kappa_f(\bullet)(B) = \lim_{n\to\infty} \sum_{i=0}^{2^n} (\frac{1}{2}^n i)\chi_{E_{n,i}}$. It remains to be shown that $E_{n,i}$ is an $\epsilon$-open set. For $x \in E_{n,i}$ and $x' \in X$ with $d(x,x') < \epsilon$ we have $d(f(x), f(x')) \leq d(x,x') \leq \epsilon$. Using property (2.1) of distance on measures gives $(f(x))(B) = (f'(x))(B)$. Therefore, $\kappa_f(x')(B) = \kappa_f(x)(B) \in [i\frac{1}{2}^n, (i+1)\frac{1}{2}^n)$ and thus $x' \in E_{n,i}$.

For the reverse implication we assume that $\kappa_f : X \times \mathcal{B}(Y) \to [0,1]$ is an $\mathcal{O}_*$-kernel and show that for all $\epsilon > 0$ and $x, x' \in X$ with $d(x,x') < \epsilon$ we have $d(f(x), f(x')) \leq \epsilon$. This implies that $f$ is nonexpansive. By property 2.1 $d(f(x), f(x')) \leq \epsilon$ holds exactly when $f(x)(B) = f(x')(B)$ for all $B \in \mathcal{O}_\epsilon(Y)$. For $\epsilon > O$, $x, x' \in X$ with $d(x,x') < \epsilon$ and $B \in \mathcal{O}_\epsilon(Y)$ there exists an $\mathcal{O}_\epsilon(X)$ approximation of $\kappa_f(\bullet)(B)$ because $\kappa_f$ is an $\mathcal{O}_*$-kernel

$$\kappa_f(\bullet)(B) = \lim_{n\to\infty} \sum_{i \in I_n} \rho_i \, \chi_{B_i}$$

with $B_i \in \mathcal{O}_\epsilon(X)$. As $d(x,x') < \epsilon$ and $B_i \in \mathcal{O}_\epsilon(X)$ hold we have $x \in B_i \iff x' \in B_i$.

$$\begin{aligned} f(x)(B) &= \kappa_f(x)(B) \\ &= \lim_{n\to\infty} \sum_{i \in I_n} \rho_i \chi_{B_i}(x) \end{aligned}$$

$$\begin{aligned}
&= \lim_{n\to\infty} \sum_{\{\, i\in I_n, x\in B_i \,\}} \rho_i \\
&= \lim_{n\to\infty} \sum_{\{\, i\in I_n, x'\in B_i \,\}} \rho_i \\
&= f(x')(B) \qquad \Box
\end{aligned}$$

This lemma gives a characterization of $\mathcal{O}_*$-kernels in terms of nonexpansiveness. It also provides a way of showing that a function $\kappa : X \times \mathcal{B}(Y) \to [0,1]$ is an $\mathcal{O}_*$-kernel. First one checks that $\kappa$ defines a function $f : X \to \mathit{Meas}(Y)$ by showing that $\kappa(x)(\bullet)$ is a measure for each $x$. Next one shows that $\kappa$, interpreted as a function from $X$ to $\mathit{Meas}(Y)$, is nonexpansive. This approach is used e.g. in lemma 7.2.5 below.

**Lemma 7.2.4** *If $X$ and $Y$ are complete ultrametric spaces then KER($X, Y$) is also a complete ultrametric space.*

**Proof** As the space $X \xrightarrow{1} \mathit{Meas}(Y)$ of nonexpansive functions is a complete ultrametric space, this a direct consequence of lemma 7.2.3. (Note that, for nonexpansive functions $f, f' : X \xrightarrow{1} \mathit{Meas}(Y)$ we have $d(f, f') = d(\kappa_f, \kappa_{f'})$.) $\Box$

As the space of all $\mathcal{O}_*$-kernels, *KER($X, Y$)*, is a complete ultrametric space for complete ultrametric spaces $X$ and $Y$, the usual metric techniques can be used.

The *composition* of two $\mathcal{O}_*$-kernels is again an $\mathcal{O}_*$-kernel. This property is the reason for introducing $\mathcal{O}_*$-kernels; kernels allow the composition of infinitely branching probabilistic processes.

**Lemma 7.2.5** *Let $\kappa \in$ KER($X, Y$) and $\kappa' \in$ KER($Y, Z$). There is a unique $\mathcal{O}_*$-kernel $(\kappa \circ \kappa') \in$ KER($X, Z$) which satisfies*

$$(\kappa \circ \kappa')(x)(B) \;=\; \int_Y \kappa'(\bullet)(B)\, d\kappa(x)(\bullet)$$

*for all $x \in X$ and $B \in \mathcal{O}_*$. This $\mathcal{O}_*$-kernel is called the composition of $\kappa$ and $\kappa'$.*

**Proof** Using the fact that a measure is fixed by its behavior on the sets in $\mathcal{O}_*$ it is clear there is exactly one function $f_{\kappa,\kappa'}$ in $X \times \mathcal{B}(Z) \to [0,1]$ which satisfies

$$f_{\kappa,\kappa'}(x)(B) \;=\; \int_Y \kappa'(\bullet)(B)\, d\kappa(x)(\bullet)$$

for all $x \in X$ and $B \in \mathcal{O}_*$ and for which $f_{\kappa,\kappa'}(x)(\bullet)$ is a measure for all $x \in X$. It is, therefore, sufficient to show that this function $f_{\kappa,\kappa'}$ is an $\mathcal{O}_*$-kernel. By lemma 7.2.3 this coincides with the property that $f_{\kappa,\kappa'}$, interpreted as a function $f_{\kappa,\kappa'} : X \to \mathit{Meas}(Z)$, is nonexpansive.

Assume $x, x' \in X$ with $d(x, x') < \epsilon$ and $B \in \mathcal{O}_\epsilon(Z)$. As $\kappa'$ is an $\mathcal{O}_*$-kernel there exists $\kappa'(\bullet)(B)$ is $\epsilon$-measurable i.e. there exists an $\mathcal{O}_\epsilon(X)$ approximation sequence of $\kappa'(\bullet)(B)$

$$\kappa'(\bullet)(B) \;=\; \lim_{n\to\infty} \sum_{i\in I_n} \rho_i\, \chi_{B_i}$$

Using the definition of the integral we get

$$\begin{aligned} f_{\kappa,\kappa'}(x)(B) &= \int_Y \kappa'(\bullet)(B)\, d\kappa(x)(\bullet) \\ &= \lim_{n\to\infty} \sum_{i\in I_n} \rho_i\, \kappa(x)(B_i) \end{aligned}$$

As $d(\kappa(x),\kappa(x')) \leq d(x,x') < \epsilon$ and $B_i \in \mathcal{O}_\epsilon(Y)$ we have that $\kappa(x)(B_i) = \kappa(x')(B_i)$. Using this gives

$$\begin{aligned} f_{\kappa,\kappa'}(x)(B) &= \lim_{n\to\infty} \sum_{i\in I_n} \rho_i\, \kappa(x)(B_i) \\ &= \lim_{n\to\infty} \sum_{i\in I_n} \rho_i\, \kappa(x')(B_i) \\ &= f_{\kappa,\kappa'}(x')(B) \end{aligned}$$

So $f_{\kappa,\kappa'}(x)(\bullet)$ and $f_{\kappa,\kappa'}(x')(\bullet)$ coincide on all sets in $\mathcal{O}_\epsilon(Y)$. Their distance, therefore, is less than or equal to $\epsilon$. $\square$

One more property of measures is needed in the next section: The continuous image of a compact support measure is again a compact support measure.

**Lemma 7.2.6** *For $f : X \to Y$ a continuous function and $\mu \in Meas(X)$ a compact support measure, the image $\mu \circ f$ of $\mu$ under $f$, given by*

$$(\mu \circ f)(B) \;=\; \mu(f^{-1}[B])$$

*for all $B$ in $\mathcal{B}(Y)$, is a compact support measure in $Meas(Y)$.*

**Proof** As $f^{-1}[\cup_{i\in I} B_i] = \cup_{i\in I} f^{-1}[B]$ it is clear that $\mu \circ f$ is again a measure. The set $f[\mathrm{spt}(\mu)]$ is a compact set as it is the continuous image of a compact set. By using lemma 3.3.16 property (c) we show that this compact set is the support of $\mu \circ f$. For $B \in \mathcal{O}_*$ we have

$$\begin{aligned} (\mu \circ f)(B) = 0 \quad &\iff \quad \mu(f^{-1}[B]) = 0 \\ &\iff \quad f^{-1}[B] \cap \mathrm{spt}(\mu) = \emptyset \\ &\iff \quad B \cap f(\mathrm{spt}(\mu)) = \emptyset \end{aligned}$$

$\square$

## 7.3 The language $\mathcal{L}_c$ with random assignment

In this section the language $\mathcal{L}_c$ is introduced. The transition system $\mathcal{T}_c$ and the operational model $\mathcal{O}$ for this language use $\mathcal{O}_*$-kernels to allow the composition of multiple infinite probabilistic choices. The infinite probabilistic choices occur due to the presence of *random assignment* in the language $\mathcal{L}_c$. In a random assignment $x := \mu$ a value is assigned to the program variable $x$ according to some measure $\mu$. The support of the measure $\mu$, i.e. the set of values that can occur, may be infinite. Besides random assignment the language $\mathcal{L}_c$

contains skip, deterministic assignment, sequential composition, conditional choice and iteration. The syntactic details of expressions used in the deterministic assignment and of the conditions used in the conditional choice and iteration are omitted. Instead a set of expressions and a set of boolean conditions are assumed to be given as well as functions that evaluate these expressions and conditions. Each expression evaluates to some element of the set of values and a condition evaluates to true or false.

**Definition 7.3.1** *Let the set of values, denoted Val, be a given ultrametric space. Let PVar denote a set of program variables. The set PVar is ranged over by* x *and* y*.*

*A (deterministic) state $\sigma$ assigns a value to each program variable. The set of all states is denoted by $\mathcal{S}$.*

$$\mathcal{S} \;=\; \mathit{PVar} \to \mathit{Val}$$

*Note that $\mathcal{S}$ is an ultrametric space. The distance on $\mathcal{S}$ is obtained from the distance on Val. For $\sigma, \sigma' \in \mathcal{S}$ we have*

$$d(\sigma, \sigma') \;=\; \inf\{\, d(\sigma(\mathrm{x}), \sigma'(\mathrm{x})) \mid \mathrm{x} \in \mathit{PVar}\,\}$$

*The set Exp is a set of expressions ranged over by* e*. The evaluation function $\mathcal{V}$ gives, for each expression, a nonexpansive function that assigns a value to an expression in each state.*

$$\mathcal{V} \;:\; \mathit{Exp} \to (\mathcal{S} \xrightarrow{1} \mathit{Val})$$

*The set BC is a set of boolean conditions ranged over by c. The evaluation function $\mathcal{B}$ yields a truth value for each boolean condition and state*

$$\mathcal{B} \;:\; \mathit{BC} \to (\mathcal{S} \to \{\, \mathit{true}, \mathit{false}\,\})$$

*The notation $\sigma \models c$ is used for $\mathcal{B}(c)(\sigma) = \mathit{true}$ and $\sigma \not\models c$ is used for $\mathcal{B}(c)(\sigma) = \mathit{false}$.*

*The granularity* $\mathrm{gran}(c)$ *of a boolean condition $c$ is the largest radius (at most 1) such that $d(\sigma, \sigma') < \mathrm{gran}(c)$ implies that $\sigma \models c$ exactly when $\sigma' \models c$. Each boolean condition is assumed to have a granularity larger than zero.*

The values form an ultrametric space. Note that the set of real numbers with the usual Euclidean distance is not an ultrametric space and thus cannot be used as the set of values. This shows that the requirement of ultrametricity for the values is a significant restriction on the language $\mathcal{L}_c$. This restriction is required to be able to describe the random assignment using compact support measures. Restrictions on (evaluation of) expressions and boolean conditions are also required to remain within the realm of $\mathcal{O}_*$-kernels when giving the operational model for the language $\mathcal{L}_c$ below. The restriction that each boolean condition must have a granularity larger than zero means that the set $B = \{\, \sigma \mid \mathcal{B}(c)(\sigma) = \mathit{true}\,\}$ is an $\epsilon$-open set, $B \in \mathcal{O}_\epsilon(\mathcal{S})$, for some $\epsilon > 0$. In words one can express this requirement as follows: If a state $\sigma$ satisfies a condition $c$ then all states which are sufficiently close to $\sigma$ (less than $\mathrm{gran}(c)$ away) also satisfy the condition. To decide a condition $c$, one does not need to know the exact value of the variables. An approximation of the values is sufficient. The granularity of the condition gives how precise the approximation needs to be.

**Example 7.3.2** *In this example the granularity for some conditions that one could use in BC is given. Basic conditions that one would expect in BC are* `true`*,* `false` *and the combination of conditions using the usual logical operators.*

$$\begin{aligned}\mathrm{gran}(\mathtt{true}) &= 1\\ \mathrm{gran}(\mathtt{false}) &= 1\\ \mathrm{gran}(c\ \mathtt{op}\ c') &= \min\{\,\mathrm{gran}(c), \mathrm{gran}(c')\,\}\\ \mathrm{gran}(\neg c) &= \mathrm{gran}(c)\end{aligned}$$

*with* $\mathtt{op} \in \{\,\wedge, \vee, \rightarrow\,\}$.

*Another type of condition one might use is* $\mathrm{x} \in O$ *which restricts the value of the program variable* x *to values in the set* $O$*. For an* $\epsilon$*-open set* $O$*, this condition can be used.*

$$\mathrm{gran}(\mathrm{x} \in O) \ = \ \epsilon$$

*where* $\epsilon$ *is the greatest radius (at most* 1*) for which* $O$ *is an* $\epsilon$*-open set.*

*When* $Val = Act^{\infty}$ *one can use a condition of the form* $\mathrm{x} \,\mathtt{=}\, w$*. For this condition we have*

$$\mathrm{gran}(\mathrm{x} \,\mathtt{=}\, w) \ = \ \tfrac{1}{2}^{\text{length of } w}$$

*which means that this type of condition is only allowed for finite words* $w$*.*

We now have all ingredients needed to give the formal definition of the language $\mathcal{L}_c$.

**Definition 7.3.3** *The programs of the language* $\mathcal{L}_c$*, ranged over by* $s$*, are given by*

$$\begin{aligned} s \quad ::= \quad & \mathtt{skip} \mid \mathrm{x} \mathrel{\mathtt{:=}} e \mid \mathrm{x} \mathrel{\mathtt{:=}} \mu \mid s \,\mathtt{;}\, s \mid \mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s\ \mathtt{fi} \mid \\ & \mathtt{while}\ c\ \mathtt{do}\ s\ \mathtt{od}\end{aligned}$$

*with* $e \in Exp$*,* $c \in BC$*,* $\mu \in Meas(Val)$

The program `skip` does nothing and the program $x \mathrel{\mathtt{:=}} e$ assigns the value of the expression $e$ from $Exp$ to the program variable x. As the value of the expression $e$ is fixed given the state, the assignment $x \mathrel{\mathtt{:=}} e$ is referred to as deterministic assignment. As mentioned above, the evaluation of an expression $e$ must be nonexpansive in the state: For states which are close, the expressions yield values which are close.

In the random assignment statement $x \mathrel{\mathtt{:=}} \mu$, a measure on values is assigned to the program variable x instead of a fixed value. To execute $x \mathrel{\mathtt{:=}} \mu$ a value is randomly selected according to the measure $\mu$ and this value is assigned to x. For finite measures the random assignment can also be described by using discrete probabilistic choice. For example when the measure $\mu$ is given by $\mu = \frac{1}{2}\Delta_w + \frac{1}{2}\Delta_{w'}$, the random assignment $\mathrm{x} \mathrel{\mathtt{:=}} \mu$ has the same effect as the program $x \mathrel{\mathtt{:=}} w \oplus_{\frac{1}{2}} \mathrm{x} \mathrel{\mathtt{:=}} w'$ from the language $\mathcal{L}_{\mathrm{pif}}$ introduced in chapter 6.

The program $s \,\mathtt{;}\, s'$ executes the program $s$ followed by the program $s'$. As both $s$ and $s'$ may contain infinite probabilistic choices, one needs to be able to compose such choices to find the operational meaning of programs. The program $\mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s'\ \mathtt{fi}$ executes $s$ if the condition $c$ holds in the current state and $s'$ otherwise. As for the expressions, there are restrictions on the conditions that can be used, as mentioned in definition 7.3.1 above: It must be possible to decide whether $c$ holds by looking at a sufficiently close approximation of the values of the variables instead of at the actual values. The program `while` $c$ `do` $s$ `od` repeatedly executes $s$ until the condition $c$ no longer holds. The same restriction on the condition $c$ is made as for the conditional choice.

For a single condition the granularity gives how precisely the values of the variables need to be known to decide whether the condition is true or not. The precision needed to be able to decide all conditions that occur in a program is given by the *granularity of the program.*

**Definition 7.3.4** *The granularity of a program s is the minimum of the granularities of the conditions used in s:*

$$\begin{aligned}
\mathrm{gran}(\texttt{skip}) &= \mathrm{gran}(\mathtt{x}\ \texttt{:=}\ e) = \mathrm{gran}(\mathtt{x}\ \texttt{:=}\ \mu) = 1 \\
\mathrm{gran}(s\ ;\ s') &= \min\{\,\mathrm{gran}(s), \mathrm{gran}(s')\,\} \\
\mathrm{gran}(\texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{fi}) &= \min\{\,\mathrm{gran}(c), \mathrm{gran}(s), \mathrm{gran}(s')\,\} \\
\mathrm{gran}(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od}) &= \min\{\,\mathrm{gran}(c), \mathrm{gran}(s)\,\}
\end{aligned}$$

### 7.3.1 Examples

The natural numbers, together with infinity, can be modeled using a unary description of the natural numbers: i.e. by taking $\mathit{Val} = \{\,1\,\}^\infty$ and using the word $1^n$ to denote $n$. The measure $\mu_g$ given by $\mu_g(w\,\mathit{Val}) = \frac{1}{2}^{\text{length of } w}$ describes the geometrical distribution.

In the following example the unary description of the natural numbers is used as well as the measure $\mu_g$. A patron in a bar plays a game with the barkeeper. The patron has a sack with $2^N - 1$ coins ($N \geq 1$) and puts up one coin against a beer put in by the barkeeper. A fair coin is tossed. The patron wins if heads is shown, the barkeeper wins with tails. The patron keeps playing double or nothing until heads comes up, promising twice as many coins in each extra round needed. The variable $\mathtt{x}$ describes the number of 'double or nothing' rounds needed for the player to win a beer. The number of times the coin is thrown is thus $\mathtt{x} + 1$. This part of the game is described by the following program

$$\mathtt{x}\ \texttt{:=}\ \mu_g$$

After the patron wins, the barkeeper checks that the player actually had that many coins to bet. If there are enough coins in the sack, the patron gets and drinks the beer and the game is repeated, otherwise the patron is thrown out of the bar. The variable $\mathtt{y}$ counts the number of beers drunk by the patron. After ten drinks the patron gets drunk and gets thrown out.

The whole game can be described as follows.

$$\mathtt{x}\ \texttt{:=}\ \mu_g\ ;\ \texttt{while}\ ((\mathtt{y} < 10) \wedge (\mathtt{x} < N))\ \texttt{do}\ \mathtt{y} = \mathtt{y} + 1\ ;\ \mathtt{x}\ \texttt{:=}\ \mu_g\ \texttt{od}$$

In this program the 'double or nothing' game is played $\mathtt{x}\ \texttt{:=}\ \mu_g$ after the tenth drink but no consequences are drawn from the game. For this program one can for instance look at the probability that the patron goes bankrupt before getting drunk, i.e. the probability that $y$ is less than 10 at the end of the program or at the expected number of drinks the patron gets. Note that both these values depend on the number of drinks the patron has had before starting the game, i.e. the initial value of the program variable $y$. (See examples 7.3.8 and examples 7.3.16 below.)

An interval of positive real numbers, $[0, \mathrm{MaxReal}]$ can be modeled using the set of values $\mathit{Val} = \{0,1\}^\infty$. In this modeling, a word $w$ models the number $\mathrm{MaxReal} \cdot \sum_{n\in\mathbb{N}} \frac{1}{2}^n \cdot w_i$ where $w_i$ is the $i$'th number in $w$ (0 if no such number exists). In this binary modeling of the real interval $[0, \mathrm{MaxReal}]$ a simulation of any distribution on $[0, \mathrm{MaxReal}]$ can be given. The uniform distribution on $[0, \mathrm{MaxReal}]$ for example is given by

$$\mu_u(w\mathit{Val}) \;=\; \frac{1}{2}^{\text{length of } w}$$

In general one can put, for a given continuous distribution *dist*,

$$\mu_{\mathit{dist}}(w\mathit{Val}) \;=\; \mathit{Dist}(r(w1^\omega)) - \mathit{Dist}(r(w0^\omega))$$

with $r(w)$ the number represented by the word $w$ and $\mathit{Dist}(x)$ denoting the chance of an outcome less than or equal to $x$. If the distribution also contains a discrete part, e.g. $\mathit{dist}(x) > 0$ for some $x \in [0, \mathrm{MaxReal}]$ then this can also be modeled by putting $\mu_{\mathit{dist}}(w_x) = \mathit{dist}(x)$ for some selected $w$ that represents $x$ (there may be more than one such $w$).

The following example uses the modeling of real numbers by sequences in $\{0,1\}^\infty$ and the translation of some distribution on $\mathbb{R}$ to this setting. A lightbulb factory produces lightbulbs which have an exponentially distributed lifetime. The expected lifetime is 10 time units, i.e. the rate of the exponential distribution is $\frac{1}{10}$. A fraction $\rho$ of the lightbulbs, however, are bad. These lightbulbs have an expected lifetime of only $\frac{1}{5}$ of a time unit. Before packing the bulbs in a box of 6 bulbs, the bulbs are tested for 1 time unit. If they last for this one time unit, then they are found to be good, else they are marked as bad and discarded. The measure $\mu_{\lambda,r}$ has an exponential distribution with rate $\lambda$, cut off at some point $r$ ($\leq$ MaxReal). (All probability for values larger than $r$ is assigned to $r$.) As possible values the interval $[0, 2]$ is considered. The following program describes the filling of one box of bulbs.

```
while ($good < 6$) do
  $x$ := $\mu_{5,1} \oplus_\rho x$ := $\mu_{\frac{1}{10},1}$ ;
  if ($x < 1$) then $bad$ := $bad + 1$ else $good$ := $good + 1$ fi
od
```

This program uses two extensions of the language $\mathcal{L}_c$. In the first place, both real numbers and integers are used as data types, giving a multi typed system. The other extension is the use of the operator $\oplus_\rho$. Both these extensions are minor and fit within the setting developed in this chapter.

For the program above one can, for instance, ask what the expected number of lightbulbs is that need to be produced to fill a box. After finding the transitions for this program in examples 7.3.8 the operational semantics of the program is used to answer this question in examples 7.3.16.

Note that the geometric distribution was also obtained in the previous chapter by using a while loop, see subsection 6.6.1. This while loop, however, is not sure to terminate. Although an additional part of the probabilistic end state is generated in each iteration of the while loop, infinitely many iterations are needed to generate the complete geometric distribution. A program in $\mathcal{L}_c$ can generate the geometric distribution in a single step.

For other distributions, such as the uniform distribution on the reals mentioned above a generating program in $\mathcal{L}_{\text{pw}}$ of the previous chapter will never terminate and not even a part of the probabilistic end state is generated. As such no no further computations are possible with this distribution. As the 'bulb factory' example above shows, programs in $\mathcal{L}_{\text{pw}}$ can be used to do further computations for any distribution.

## 7.3.2 The transition system $\mathcal{T}_c$

The operational semantics for $\mathcal{L}_c$ is based on a transition system $\mathcal{T}_c$. As usual resumptions in the set *Res* are used to describe the past of a program that remains to be executed. A resumption $r$ is either a program or the empty resumption E describing a finished computation.

$$r \ ::= s \mid \mathrm{E}$$

A configuration $t = \langle r, \sigma \rangle$ of the transition system consists of the part of the program that remains to be executed and the value of the program variables. The set of all configurations is denoted by *Conf*.

$$\mathit{Conf} \ = \ \mathit{Res} \times \mathcal{S}$$

The set of configurations *Conf* is turned into an ultrametric space by defining the following distance $d$ on *Conf*

$$\begin{array}{rcl} d(\langle s, \sigma \rangle, \langle s', \sigma' \rangle) & = & 1 \quad \text{if } s \neq s' \\ d(\langle s, \sigma \rangle, \langle s, \sigma' \rangle) & = & \left\{ \begin{array}{ll} 1 & \text{if } d(\sigma, \sigma') \geq \mathrm{gran}(s) \\ d(\sigma, \sigma') & \text{otherwise} \end{array} \right. \end{array}$$

The distance between configurations is 1 if the programs are different. If the programs are the same, the distance is given by the distance between the states, provided that the states are close enough i.e. the distance of the states is less than the granularity of the program. The usage of the granularity of the program in this definition is required to obtain non-expansiveness for the transition function $\rightarrow$ below. Note that when $d(\langle s, \sigma \rangle, \langle s', \sigma' \rangle) < 1$ then the programs $s$ and $s'$ are the same and $\sigma$ will satisfy a boolean condition present in the program $s$ exactly when $\sigma'$ satisfies this condition.

For ease of notation, the notion of granularity is extended to configurations. The granularity of a configuration is the granularity of the statement component of the configuration.

**Definition 7.3.5** *The granularity of a configuration $t$ is defined as the granularity of the statement component of the configuration.*

$$\mathrm{gran}(\langle s, \sigma \rangle) \ = \ \mathrm{gran}(s)$$

The observation that is made with each step in the transition system consists of the values of the variables, i.e. the set of transition labels consists of the states.

$$Lab \;=\; \mathcal{S}$$

A transition $t \xrightarrow{\sigma} t'$ in the transition system is called a *deterministic step.* The infinite probabilistic choice caused by the random assignment cannot be described by transitions labeled with the appropriate probability as was done with the finite probabilistic choice in the transitions systems in previous chapters. Instead a new type of transition, called a *probabilistic step*, is introduced. In a probabilistic step $t \Rightarrow \mu$ the result of taking a step is not another configuration $t'$ but a measure $\mu$ over configurations. The measure $\mu$ gives the probability $\mu(B)$ of ending up within a given set of configurations $B$ after taking the step. To be able to compose probabilistic steps, the collection of all probabilistic steps, $\Rightarrow$, must form an $\mathcal{O}_*$-kernel, $\Rightarrow \in \mathit{KER}(\mathit{Conf}_{\mathrm{prob}}, \mathit{Conf})$. Here $\mathit{Conf}_{\mathrm{prob}}$ is the set of configurations which can take a probabilistic step.

**Definition 7.3.6** *The set of probabilistic configurations, $\mathit{Conf}_{prob} \subseteq \mathit{Conf}$ is the smallest set satisfying*

$$\begin{array}{rll}
\langle \mathtt{x} \;\mathtt{:=}\; \mu, \sigma\rangle \in \mathit{Conf}_{prob} & & \\
\langle s \;\mathtt{;}\; s', \sigma\rangle \in \mathit{Conf}_{prob} & \textit{if} & \langle s, \sigma\rangle \in \mathit{Conf}_{prob} \\
\langle \mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s'\ \mathtt{fi}, \sigma\rangle \in \mathit{Conf}_{prob} & \textit{if} & \sigma \models c \wedge \langle s, \sigma\rangle \in \mathit{Conf}_{prob}\ \textit{or} \\
 & & \sigma \not\models c \wedge \langle s', \sigma\rangle \in \mathit{Conf}_{prob} \\
\langle \mathtt{while}\ c\ \mathtt{do}\ s\ \mathtt{od}, \sigma\rangle \in \mathit{Conf}_{prob} & \textit{if} & \sigma \models c \wedge \langle s, \sigma\rangle \in \mathit{Conf}_{prob}
\end{array}$$

*The set of deterministic configurations $\mathit{Conf}_{det} \subseteq \mathit{Conf}$ is given by*

$$\mathit{Conf}_{det} \;=\; \{\, \langle s, \sigma\rangle \mid \langle s, \sigma\rangle \notin \mathit{Conf}_{prob} \,\}$$

Clearly $\mathtt{x} \;\mathtt{:=}\; \mu$ will produce a probabilistic step. The first step of $s \;\mathtt{;}\; s'$ is obtained from the first step of $s$. The first steps of $\mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s'\ \mathtt{fi}$ and $\mathtt{while}\ c\ \mathtt{do}\ s\ \mathtt{od}$ are the same as the first step of $s$ if the condition $c$ holds in the current state. If the condition $c$ does not hold then the first step of $\mathtt{if}\ c\ \mathtt{then}\ s\ \mathtt{else}\ s'\ \mathtt{fi}$ is that of $s'$.

A zero step from configuration $t$ to configuration $t'$, denoted $t \longrightarrow_0 t'$, indicates that $t$ inherits all transitions of the configuration $t'$. This includes both the deterministic and the probabilistic transitions of $t'$.

The notation $t \longrightarrow_0 t'$ is used as shorthand for the rules

$$\frac{t' \xrightarrow{\sigma} t''}{t \xrightarrow{\sigma} t''} \qquad \text{and} \qquad \frac{t' \Rightarrow \mu}{t \Rightarrow \mu}$$

Because there are two types of transitions, $t \longrightarrow_0 t'$ denotes a pair of rules instead of a single rule as in e.g. chapter 5.

We are now ready to give the definition of the transition system $\mathcal{T}_c$. It is easy to see from the specification of $\mathcal{T}_c$ that indeed $\mathit{Conf}_{\mathrm{prob}}$ consists of exactly those configurations that produce probabilistic steps and the deterministic configurations are exactly the configurations which produce a deterministic step.

**Definition 7.3.7** *The transition system $\mathcal{T}_c$ is given by $\mathcal{T}_c$ = (Conf, Lab, $\rightarrow$, $\Rightarrow$, Spec) with deterministic transitions $\rightarrow\ \subseteq$ Conf $\times$ Lab $\times$ Conf and probabilistic steps $\Rightarrow\ \in$ KER(Conf$_{\mathrm{prob}}$, Conf) derived from the specification Spec which contains*

- $$\langle \texttt{skip}, \sigma\rangle \xrightarrow{\sigma} \langle \mathrm{E}, \sigma\rangle \qquad \textit{(Skip)}$$
- $$\langle \mathrm{x} \texttt{ := } e, \sigma\rangle \xrightarrow{\sigma[\mathcal{V}(e)(\sigma)/\mathrm{x}]} \langle \mathrm{E}, \sigma[\mathcal{V}(e)(\sigma)/\mathrm{x}]\rangle \qquad \textit{(Assign)}$$
- $$\langle \mathrm{x} \texttt{ := } \mu, \sigma\rangle \Rightarrow [\mathrm{E}, \sigma[\mu/\mathrm{x}]] \qquad \textit{(Prob)}$$

  *where* $[\mathrm{E}, \sigma[\mu/\mathrm{x}]](B) = \mu(\{\, w \mid \langle E, \sigma[w/\mathrm{x}]\rangle \in B \,\})$.
- $$\frac{t \xrightarrow{\sigma} t'}{t; s \xrightarrow{\sigma} t'; s} \qquad \textit{(Seq 1)}$$

  *where* $\langle r, \sigma\rangle; s = \langle r; s, \sigma\rangle$ *with* $\mathrm{E}; s = s$.
- $$\frac{t \Rightarrow \mu}{t; s \Rightarrow \mu; s} \qquad \textit{(Seq 2)}$$

  *where* $\mu; s(B) = \mu(\{\, t \mid t; s \in B \,\})$.
- $$\begin{array}{ll} \langle \texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi}, \sigma\rangle \rightarrow_0 \langle s, \sigma\rangle & \textit{if } \sigma \models c \\ \langle \texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi}, \sigma\rangle \rightarrow_0 \langle s', \sigma\rangle & \textit{if } \sigma \not\models c \end{array} \qquad \textit{(If)}$$
- $$\begin{array}{ll} \langle \texttt{while } c \texttt{ do } s \texttt{ od}, \sigma\rangle \rightarrow_0 \langle s \texttt{ ; while } c \texttt{ do } s \texttt{ od}, \sigma\rangle & \textit{if } \sigma \models c \\ \langle \texttt{while } c \texttt{ do } s \texttt{ od}, \sigma\rangle \rightarrow_0 \langle \texttt{skip}, \sigma\rangle & \textit{if } \sigma \not\models c \end{array} \qquad \textit{(While)}$$

The program `skip` terminates without changing the state. The program $x$ `:=` $e$ changes the value of $x$ in a deterministic step. The program $s$ `;` $s'$ behaves like $s$ until $s$ terminates ($t' = \langle \mathrm{E}, \sigma_0\rangle$ for some state $\sigma_0$), after which it behaves like $s'$. Both deterministic and probabilistic steps of $s$ are mimicked by $s$ `;` $s'$. That $\mu$ `;` $s$ is indeed a measure for any statement $s$ is shown as part of lemma 7.3.11 below. The program `if` $c$ `then` $s$ `else` $s'$ `fi` behaves like $s$ or like $s'$ depending on the value of the condition $c$. The program `while` $c$ `do` $s$ `od` executes the body $s$ and repeats the loop when the condition $c$ holds and skips the body when the condition $c$ does not hold.

**Examples 7.3.8** *In this example the transitions for the programs introduced in subsection 7.3.1 are found. Put*

$$\begin{array}{rcl} s & = & x \texttt{ := } \mu_g \texttt{ ; } s_1 \\ s_1 & = & \texttt{while } ((y < \texttt{10}) \wedge (x < N)) \texttt{ do } y = y + \texttt{1 ; } x \texttt{ := } \mu_g \texttt{ od} \end{array}$$

*Axiom (Prob) gives*

$$\langle \mathrm{x} \texttt{ := } \mu_g, \sigma\rangle \Rightarrow [\mathrm{E}, \sigma[\mu_g/\mathrm{x}]]$$

*so by rule (Seq) also*

$$\langle \mathrm{s}, \sigma\rangle \Rightarrow [\mathrm{s}_1, \sigma[\mu_g/\mathrm{x}]]$$

*The next step depends on the value of* $y$ *and on the value assigned to the variable* x*. If the value of* $y$ *is less than* 10 *and the value* $M$ *assigned to* x *is less than* $N$ *then we reason as follows. By axiom (Assign)*

$$\langle y \texttt{ := } y + 1, \sigma[M/\mathrm{x}]\rangle \xrightarrow{\sigma[M/\mathrm{x}][y+1/y]} \langle \mathrm{E}, \sigma[M/\mathrm{x}][y+1/y]\rangle$$

*so by rule (Seq)*

$$\langle y \texttt{ := } y + 1 \texttt{ ; } \mathrm{x} \texttt{ := } \mu_g \texttt{ ; } s_1, \sigma[M/\mathrm{x}]\rangle \xrightarrow{\sigma[M/\mathrm{x}][y+1/y]} \langle \mathrm{x} \texttt{ := } \mu_g \texttt{ ; } s_1, \sigma[M/\mathrm{x}][y+1/y]\rangle$$

*and thus by rule (While) also*

$$\langle s_1, \sigma[M/\mathrm{x}]\rangle \xrightarrow{\sigma[M/\mathrm{x}][y+1/y]} \langle \mathrm{x} \texttt{ := } \mu_g \texttt{ ; } s_1, \sigma[M/\mathrm{x}][y+1/y]\rangle$$

*When* $M$ *is greater than or equal to* $N$ *or the value of* $y$ *is greater than or equal to* 10 *we obtain the conclusion*

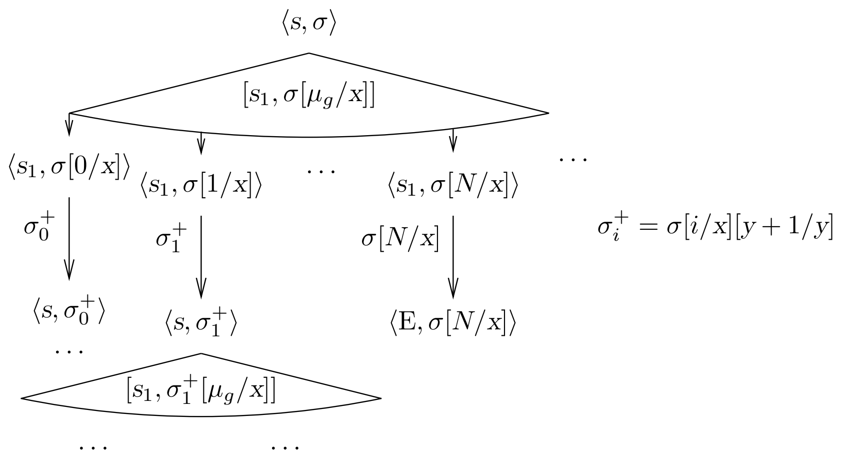
$$\langle s_1, \sigma[M/\mathrm{x}]\rangle \xrightarrow{\sigma[M/\mathrm{x}]} \langle \mathrm{E}, \sigma[M/\mathrm{x}]\rangle$$

*by rule (While) from the premise*

$$\langle \texttt{skip}, \sigma[M/\mathrm{x}]\rangle \xrightarrow{\sigma[M/\mathrm{x}]} \langle \mathrm{E}, \sigma[M/\mathrm{x}]\rangle$$

*which in turn is obtained from an application of axiom (Skip).*

*The following picture shows part of the transition tree obtained for the configuration* $\langle s, \sigma\rangle$ *with* $\sigma(y) < 10$*.*


$\langle s, \sigma\rangle$
$[s_1, \sigma[\mu_g/\mathrm{x}]]$
$\langle s_1, \sigma[0/\mathrm{x}]\rangle$
$\langle s_1, \sigma[1/\mathrm{x}]\rangle$
$\langle s_1, \sigma[N/\mathrm{x}]\rangle$
$\sigma_0^+$
$\sigma_1^+$
$\sigma[N/\mathrm{x}]$
$\sigma_i^+ = \sigma[i/\mathrm{x}][y+1/y]$
$\langle s, \sigma_0^+\rangle$
$\langle s, \sigma_1^+\rangle$
$\langle \mathrm{E}, \sigma[N/\mathrm{x}]\rangle$
$[s_1, \sigma_1^+[\mu_g/\mathrm{x}]]$

*The following program* $s'$ *was also introduced in subsection 7.3.1.*

$$\begin{aligned}
s' &= \texttt{while } (\mathit{good} < 6) \texttt{ do } s_1' \texttt{ od}\\
s_1' &= \mathrm{x} \texttt{ := } \mu_{5,1} \oplus_\rho \mathrm{x} \texttt{ := } \mu_{\frac{1}{10},1} \texttt{ ; } s_2'\\
s_2' &= \texttt{if } (\mathrm{x} < 1) \texttt{ then } \mathit{bad} = \mathit{bad} + 1 \texttt{ else } \mathit{good} = \mathit{good} + 1 \texttt{ fi}
\end{aligned}$$

*Although a general probabilistic choice can also be added easily, this special case can also be written as* x := $\mu$ *with* $\mu = \rho\mu_{5,1} + (1-\rho)\mu_{\frac{1}{10},1}$ *so the extension is not needed. The transitions for the program* $s'$ *are as follows. When giving a deterministic transition in the remainder of this example only the part of the state that changes is shown on top of the arrow instead of the whole state.*

$$\langle s'_1\,, \langle good = 0, bad = 0, \mathrm{x} = 0\rangle\rangle \;\Rightarrow\; [s'_2\,, \langle good = 0, bad = 0, \mathrm{x} = \mu\rangle]$$

*with* $\mu$ *the measure on values given by* $\mu = \rho \cdot \mu_{5,1} + (1-\rho)\cdot\mu_{\frac{1}{10},1}$*. As* $0 < 6$ *holds, rule (While) gives*

$$\langle s', \langle good = 0, bad = 0, \mathrm{x} = r\rangle\rangle \;\Rightarrow\; [s'_2\;;\, s', \langle good = 0, bad = 0, \mathrm{x} = \mu\rangle]$$

*For a value* $r < 1$ *we obtain*

$$\langle s'_2\;;\, s', \langle good = 0, bad = 0, \mathrm{x} = r\rangle\rangle \xrightarrow{bad=1} \langle s', \langle good = 0, bad = 1, \mathrm{x} = r\rangle\rangle$$

*from axiom (Assign) and rules (If) and (Seq). For other values for* $r$ *(due to the context necessarily* $r = 1$*) we obtain*

$$\langle s'_2\;;\, s', \langle good = 0, bad = 0, \mathrm{x} = r\rangle\rangle \xrightarrow{good=1} \langle s', \langle good = 1, bad = 0, \mathrm{x} = r\rangle\rangle$$

*also from axiom (Assign) and rules (If) and (Seq).*

### 7.3.3 Properties of the transition system

In this subsection correctness of the definition of transition system $\mathcal{T}_c$ is shown as well as several properties of $\mathcal{T}_c$ that are needed for the definition of the operational model $\mathcal{O}$ in the next subsection.

Several of the proofs in this section use induction on the weight $wgt_c$ of configurations. The weight of a configuration $\langle r, \sigma\rangle$ is given by the weight $wgt$ of the resumption $r$. The state does not play a role.

**Definition 7.3.9** *The functions* $wgt_c : Conf \to \mathbb{N}$ *and* $wgt : Res \to \mathbb{N}$ *are given by*

$$\begin{array}{rcl}
wgt_c(\langle r, \sigma\rangle) & = & wgt(r) \\
wgt(\mathrm{E}) & = & 0 \\
wgt(\texttt{skip}) & = & 1 \\
wgt(\mathrm{x} \texttt{ := } e) & = & 1 \\
wgt(\mathrm{x} \texttt{ := } \mu) & = & 1 \\
wgt(s\ ;\ s') & = & wgt(s) + 1 \\
wgt(\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi}) & = & wgt(s) + wgt(s') + 1 \\
wgt(\texttt{while } c \texttt{ do } s \texttt{ od}) & = & wgt(s) + 2
\end{array}$$

Well-definedness of $wgt$ and $wgt_c$ is clear. The definition of weight of the program `while` $c$ `do` $s$ `od` is chosen to be $wgt(s) + 2$ because this gives that the weight of the left hand side in the rule (While) for the case $\sigma \models c$ is greater than that of the right hand side

$$\begin{array}{rclcl} wgt_c(\langle \texttt{while}\ c\ \texttt{do}\ s\ \texttt{od}, \sigma\rangle) & = & wgt(\texttt{while}\ c\ \texttt{do}\ s\ \texttt{od}) & = & wgt(s) + 2 \\ wgt_c(\langle s\, \texttt{;}\, \texttt{while}\ c\ \texttt{do}\ s\ \texttt{od}, \sigma\rangle) & = & wgt(s\, \texttt{;}\, \texttt{while}\ c\ \texttt{do}\ s\ \texttt{od}) & = & wgt(s) + 1 \end{array}$$

In general we have that if $t \rightarrow_0 t'$ then $wgt_c(t) > wgt_c(t')$.

**Lemma 7.3.10**

*(a) For each rule in Spec, the premise is 'simpler' than the conclusion of the rule. In this comparison the complexity of a transition is the complexity (weight) of the configuration at the left hand side of the transition.*

*(b) Each configuration $t$ in $Conf_{prob}$ has a single probabilistic step, i.e. there is exactly one measure $\mu_t \in Meas(Conf)$ such that $t \Rightarrow \mu_t$ holds.*

*(c) Each configuration $t$ in $Conf_{det}$ has a single deterministic step, i.e. there is exactly one state $\sigma$ and one configuration $t'$ such that $t \xrightarrow{\sigma} t'$ holds.*

The second and third property give that for each configuration of the form $\langle s, \sigma\rangle$ there is exactly one probabilistic or deterministic step. The first property implies that this step can always be found using finitely many applications of the rules in *Spec*.

Because each probabilistic configuration has a unique probabilistic step, the collection of probabilistic steps $\Rightarrow$ can be seen as a function from $Conf_{\mathrm{prob}}$ to $Meas(Conf)$. In line with this interpretation, the notation $\Rightarrow(t)(B)$ is used for $\mu(B)$ with $\mu$ the unique measure such that $t \Rightarrow \mu$. The definition of the transition system requires that $\Rightarrow$ is an $\mathcal{O}_*$-kernel. The following lemma shows that this is indeed the case.

**Lemma 7.3.11** *$\Rightarrow$ is a $\mathcal{O}_*$-kernel in $KER(Conf_{prob}, Conf)$.*

**Proof** By using lemma 7.2.3 it is sufficient to show that $\Rightarrow(t)(\bullet)$ is a measure for all $t \in Conf_{\mathrm{prob}}$ and that $\Rightarrow$, interpreted as a function from $Conf_{\mathrm{prob}}$ to $Meas(Conf)$, is nonexpansive.

Assume $t, t' \in Conf_{\mathrm{prob}}$ with $d(t, t') < \epsilon$, $\epsilon \in (0, 1]$. By induction on $wgt_c(t)$ we show that $\Rightarrow(t)(\bullet)$ and $\Rightarrow(t')(\bullet)$ are two measures with distance at most $\epsilon$. Only a few representative cases are given.

[$t = \langle \mathrm{x} := \mu, \sigma\rangle$] then $t' = \langle \mathrm{x} := \mu, \sigma'\rangle$ with $d(\sigma, \sigma') < \epsilon$. The probabilistic transition for $t$ is $t \Rightarrow [\mathrm{E}, \sigma[\mu/\mathrm{x}]]$ and the transition for $t'$ is $t' \Rightarrow [\mathrm{E}, \sigma'[\mu/\mathrm{x}]]$.

As the function which takes a state $\sigma$ to the configuration $\langle \mathrm{E}, \sigma[\mathcal{V}(e)(\sigma)/\mathrm{x}]\rangle$ is nonexpansive and thus continuous, $[\mathrm{E}, \sigma[\mu/\mathrm{x}]]$ is indeed a compact support measure by lemma 7.2.6. Similarly $[\mathrm{E}, \sigma'[\mu/\mathrm{x}]]$ is also a compact support measure.

Let $B$ be an $\epsilon$-open set $B \in \mathcal{O}_\epsilon(Conf)$ then, as $d(\langle \mathrm{E}, \sigma[w/\mathrm{x}]\rangle, \langle \mathrm{E}, \sigma'[w/\mathrm{x}]\rangle) \leq d(\langle \mathrm{E}, \sigma\rangle, \langle \mathrm{E}, \sigma'\rangle) < \epsilon$ holds for every value $w$

$$\begin{array}{rcl} [\mathrm{E}, \sigma[\mu/\mathrm{x}]](B) & = & \mu(\{\, w \mid \langle E, \sigma[w/\mathrm{x}]\rangle \in B \,\}) \\ & = & \mu(\{\, w \mid \langle E, \sigma'[w/\mathrm{x}]\rangle \in B \,\}) \ = \ [\mathrm{E}, \sigma'[\mu/\mathrm{x}]](B) \end{array}$$

which means $d([\mathrm{E}, \sigma[\mu/\mathrm{x}]], [\mathrm{E}, \sigma'[\mu/\mathrm{x}]]) \leq \epsilon$.

[$t = \langle s \texttt{ ; } s', \sigma\rangle$] then $t' = \langle s \texttt{ ; } s', \sigma'\rangle$ with $d(\sigma,\sigma') < \epsilon_0 = \min\{\,\epsilon, \mathrm{gran}(t)\,\}$. By induction $\langle s,\sigma\rangle \Rightarrow \mu$ for some measure $\mu$. The transition system gives $\langle s \texttt{ ; } s', \sigma\rangle \Rightarrow \mu; s'$. As the function that assigns $t_0; s'$ to the configuration $t_0$ is continuous, $\mu; s'$ is a compact support measure by lemma 7.2.6.

Also by induction $\langle s \texttt{ ; } s', \sigma'\rangle \Rightarrow \mu'; s'$ for some measure $\mu'$ with $d(\mu,\mu') < \epsilon_0$. Again $\mu'; s'$ is a measure. The distance between $\mu; s'$ and $\mu'; s'$ is at most $\epsilon$: For $B \in \mathcal{O}_\epsilon$ we have that $\{\, t \mid t; s \in B\,\}$ is an $\epsilon_0$-open set and thus

$$\mu; s'(B) = \mu(\{\, t \mid t; s \in B\,\}) = \mu'(\{\, t \mid t; s \in B\,\}) = \mu'; s'(B)$$

[$t = \langle \texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi}, \sigma\rangle$ with $\sigma \models c$, $\langle s,\sigma\rangle \in \mathit{Conf}_{\mathrm{prob}}$] then we have $t' = \langle \texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi}, \sigma'\rangle$ with $d(\sigma,\sigma') < \epsilon_0 = \min\{\,\epsilon, \mathrm{gran}(t)\,\}$. In particular $d(\sigma,\sigma') < \mathrm{gran}(c)$ so $\sigma' \models c$. By induction there exists a measure $\mu'$ with $\langle s,\sigma'\rangle \Rightarrow \mu'$ and $d(\mu,\mu') < \epsilon$ But then, by rule(If), also $t' \Rightarrow \mu'$. □

Just like a probabilistic configuration, a deterministic configuration has a single step. The transition relation $\rightarrow$ can therefore be seen as a function from $\mathit{Conf}_{\mathrm{det}}$ to $\mathcal{S} \times \mathit{Conf}$, yielding the observation and the next configuration for the unique transition of a deterministic configuration. The function $\rightarrow$ is nonexpansive. Corollary 7.3.13 states a property which directly follows from this and is important for the correctness of the definition of the operational model $\mathcal{O}$ in the next subsection.

**Lemma 7.3.12** *The function $\rightarrow$ is nonexpansive, i.e. for all $\epsilon \in (0,1]$ if $t_1 \xrightarrow{\sigma} t_1'$ and $d(t_1,t_2) < \epsilon$ then $t_2 \xrightarrow{\sigma'} t_2'$ with $d(\sigma,\sigma') \le \epsilon$ and $d(t_1',t_2') \le \epsilon$.*

**Proof** The proof is similar to the proof of lemma 7.3.11 above. Only a few cases are treated. Assume $t_1, t_2 \in \mathit{Conf}_{\mathrm{det}}$ with $d(t_1,t_2) < \epsilon$

[$t_1 = \langle \mathrm{x} \texttt{ := } e, \sigma_1\rangle$] then $t_2 = \langle \mathrm{x} \texttt{ := } e, \sigma_2\rangle$ with $d(\sigma_1,\sigma_2) < \epsilon$. The transition for $t_1$ is $t_1 \xrightarrow{\sigma_1'} \langle \mathrm{E}, \sigma_1'\rangle$ with $\sigma_1' = \sigma_1[\mathcal{V}(e)(\sigma_1)/\mathrm{x}]$ and the transition for $t_2$ is $t_2 \xrightarrow{\sigma_2'} \langle \mathrm{E}, \sigma_2'\rangle$ with $\sigma_2' = \sigma_2[\mathcal{V}(e)(\sigma_2)/\mathrm{x}]$. For the states $\sigma_1'$ and $\sigma_2'$ we have

$$\begin{array}{rcl} d(\sigma_1',\sigma_2') & \le & \max\{\, d(\sigma_1,\sigma_2), d(\mathcal{V}(e)(\sigma_1), \mathcal{V}(e)(\sigma_2))\,\} \\ & \le & [\mathcal{V}(e) \text{ nonexpansive}]\ d(\sigma_1,\sigma_2) \end{array}$$

This also gives $d(\langle \mathrm{E},\sigma_1'\rangle, \langle \mathrm{E},\sigma_2'\rangle) \le \epsilon$.

[$t_1 = \langle \texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi}, \sigma_1\rangle$] then $t_2 = \langle \texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi}, \sigma_2\rangle$ with $d(\sigma_1,\sigma_2) < \epsilon_0 = \min\{\,\epsilon, \mathrm{gran}(c)\,\}$. The transition for $t_1$ is that of $\langle s,\sigma_1\rangle$, i.e. if $\langle s,\sigma_1\rangle \xrightarrow{\sigma_1'} t_1'$ then $t_1 \xrightarrow{\sigma_1'} t_1'$. By induction also $\langle s,\sigma_2\rangle \xrightarrow{\sigma_2'} t_2'$ for some $t_2'$ with $d(t_1',t_2') \le \epsilon$ and $d(\sigma_1',\sigma_2') \le \epsilon$. Because $\sigma_2$ also satisfies $c$ we have that the transition of $t_2$ is $t_2 \xrightarrow{\sigma_2'} t_2'$. □

In the proof of this lemma the nonexpansiveness of the evaluation of expressions is used. In lemma 7.3.11 above we saw that the fact that if a condition $c$ of a conditional choice in a program is satisfied by the state $\sigma$ in a configuration $t$ then the state of any configuration $t'$ close to $t$ also satisfies this condition. The nonexpansiveness of the deterministic transitions guarantees that if $t$ and $t'$ are close they will still satisfy the same conditions after taking a deterministic step. For example if $t = \langle x \texttt{ := } e\,\texttt{;}\,\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi}, \sigma\rangle$ and $t' = \langle x \texttt{ := } e\,\texttt{;}\,\texttt{if } c \texttt{ then } s \texttt{ else } s' \texttt{ fi}, \sigma'\rangle$ are close then in the second step both will choose $s$ or both will choose $s'$.

Nonexpansiveness of $\rightarrow$ is also needed in a more direct way for the correctness of the definition of the operational model $\mathcal{O}$. The operational model $\mathcal{O}$ is defined as a kernel (which, by lemma 7.2.3 corresponds to a nonexpansive function to measures). To show that it is indeed a kernel the following property is used.

**Corollary 7.3.13** *If $f : Conf \rightarrow Meas(\mathcal{S}^\infty)$ is a nonexpansive function then for configurations $t_1, t_2 \in Conf_{det}$ with transitions $t_1 \xrightarrow{\sigma} t_1'$ and $t_2 \xrightarrow{\sigma'} t_2'$ respectively we have*

$$d(f(t_1')/\sigma, f(t_2')/\sigma') \leq d(t_1, t_2)$$

## 7.3.4 The operational domain and the operational semantics

The observable behavior produced by a deterministic transition in the transition system $\mathcal{T}_c$ is value of the program variables, i.e. the state, at that point of execution. The observable behavior produced by a single run of a program is a sequence of states. By using multiple runs the probabilities for 'observable events' can be found; the complete observable behavior produced by a program is described by a measure over sequences of states.

**Definition 7.3.14** *The operational domain $\mathbb{P}_o$ is given by $\mathbb{P}_o = Meas(\mathcal{S}^\infty)$*

Usually the operational model is a function from configurations to the operational domain $\mathbb{P}_o$. Here more structure is required to be able to define $\mathcal{O}$. The model $\mathcal{O}$ is defined as an $\mathcal{O}_*$-kernel on configurations and sequences of states.

**Definition 7.3.15** *The operational model $\mathcal{O}$ is the $\mathcal{O}_*$-kernel in $KER(Conf, \mathcal{S}^\infty)$ satisfying*

$$\begin{array}{rcl} \mathcal{O}(\langle \mathrm{E}, \sigma\rangle)(B) & = & \Delta_\epsilon(B) \\ \mathcal{O}(t)(B) & = & \left\{ \begin{array}{ll} \mathcal{O}(t')(B/\sigma) & \mathit{if}\; t \xrightarrow{\sigma} t' \\ \int \lambda t'.\mathcal{O}(t')(B)\, d\mu & \mathit{if}\; t \Rightarrow \mu \end{array} \right. \end{array}$$

*for all sets $B$ in $\mathcal{O}_*$.*

The empty configuration E represents a finished computation and thus produces an empty sequence of states with probability 1. For configurations which take a deterministic step, the operation $\bullet/\sigma$ is used (see definition 3.3.18). A probabilistic configuration $t$ has a transition $t \Rightarrow \mu$. The probability of ending up in an $\epsilon$-open set $B$ is found by integration. In this way the probabilities of ending up after taking the probabilistic transition $(\mathcal{O}(t')(B))$ are combined. The probabilities are only given for $\epsilon$-open sets (for any $\epsilon > 0$). For an $\epsilon$-open set $B$ the function $\lambda t'.\mathcal{O}(t')(B)$ is indeed a measurable function. For general Borel sets $B$ this is not known, so the integral may not be defined. The probabilities for other Borel sets are fixed by the properties of a measure, so there is no need to give them directly.

**Examples 7.3.16** *For the programs introduced in subsection 7.3.1 the transitions were found in examples 7.3.8. Using these transitions, the operational model $\mathcal{O}$ can be found and used to find the probabilities of given events.*

*For the 'bar game' program, one can for example look at the probability the patron goes broke before getting a single drink, i.e. at the probability of the set $B = \{ \sigma_1 \dots \sigma_n \mid \sigma_n(y) = 0, n \geq 1, \sigma_1, \dots, \sigma_n \in \mathcal{S} \}$. Naturally we assume that the patron has had no drinks so far, thus for the starting state $\sigma$ we have $\sigma(y) = 0$.*

*The first transition in the program s describing the bar game (see examples 7.3.8 above) is a probabilistic transition, $\langle s, \sigma \rangle \Rightarrow \mu$. The definition of the model $\mathcal{O}$ uses integration for probabilistic configurations.*

$$\begin{array}{rcl} \mathcal{O}(\langle s, \sigma \rangle)(B) & = & \int \mathcal{O}(\bullet)(B)\, d\mu \\ & = & \int \mathcal{O}(\langle s_1, \sigma[\bullet/x] \rangle)(B)\, d\mu_g \end{array}$$

*The next step is a deterministic transition taken by $s_1$ which depends on the value of x. If the value of x is greater or equal to $N$ then the patron gets thrown out and the program is done, otherwise a drink is served and another round is started.*

$$\begin{array}{l} \mathcal{O}(\langle s_1, \sigma[n/x] \rangle)(B) \\ \quad = \left\{ \begin{array}{ll} \mathcal{O}(\langle \mathrm{E}, \sigma[n/x] \rangle)(B/\sigma[n/x]) & \textit{if } n > N \\ \mathcal{O}(\langle s_1, \sigma[n/x][y/y+1] \rangle)(B/\sigma[n/x][y/y+1]) & \textit{otherwise} \end{array} \right. \\ \quad = \left\{ \begin{array}{ll} 1 & \textit{if } n \geq N \\ 0 & \textit{otherwise} \end{array} \right. \end{array}$$

*Substituting this in the equation for $(\langle s, \sigma \rangle)$ above gives*

$$\begin{array}{rcl} \mathcal{O}(\langle s, \sigma \rangle)(B) & = & \int_{\{ n \in \mathit{Val} \mid n \geq N \}} 1\, d\mu_g + \int_{\{ n \in \mathit{Val} \mid n < N \}} 0\, d\mu_g \\ & = & \mu_g(\{ n \in \mathit{Val} \mid n \geq N \}) \\ & = & \frac{1}{2}^N \end{array}$$

*The probability of the patron going broke without getting a single drink is $\frac{1}{2}^N$.*

*For the second program describing the lightbulb production, one can look at the expected number of bulbs that need to be produced to fill a box. To this end the probability of producing n bad bulbs is found for $n = 0, 1, \dots$. Below the calculation is given in case 5 bulbs have already passed the test. The notation $\langle n, m, r \rangle$ is used for the state which assigns n to good, m to bad and r to x. Also the notation $B_{n,m}$ is used for the set of outcomes ending with n good and m bad bulbs, i.e. $B_{n,m} = \{ w\langle n, m, r' \rangle \mid w \in \mathit{Stat}^*, r' \in [0, 2] \}$. First we note that for the probability of discarding m more bulbs the number of bulbs already discarded does not matter, thus for any $i \in \mathbb{N}$ and $r \in [0, 2]$ we have*

$$\mathcal{O}(\langle s', \langle 5, i, r \rangle \rangle)(B_{6,i+m}) \;=\; \mathcal{O}(\langle s', \langle 5, 0, r \rangle \rangle)(B_{6,m})$$

*This can be checked by induction on $m$ with a calculation similar to the first part of the calculation below.*

*The probability of producing no bad bulbs is found as follows with start value $r$ for $x$*

$$\begin{array}{ll} \mathcal{O}(\langle s', \langle 5,0,r\rangle\rangle)(B_{6,0}) & \\ \quad = & \int_{Conf} \mathcal{O}(\bullet)(B_{6,0})\, d[s'_1\,, \langle 5,0,\mu\rangle] \\ \quad = & \int_{[0,2]} \mathcal{O}(\langle s'_1\,, \langle 5,0,\bullet\rangle\rangle)(B_{6,0})\, d\mu \\ \quad = & \int_{[0,1)} \mathcal{O}(\langle bad \texttt{ := } bad + \texttt{1 ;}\ s', \langle 5,0,\bullet\rangle\rangle)(B_{6,0})\, d\mu \\ & + \int_{[1,2]} \mathcal{O}(\langle good \texttt{ := } good + \texttt{1 ;}\ s', \langle 5,0,\bullet\rangle\rangle)(B_{6,0})\, d\mu \\ \quad = & \int_{[0,1)} 0\, d\mu + \int_{[1,2]} 1\, d\mu \\ \quad = & \mu([1,2]) \end{array}$$

*with $\mu = \rho \cdot \mu_{5,1} + (1-\rho)\cdot \mu_{\frac{1}{10},1}$. Thus we get*

$$\begin{array}{rcl} \mathcal{O}(\langle s', \langle 5,0,r\rangle\rangle)(B_{6,0}) & = & \rho \cdot \mu_{5,1}([1,2]) + (1-\rho)\cdot \mu_{\frac{1}{10},1}([1,2]) \\ & = & \rho e^{-5} + (1-\rho) e^{-\frac{1}{10}} \end{array}$$

*Other probabilities can be found using similar calculations.*

The definition of the higher order function $\Phi$ uses integration over the set $Conf < t$, the set of all configurations with a smaller weight than $t$. Note that $Conf < t$ is a 1-open set. If $t \Rightarrow \mu$ then the measure $\mu$ assigns probability 0 to the set of configurations with weight equal to or higher than the weight of the configuration $t$. As a result $\int_{Conf<t} f\, d\mu = \int f\, d\mu$ for all measurable functions $f$.

**Lemma 7.3.17** *Put $Sem_O = KER(Conf, \mathcal{S}^\infty)$ and define the higher order mapping $\Phi : Sem_O \to Sem_O$ by*

$$\begin{array}{rcl} \Phi(S)(\langle \mathrm{E}, \sigma\rangle)(B) & = & \Delta_\epsilon(B) \\ \Phi(S)(t)(B) & = & \left\{ \begin{array}{ll} S(t')(B/\sigma) & t \xrightarrow{\sigma} t' \\ \int_{Conf<t} \Phi(S)(\bullet)(B)\, d\mu & t \Rightarrow \mu \end{array} \right. \end{array}$$

*for $B \in \mathcal{O}_*(\mathcal{S}^\infty)$ and with $Conf < t = \{\, t' \in Conf \mid wgt_c(t') < wgt_c(t)\,\}$ The function $\Phi$ has a unique fixed point and, therefore, there exists exactly one kernel $\mathcal{O}$ in $Sem_O$ which satisfies the equations in definition 7.3.15.*

**Proof** It is sufficient to show that $\Phi$ is well-defined and contractive.

For well-definedness we need to show that $\Phi(S)$ is an $\mathcal{O}_*$-kernel. The function $\Phi$ is split into parts, depending on the weight and the type of the configuration: Define the sets $Conf^n$, $Conf^n_{\mathrm{prob}}$ and $Conf^n_{\mathrm{det}}$ by

$$\begin{array}{rcl} Conf^n & = & \{\, t \in Conf \mid wgt_c(t) \leq n\,\} \\ Conf^n_{\mathrm{prob}} & = & \{\, t \in Conf_{\mathrm{prob}} \mid wgt_c(t) \leq n\,\} \\ Conf^n_{\mathrm{det}} & = & \{\, t \in Conf_{\mathrm{det}} \mid wgt_c(t) \leq n\,\} \end{array}$$

and the functions $\Phi^n(S) : \mathit{Conf}^n \times \mathcal{B}(\mathcal{S}^\infty) \to [0,1]$, $\Phi^n_{\mathit{prob}}(S) : \mathit{Conf}^n_{\mathrm{prob}} \times \mathcal{B}(\mathcal{S}^\infty) \to [0,1]$ and $\Phi^n_{\mathit{det}}(S) : \mathit{Conf}^n_{\mathrm{det}} \times \mathcal{B}(\mathcal{S}^\infty) \to [0,1]$ by

$$\begin{array}{rcl} \Phi^0(S)(t)(B) & = & \Delta_\epsilon(B) \\ \Phi^n(S)(t)(B) & = & \Phi^n_{\mathit{prob}} + \Phi^n_{\mathit{det}} \\ \Phi^n_{\mathit{prob}}(S)(t)(B) & = & \int \Phi^{n-1}(S)(\bullet)(B)\, d\mu \quad t \Rightarrow \mu \\ \Phi^n_{\mathit{det}}(S)(t)(B) & = & S(t')(B/\sigma) \quad t \xrightarrow{\sigma} t' \end{array}$$

To show that $\Phi(S)$ is an $\mathcal{O}_*$-kernel it is sufficient to show that $\Phi^n(S)$ is an $\mathcal{O}_*$-kernel for each $n \in \mathbb{N}$ because then (see lemma 7.2.3)

1. $\Phi(S)(t)(\bullet) = \Phi^n(S)(t)(\bullet)$ with $n = \mathit{wgt}_c(t)$, so $\Phi(S)(t)$ is a measure for each configuration $t$

2. $d(\Phi(S)(t)(\bullet), \Phi(S)(t')(\bullet)) = d(\Phi^n(S)(t)(\bullet), \Phi^n(S)(t')(\bullet)) \le d(t,t')$ with $n = \max\{\, \mathit{wgt}_c(t), \mathit{wgt}_c(t')\,\}$.

Assume $n > 0$, as $\Phi^0(S)$ is clearly an $\mathcal{O}_*$-kernel. If the distance between two configurations in $\mathit{Conf}^n$ is less than one, then both are in $\mathit{Conf}^n_{\mathrm{det}}$ or both are in $\mathit{Conf}^n_{\mathrm{prob}}$. From this fact it is easy to see that $\Phi^n(S)$ is an $\mathcal{O}_*$-kernel when both $\Phi^n_{\mathit{det}}(S)$ and $\Phi^n_{\mathit{prob}}(S)$ are $\mathcal{O}_*$-kernels.

As $S$ is an $\mathcal{O}_*$-kernel, $\Phi^n_{\mathit{det}}(S)(t)(\bullet) = S(t)/\sigma$ is a measure for each $t$ in $\mathit{Conf}^n_{\mathrm{det}}$ $(t \xrightarrow{\sigma} t')$. Corollary 7.3.13 gives that $\Phi^n_{\mathit{prob}}(S)$, seen as a function from configurations to measures, is nonexpansive. According to lemma 7.2.3 this means that $\Phi^n_{\mathit{prob}}(S)$ is an $\mathcal{O}_*$-kernel.

As $\Rightarrow(t)$ assigns probability 0 to the set of configurations with weight equal to or higher than the weight of the configuration $t$, the probabilistic steps $\Rightarrow$, which form an $\mathcal{O}_*$-kernel in *KER(Conf*$_{\mathit{prob}}$*, Conf)*, can also be seen as an $\mathcal{O}_*$-kernel in *KER(Conf*$^n_{\mathit{prob}}$*, Conf*$^{n-1}$*)* for each $n \in \mathbb{N}$. Lemma 7.2.5 gives that

$$\Phi^n_{\mathit{prob}} \;=\; \Rightarrow \circ\, \Phi^{n-1}$$

is an $\mathcal{O}_*$-kernel in *KER(Conf*$^n_{\mathit{prob}}$*,* $\mathcal{S}^\infty$*)* for each $n \in \mathbb{N}$.

It remains to be shown that $\Phi$ is contractive. For $S, S' \in \mathit{Sem}_O$ with $d(S,S') < \epsilon$ we show $d(\Phi(S)(t), \Phi(S')(t))$ is less than or equal to $\frac{1}{2}\epsilon$ by induction on the weight of the configuration $t$. Take $B$ any $\frac{1}{2}\epsilon$-open set and configuration $t \in \mathit{Conf}_{\mathrm{det}}$ with $t \xrightarrow{\sigma} t'$ then

$$\begin{array}{rcl} \Phi(S)(t)(B) & = & S(t)(B/\sigma) \\ & = & S'(t)(B/\sigma) \\ & = & \Phi(S')(t)(B) \end{array}$$

as $B/\sigma$ is $\epsilon$-open and $d(S(t), S'(t)) \le d(S,S') < \epsilon$. For a configuration $t \in \mathit{Conf}_{\mathrm{prob}}$ with $t \Rightarrow \mu$, and again $B$ any $\frac{1}{2}\epsilon$-open set

$$\begin{array}{rcl} \Phi(S)(t)(B) & = & \int_{\mathit{Conf}<t} \Phi(S)(\bullet)(B)\, d\mu \\ & = & [\text{ind. hyp.}] \int_{\mathit{Conf}<t} \Phi(S')(\bullet)(B)\, d\mu \\ & = & \Phi(S')(t)(B) \end{array} \qquad \Box$$

The operational model $\mathcal{O}$ is an $\mathcal{O}_*$-kernel. The operational semantics should be a function that, for a program and a start state, yields a process in $\mathbb{P}_o = \mathit{Meas}(\mathcal{S}^\infty)$. The definition of the operational semantics $\mathcal{O}[\![\bullet]\!]$ removes this small discrepancy.

**Definition 7.3.18** *The operational semantics* $\mathcal{O}[\![\bullet]\!] : \mathcal{L}_c \times \mathcal{S} \to \mathbb{P}_o$ *is given by*

$$\mathcal{O}[\![s, \sigma]\!] \;=\; \lambda B.\mathcal{O}(\langle s, \sigma\rangle)(B)$$

The operational semantics yields, given a program and a start state, the probability for sequences of states that may be observed during the execution of the program i.e. a measure over sequences of states. The operational semantics is obtained by forgetting the extra structure imposed on the model $\mathcal{O}$.

## 7.4 Conclusions and bibliographical remarks

In this chapter we have seen how $\mathcal{O}_*$-kernels, an extension of compact support measures, can be used to model infinite probabilistic choices. The results in this chapter have not been previously published. The form of infinite choice considered was a language $\mathcal{L}_c$ with random assignment. In the random assignment a choice is made out of a possibly infinite set of values according to some measure over these values. As such the modeling of $\mathcal{L}_c$ requires the composition of infinite probabilistic choices. In chapter 3 measures are also used. There, however, the measures are only used to describe the complete meaning of a program and there is no need to compose such measures. Likewise in chapter 6 a single step will only give finitely many options and only a finite form of composition is needed. In the language $\mathcal{L}_c$ a measure is obtained in a single step. To enable the composition of measures, needed to give the complete processes describing the meaning of a program in $\mathcal{L}_c$, the $\mathcal{O}_*$-kernels are used.

There are two major restrictions when using $\mathcal{O}_*$-kernels to model infinite probabilistic choice. The first restriction has to do with the use of ultrametric spaces. As with compact support measures, $\mathcal{O}_*$-kernels are based on complete ultrametric spaces. The space of all kernels is again a complete ultrametric space, allowing the use of the usual metric techniques. The use of (the properties of) nonexpansive functions plays an important role in metric semantics. The $\mathcal{O}_*$-kernels fit very nicely in the metric setting due to the correspondence between $\mathcal{O}_*$-kernels and nonexpansive functions (as formalized by lemma 7.2.3). The disadvantage of using compact support measures as the basis for $\mathcal{O}_*$-kernels is that the set of values has to be an ultrametric space. In the setting of continuous probability, the spaces involved are typically metric spaces but not ultrametric spaces, e.g. the real numbers $\mathbb{R}$ with the Euclidean distance. The real numbers are often used in combination with continuous distributions to describe time based systems (e.g. CTMCs, see section 1.3) or hybrid systems (e.g. [158, 157]). By using different modelings, these spaces can be simulated. For example the real numbers can be modeled using a binary string in $\{\,0, 1\,\}^\infty$ as was done in subsection 7.3.1. A problem with this modeling is that the requirements imposed on expressions and conditions in the language $\mathcal{L}_c$ disallow even some of the most elementary operations on real numbers such as addition of two numbers. A different simulation of the real numbers may not have this restrictions. Ideally the simulation is also computable [186], allowing the calculation of operations.

The other restriction is caused by the requirements on expressions and conditions in the language $\mathcal{L}_C$. To remain within the setting of $\mathcal{O}_*$-kernels, the deterministic assignment has to be restricted to nonexpansive assignment and boolean conditions have to have a positive granularity. The requirement that the boolean conditions have a positive granularity means that an approximation of the value of the variables with some fixed precision is sufficient to see if a condition is true. The requirement of nonexpansiveness on expressions guarantees that no precision is lost in a deterministic assignment. In many cases these are not unreasonable requirements. In the example of the real numbers modeled by $\{0,1\}^\infty$ (see subsection 7.3.1), however, these restrictions have, perhaps unexpected, consequences. For example the operation of addition cannot be used because it causes loss of precision: the numbers modeled by 0000 and 0001 are close as they start with the same sequence 000 while $0111 + 0000 = 0111$ and $0111 + 0001 = 1000$ already differ in the first number.

It may be possible to weaken the requirement on $\mathcal{O}_*$-kernels so that the requirement of nonexpansiveness of the deterministic assignment is no longer needed. However this is likely to require measure theoretic investigations falling out of the scope of the research conducted here. Also one would have to give up the nice correspondence of kernels and nonexpansive functions. The new requirements on the kernels should satisfy the two essential properties that make $\mathcal{O}_*$-kernels useful in the metric modeling of infinitely branching probabilistic processes: The space of all kernels should be a complete metric space and one should be able to compose kernels (cf. lemma 7.2.5).

The $\mathcal{O}_*$-kernels give a metric version of the stochastic kernels (see [169, 140, 141]) extending the compact support measures. As the distance of $\mathcal{O}_*$-kernels is based on the distance of compact support measures, this distance is 'qualitative on the probabilities' rather than quantitative as in e.g. [53, 52]. (See the discussion in section 3.6.) We know of no other approaches that also use a qualitative approach with kernels.

In [170, 169] a simple language is studied which can be extended with random assignment to obtain a language like $\mathcal{L}_C$. The semantics of this language is given using kernels and gives the input-output behavior of programs. As such it does not provide information on possible infinite behavior.

In [140] Kozen studies a language with random assignment similar to the language studied here. Any distribution may be used in the random assignment but the same distribution is used for every random assignment in the program. The random assignment is decided by popping the next number of a stack of random numbers that is assumed to be available. In this way all probability is resolved ahead of time by generation of the stack of random numbers.

Metric models of continuous probabilistic processes have also been given by Van Breugel and Worrell in [53, 52]. The general coalgebraic approach used there allows for an easy combination of the Hutchinson metric to describe distances between labeled actions and the functor $id_{\frac{1}{2}}$ to compose sequential steps in a contractive way. The pseudometric obtained is not an ultra-metric. Another main difference with this work is that Van Breugel and Worrell use a branching domain while we use a linear domain. The essential operation $\bullet/\sigma$ of 'prefixing on a measure' is not a contractive operation when using the Hutchinson metric which makes it less suited for operational models based on sequences like the function $\mathcal{O}$ given in this chapter. For branching domains, however, the approach of Van Breugel and Worrell results in a very intuitive distance.

An important cause of infinite choices in the modeling of processes is the explicit modeling of (real) time. The presence of time introduces a continuous component into the model. Real time is introduced into a process algebra in [18]. The reasoning in [18] requires checking infinitely many equations for some conclusions. This issue is addressed in [83]. In [20] a logic for reasoning about real time processes is introduced. These papers do not deal with probability.

A large body of research where continuous probability is used exists in the field of performance modeling. A basic formalism used in this setting is that of a continuous time Markov chain (CTMC). A CTMC is a labeled transitions system where the labels express the expected delay before taking this transition and with this also the probability of taking this transition. The delays are always exponentially distributed.

Model checking of properties of CTMCs, for example expressed in continuous stochastic logic (e.g. [27]), suffers from the well known state space explosion problem. To fight this problem efficient model checking techniques have been developed for CTMCs using variations on binary decision diagrams (BDDs) [138, 4, 143]. A different approach that reduces model checking to the analysis of transient state probabilities is given in [25]. Also compositional models have been developed. Initially queuing networks and (generalized) stochastic Petri nets were used to specify CTMCs. To defined CTMCs on a higher-level and in a compositional way different formalisms have also been introduced such as stochastic process algebras [69, 125, 121, 64, 118]. The process algebra EMPA [46] extends the basic setting of stochastic process algebra with nondeterminism and priority.

The results in the thesis of Hermanns [117] provide the foundation for the development of the TIPPtool. Starting from the stochastic process algebra TIPP (TImed Processes for Performance evaluation) introduced by Herzog [124] the TIPPtool has been designed to allow compositional specification and performance evaluation [119, 123, 120]. In her thesis [125] Hillston treats performance evaluation using stochastic extensions of Petri nets as well as probabilistic process algebras.

The restriction in CTMCs that only exponential distributions can be used is removed in semi-Markov chains where general continuous distributions are allowed. Model checking in this setting is studied in [3, 130, 144].

In [64] D'Argenio also presents a stochastic process algebra with time called spades which is used to reason about stochastic automata in a compositional way (see also [68]). The stochastic automata combine ideas from timed automata and generalized semi-Markov processes. The stochastic automata can contain nondeterminism. In quantitative analysis this is removed by use of probabilistic adversaries. Other work in the area of timed and stochastic automata can be found in [65, 66, 71].

The paper [78] introduces sequences of finite approximations of labeled Markov processes that enable the checking of properties expressed in a probabilistic dynamic logic (which characterizes bisimulation): Any property satisfied by the labeled Markov process is already satisfied on one of the finite approximations in the sequence.

# Bibliography

[1] L. Aceto. *Action Refinement in Process Algebras.* PhD thesis, University of Sussex, 1990.

[2] L. Aceto and M. Hennessy. Towards action-refinement in process algebras. *Information and Computation*, 103:204–269, 1993.

[3] L. de Alfaro. *Formal Verification of Probabilistic Systems.* PhD thesis, Stanford University, 1997.

[4] L. de Alfaro, M.Z. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of concurrent probabilistic processes using MTBDDs and the Kronecker representation. In S. Graf and M. Schwartzbach, editors, *Proc. of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* LNCS 1785, 2000.

[5] N. Alon, J.H. Spencer, and P. Erdös. *The Probabilistic Method.* Series in Discrete Mathematics and Optimization. Wiley, 1992.

[6] P. America and J.J.M.M. Rutten. Solving reflexive domain equations in a category of complete metric spaces. *Journal of Computer and System Sciences*, 39:343–375, 1989.

[7] S. Andova. Process algebra with interleaving probabilistic parallel composition. Technical Report CSR 99-04, Eindhoven University of Technology, 1999.

[8] S. Andova. Process algebra with probabilistic choice. In J.-P. Katoen, editor, *Proc. ARTS'99*, pages 111–129. LNCS 1601, 1999.

[9] S. Andova. Time and probability in process algebra. In T. Rus, editor, *Proc. AMAST 2000*, pages 323–338. LNCS 1816, 2000.

[10] S. Andova. *Probabilistic Process Algebra.* PhD thesis, Technical University Eindhoven, in preparation.

[11] S. Andova and J.C.M. Baeten. Abstraction in probabilistic process algebra. In *Proc. TACAS'01*, pages 204–219. LNCS 2031, 2001.

[12] K.R. Apt. Ten years of Hoare's logic: A survey-part I. *ACM Transactions on Programming Languages and Systems*, 3(4):431–483, 1981.

[13] K.R. Apt. Ten years of Hoare's logic: A survey-part II: nondeterminism. *Theoretical Computer Science*, 28:83–109, 1984.

[14] K.R. Apt and E.-R. Olderog. *Verification of Sequential and Concurrent Programs.* Texts and Monographs in Computer Science. Springer-Verlag, 1991.

[15] A. Arnold and M. Nivat. Metric interpretations of infinite trees and semantics of nondeterministic recursive programs. *Theoretical Computer Science*, 11(2):181–205, 1980.

[16] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation.* IOP Publishing Ltd and Oxford University Press, 1997.

[17] T. Baeck, J.M. de Graaf, J.N. Kok, and W.A. Kosters. Theory of genetic algorithms. *Bulletin of the EATCS*, 63:161–192, October 1997.

[18] J.C.M. Baeten and J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[19] J.C.M. Baeten and J.A. Bergstra. Process algebra with partial choice. In B. Jonsson and J. Parrow, editors, *Proc. CONCUR'94*, pages 465–480. LNCS 836, 1994.

[20] J.C.M. Baeten, J.A. Bergstra, and R.N. Bol. A real time process logic. In D.M. Gabbay and H.J. Ohlbach, editors, *Proc. ICTL'94*, pages 30–47. LNCS 827, 1994.

[21] J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 121:234–255, 1995.

[22] J.C.M. Baeten and W.P. Weijland. *Process algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science.* Cambridge University Press, 1990.

[23] C. Baier. *On the Algorithmic Verification of Probabilistic Systems.* Habilitation, Universität Mannheim, 1998.

[24] C. Baier, B. Engelen, and M. Majster-Cederbaum. Deciding bisimilarity and similarity for probabilistic processes. *Journal of Computer and System Sciences*, 60:187 – 231, 2000.

[25] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Proc. CAV 2000.* LNCS 1855, 2000.

[26] C. Baier and H. Hermanns. Weak bisimulation for fully probabilistic processes. In *Proc. CAV'97*, pages 119–130. LNCS 1254, 1997.

[27] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *Proc. CONCUR'99*, pages 146–161. LNCS 1664, 1999.

[28] C. Baier and M. Kwiatkowska. Domain equations for probabilistic processes (extended abstract). In M. Mislove, M. Nivat, C. Papadimitriou, C. Palamidessi, and J. Parrow, editors, *Proc. Express'97.* Electronic Notes in Theoretical Computer Science 7, 1997.

[29] C. Baier, M. Kwiatkowska, and G. Norman. Computing probability bounds for linear time formulas over concurrent Markov chains. In *Proc. Workshop on Probabilistic Methods in Verification (PROBMIV'98)*, volume 22 of *Electronic Notes in Theoretical Computer Science*, 1999.

[30] C. Baier and M.E. Majster-Cederbaum. The connection between an event structure semantics and an operational semantics for TSCP. *Acta Informatica*, 31:81–104, 1994.

[31] C. Baier and M.E. Majster-Cederbaum. Denotational semantics in the CPO and metric approach. *Theoretical Computer Science*, 135:171–220, 1994.

[32] C. Baier and M.E. Majster-Cederbaum. Metric semantics from partial order semantics. *Acta Informatica*, 34:701–735, 1997.

[33] C. Baier and M.I.A. Stoelinga. Norm functions for probabilistic bisimulations with delays. In J. Tiuryn, editor, *Proc. FOSSACS 2000*, pages 1–16. LNCS 1784, 2000.

[34] J.W. de Bakker. *Mathematical Theory of Program Correctness.* Series in Computer Science. Prentice-Hall International, 1980.

[35] J.W. de Bakker, J.-J.Ch. Meyer, E.-R. Olderog, and J.I. Zucker. Transition systems, metric spaces and ready sets in the semantics of uniform concurrency. *Journal of Computer and System Sciences*, 36:158–224, 1988.

[36] J.W. de Bakker and J.J.M.M. Rutten, editors. *Ten Years of Concurrency Semantics, selected papers of the Amsterdam Concurrency Group.* World Scientific, 1992.

[37] J.W. de Bakker and E.P. de Vink. Bisimulation semantics for concurrency with atomicity and action refinement. *Fundamenta Informaticae*, 20:3–34, 1994.

[38] J.W. de Bakker and E.P. de Vink. *Control Flow Semantics.* Foundations of Computing Series. The MIT Press, 1996.

[39] J.W. de Bakker and E.P. de Vink. Denotational models for programming languages: Applications of Banach's fixed point theorem. *Topology and its Applications*, 85:35–52, 1998.

[40] J.W. de Bakker and J.H.A. Warmerdam. Metric pomset semantics for a concurrent language with recursion. In I. Guessarian, editor, *Proc. of the LITP Spring School on Theoretical Computer Science*, pages 21–49. LNCS 469, 1990.

[41] J.W. de Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54:70–120, 1982.

[42] J.A. Bergstra and J.W. Klop. The algebra of recursively defined processes and the algebra of regular processes. In J. Paredaens, editor, *Proc. ICALP'84*, pages 82–95. LNCS 172, 1984.

[43] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.

[44] J.A. Bergstra, A.J. Ponse, and S.A. Smolka, editors. *Handbook of process algebra.* Elsevier, Amsterdam, 2001.

[45] M. Bernardo and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. Foundations of Software Technology and Theoretical Computer Science*, pages 499 – 513. LNCS 1026, 95.

[46] M. Bernardo, L. Donatiello, and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.

[47] B. Bollobás. *Graph theory.* Springer-Verlag, 1979.

[48] M.M. Bonsangue. *Topological Dualities in Semantics.* PhD thesis, Vrije Universiteit, November 1996.

[49] M.M. Bonsangue and J.N. Kok. Specifying computations using hyper transition systems. In *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science (MFCS '97)*, pages 169–178, August 1997.

[50] F. C. van Breugel. Failures, finiteness, and full abstraction. Technical Report TR-97-18, Dipartimento di Informatica, Università degli Studi di Pisa, 1997.

[51] F. C. van Breugel. *Comparative Metric Semantics for Programming Languages.* Progress in Theoretical Computer Science. Birkhäuser Verlag, Boston, 1998.

[52] F. C. van Breugel and J. Worrell. An algorithm for quantitative verification of probabilistic transition systems. In K.G. Larsen and M. Nielsen, editors, *Proc. 12th CONCUR'01, Aalborg*, pages 336 – 350. LNCS 2154, 2001.

[53] F. C. van Breugel and J. Worrell. Towards quantitative verification of probabilistic transition systems. In F. Orejas, P.G. Spirakis, and J. van Leeuwen, editors, *Proc. 28th ICALP'01*, pages 421 – 432. LNCS 2076, 2001.

[54] F.C. van Breugel. *Topological Models in Comparative Semantics.* PhD thesis, Vrije Universiteit, September 1994.

[55] F.C. van Breugel. An introduction to metric semantics: Operational and denotational models for programming and specification languages. *Theoretical Computer Science*, 258:1–98, 2001.

[56] L. Castellano, G. De Michelis, and L. Pomello. Concurrency vs. interleaving: an instructive example. *Bulletin of the EATCS*, 31:12–15, 1987.

[57] I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In *Proc. CONCUR'90*, pages 126–140. LNCS 458, 1990.

[58] I. Christoff. *Testing Equivalences for Probabilistic Processes.* PhD thesis, Uppsala University, 1990.

[59] L. Christoff. *Specification and Verification Methods for Probabilistic Processes.* PhD thesis, Uppsala University, 1993.

[60] E. Clarke, M. Fujita, P. McGeer, J. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. In *Proc. IWLS'93: International Workshop on Logic Synthesis*, 1993.

[61] R. Cleaveland, Z. Dayar, S. Smolka, and S. Yuen. Testing preorders for probabilistic processes. *Information and Computation*, 154(2):93–148, November 1999.

[62] S.A. Cook. Soundness and completeness of an axiom system for program verification. *SIAM Journal of Computing*, 7(1):70–90, 1978.

[63] J.P. Courtiat and D.E. Saidouni. Relating maximality-based semantics to action refinement in process algebras. In D. Hogrefe and S. Leue, editors, *Formal Description Techniques VII*, pages 292–308. Chapman & Hall, 1995.

[64] P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems.* PhD thesis, University of Twente, 1999.

[65] P.R. D'Argenio. A compositional translation of stochastic automata into timed automata. Technical Report CTIT 00-08, University of Twente, 2000.

[66] P.R. D'Argenio and E. Brinksma. A calculus for timed automata (Extended abstract). In B. Jonsson and J. Parrow, editors, *Proceedings of the 4th International School and Symposium on Formal Techniques in Real Time and Fault Tolerant Systems,* Uppsala, Sweden, volume 1135 of *Lecture Notes in Computer Science*, pages 110–129. Springer-Verlag, 1996.

[67] P.R. D'Argenio, H. Hermanns, and J.-P. Katoen. On generative parallel composition. In *Proc. Probmiv'98*, pages 105–122. ENTCS 22, 1999.

[68] P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. A stochastic automata model and its algebraic approach. In *Proc. PAPM'97*, pages 1–16. CTIT Technical Report no. 97–14, University of Twente, 1997.

[69] P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. An algebraic approach to the specification of stochastic systems (extended abstract). In D. Gries and W.-P. de Roever, editors, *Proceedings of the IFIP Working conference on Programming Concepts and Methods, PROCOMET'98,* Shelter Island, New York, USA, IFIP Series, pages 126–147. Chapman & Hall, 1998.

[70] P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. A compositional approach to generalised semi-Markov processes. In *Proc. WODES'98.* IEE, 1998.

[71] P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. A stochastic process algebra for discrete event simulation. Technical report, University of Twente, 1998.

[72] P. Darondeau and P. Degano. Causal trees. In G. Ausiello, M. Dezani-Ciancaglini, and S. Ronchi Della Rocca, editors, *Proc. ICALP'89*, pages 234–248. LNCS 372, 1989.

[73] P. Darondeau and P. Degano. Refinement of actions in event structures and causal trees. *Theoretical Computer Science*, 118:21–48, 1993.

[74] P. Degano and R. Gorrieri. A causal operational semantics of action refinement. *Information and Computation*, 122:97–119, 1995.

[75] J. Desharnais, A. Edalat, and P. Panangaden. A logical characterization of bisimulation for labeled Markov processes. In *Proc. LICS'98*, pages 478–487, Indianapolis, 1998.

[76] J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labelled Markov processes. *Information and Computation*, 2001.

[77] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Metrics for labeled Markov systems. In J.C.M. Baeten and S. Mauw, editors, *Proc. CONCUR'99*, pages 258–273. LNCS 1664, 1999.

[78] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden. Approximating labeled Markov processes. In *Proc. LICS 2000*, pages 95–106, 2000.

[79] J. Dugundji. *Topology.* Allyn and Bacon, 1976.

[80] U.H. Engberg. *Partial Orders and Fully Abstract Models for Concurrency.* PhD thesis, Aarhus University, 1990. Also published as Technical Report DAIMI PB–307, Computer Science Department, Aarhus University 1990.

[81] R. Engelking. *General Topology.* Sigma Series in Pure Mathematics 6, Heldermann Verlag, revised and completed edition, 1989.

[82] W.J. Fokkink. *Introduction to Process Algebra.* Texts in Theoretical Computer Science, An EATCS Series. Springer-Verlag, 2000.

[83] W.J. Fokkink and A.S. Klusener. An effective axiomatization for real time ACP. *Information and Computation*, 122(2):286–299, 1995.

[84] N. Francez. *Program Verification.* International Computer Science Series. Addison-Wesley, 1992.

[85] H. Gaifman. Modeling concurrency by partial orders and nonlinear transition systems. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *REX'88*, pages 467–488. LNCS 354, 1989.

[86] A. Giacalone, C. Jou, and S.A. Smolka. Algebraic reasoning for probabilisitic concurrent systems. In M. Broy and C.B Jones, editors, *Proc. Working Conference on Programming Concepts and Methods*, pages 443–458. IFIP TC2, Sea of Gallilee, 1990.

[87] W. Gibbs. Software's chronic crisis. *Scientific American*, September 1994.

[88] R.J. van Glabbeek. The linear time - branching time spectrum (extended abstract). In J.C.M. Baeten and J.W. Klop, editors, *Proc. CONCUR'90*, pages 278–298. LNCS 458, 1990.

[89] R.J. van Glabbeek. The linear time - branching time spectrum II: the semantics of sequential systems with silent moves (extended abstract). In E. Best, editor, *Proc. CONCUR'93*, pages 66–81. LNCS 715, 1993.

[90] R.J. van Glabbeek. *Comparative Concurrency Semantics and Refinement of Actions.* CWI Tract 109. CWI, 1996. Revision of Ph.D thesis of the same title.

[91] R.J. van Glabbeek and J.J.M.M. Rutten. The processes of De Bakker and Zucker represent bisimulation equivalence classes. In *J.W. de Bakker 25 Jaar Semantiek, Liber Amicorum*, pages 243–246. CWI, Amsterdam, 1989.

[92] R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative and stratified models of probabilistic processes. *Information and Computation*, 121:59–80, 1995.

[93] R.J. van Glabbeek and F.W. Vaandrager. Petri net models for algebraic theories of concurrency. In J.W. de Bakker, A.J. Nijman, and P.C. Treleaven, editors, *PARLE'87, volume II*, pages 224–242. LNCS 259, 1987.

[94] U. Goltz, R. Gorrieri, and A. Rensink. Comparing syntactic and semantic action refinement. *Information and Computation*, 125:118–143, 1996.

[95] R. Gorrieri. *Refinement, Atomicity and Transactions for Process Description Languages.* PhD thesis, University of Pisa, 1991. Also available as Technical Report TD–2/91, Dipartimento di Informatica, Università degli Studi di Pisa.

[96] R. Gorrieri and C. Laneve. Split and ST bisimulation semantics. *Information and Computation*, 118:272–288, 1995.

[97] R. Gorrieri and A. Rensink. Action refinement. In J.A. Bergstra, A.J. Ponse, and S.A. Smolka, editors, *Handbook of process algebra*, chapter 16. Elsevier, Amsterdam, 2001.

[98] J. Grabowski. On partial languages. *Fundamenta Informaticae*, 4(1):427–498, 1981.

[99] C. Gregorio-Rodríguez and M. Núñez. Denotational semantics for probabilistic refusal testing. In M. Huth and M.Z. Kwiatkowska, editors, *Proc. Probmiv'98.* ENTCS 22, 1998.

[100] R. Gupta, S. Bhaskar, and S.A. Smolka. On randomization in sequential and distributed algorithms. *ACM Computing Surveys*, 26(1):7–86, 1994.

[101] Anthony Hall. Seven myths of formal methods. *IEEE Software*, 7(5):11–19, September 1990.

[102] P.R. Halmos. *Measure Theory.* Van Nostrand, 1950.

[103] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proc. 11th IEEE Real-Time Systems Symposium*, Orlando, Florida, 1990.

[104] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, 6:512–535, 1994.

[105] H.A. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Uppsala University, 1991.

[106] J.I. den Hartog. Comparative semantics for atomization and action refinement with synchronization. Master's thesis, Vrije Universiteit Amsterdam, 1996.

[107] J.I. den Hartog. Comparative semantics for a process language with probabilistic choice and non-determinism. Technical Report IR–445, Vrije Universiteit, Amsterdam, February 1998.

[108] J.I. den Hartog. Verifying probabilistic programs using a Hoare like logic. In P.S. Thiagarajan and R. Yap, editors, *LNCS 1742 (ASIAN'99)*, pages 113–125. Springer, 1999.

[109] J.I. den Hartog and E.P. de Vink. Mixing up nondeterminism and probability: A preliminary report. In *Proc. Workshop on Probabilistic Methods in Verification (PROBMIV'98)*, volume 22 of *Electronic Notes in Theoretical Computer Science.* Elsevier, 1999.

[110] J.I. den Hartog and E.P. de Vink. Taking chances on $\|$ and *fail*: Extending strong and probabilistic bisimulation. Technical Report IR–454, Vrije Universiteit, Amsterdam, March 1999.

[111] J.I. den Hartog and E.P. de Vink. Verifying probabilistic programs using a Hoare like logic. *International Journal of Foundations of Computer Science*, 13(3):315–340, 2002.

[112] J.I. den Hartog, E.P. de Vink, and J.W. de Bakker. Full abstractness of a metric semantics for action refinement. *Fundamenta Informaticae*, 40:335–382, 1999.

[113] J.I. den Hartog, E.P. de Vink, and J.W. de Bakker. Metric semantics and full abstractness for action refinement and probabilistic choice. In *Proc. MFCSIT 2000.* ENTCS 40, 2001.

[114] V. Hartonas-Garmhausen. *Probabilistic Symbolic Model Checking with Engineering Models and Applications.* PhD thesis, Carnegie Mellon University, 1998.

[115] M. Hennessy. Axiomatising finite concurrent processes. *SIAM Journal on Computing*, 17:997–1017, 1988.

[116] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35:63–67, 1990.

[117] H. Hermanns. *Interactive Markov Chains.* PhD thesis, University of Erlangen-Nurnberg, July 1998.

[118] H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274 (1-2):43–87, 2002.

[119] H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the tipptool. *Performance Evaluation*, 39:5–35, 2000.

[120] H. Hermanns, U. Herzog, U. Klehmet, M. Siegle, and V. Mertsiotakis. Compositional performance analysis with the tipptool. In LNCS 1469, editor, *10th Int. PERFORMANCE TOOLS'98 Conference*, pages 51–62, 1998.

[121] H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic process algebras – between LOTOS and Markov chains. *Computer Networks and ISDN Systems*, 30(9-10):901–924, 1998.

[122] H. Hermanns and J.-P. Katoen. Automated compositional Markov chain generation for a plain-old telephone system. *Science of Computer Programming*, 36(1):97 – 127, 2000.

[123] H. Hermanns and M. Siegle. Tipptool: Compositional specification and analysis of Markovian performance models. In *Computer Aided Verification (CAV'99)*, pages 487–490. LNCS 1633, 1999.

[124] U. Herzog. Formal description, time and performance analysis: A framework. Technical Report 15/90 IMMD VII, Friedrich-Alexander-University, Erlangen-Nürnberg, Germany, September 1990.

[125] J. Hillston. *A Compositional Approach to Performance Modelling.* PhD thesis, Cambridge University Press, 1996. Distinguished Dissertation in Computer Science.

[126] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.

[127] C.A.R. Hoare. *Communicating Sequential Processes.* Series in Computer Science. Prentice-Hall International, Englewood Cliffs, NJ, 1985.

[128] J. Hooman. Program design in PVS. In *Proceedings Workshop on Tool Support for System Development and Verification*, June 1996.

[129] M. Huth and M.Z. Kwiatkowska. Quantitative analysis and model checking. In *Proc. LICS'97*, pages 111–122, Warsaw, 1997.

[130] G. G. Infante López, H. Hermanns, and J.-P. Katoen. Beyond memoryless distributions: Model checking semi-Markov chains. In L. de Alfaro and S. Gilmore, editors, *Proc. PAPM-PROBMIV 2001*, pages 23–38. LNCS 2165, 2001.

[131] B. Jacobs and J.J.M.M. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the EATCS*, 62, 1997.

[132] L. Jategaonkar and A. Meyer. Testing equivalence for Petri nets with action refinement: preliminary report. In R. Cleaveland, editor, *Proc. CONCUR'92*, pages 17–31. LNCS 630, 1992.

[133] C. Jones. *Probabilistic Nondeterminism.* PhD thesis, ECS–LFCS–90–105, University of Edinburgh, 1990.

[134] C. Jones and G. Plotkin. A probabilistic powerdomain of evaluations. In *Proc. LICS'89*, pages 186–195. Asilomar, 1989.

[135] B. Jonsson and K. Larsen. Specification and refinement of probabilistic processes. In *Proc. LICS'91*, pages 266–277, Amsterdam, 1991.

[136] C.C. Jou and S.A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In *Proc. CONCUR'90*, pages 367–383. LNCS 458, 1990.

[137] J.-P. Katoen. *Quantitative and Qualitative Extensions of Event Structures.* PhD thesis, University of Twente, 1996. CTIT PhD-thesis series No. 96–09.

[138] J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic ctmc model checking. In L. de Alfaro and S. Gilmore, editors, *Proc. PAPM-PROBMIV 2001*, pages 23–38. LNCS 2165, 2001.

[139] J.N. Kok and J.J.M.M. Rutten. Contractions in comparing concurrency semantics. *Theoretical Computer Science*, 76:179–222, 1990. Extended abstract in Proc. ICALP'88, T. Lepistö and A. Salomaa (eds.), LNCS 317, pp. 317–332, 1988.

[140] D. Kozen. Semantics of probabilistic programs. *Journal of Computer and System Sciences*, 22:328–350, 1981.

[141] D. Kozen. A probabilistic PDL. *Journal of Computer and System Sciences*, 30:162–178, 1985.

[142] M. Kwiatkowska and G. Norman. Probabilistic metric semantics for a simple language with recursion. In W. Penczek and A. Szalas, editors, *Proc. MFCS'96*, pages 419–430. LNCS 1113, 1996.

[143] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with prism: A hybrid approach. Technical Report CSR-01-10, School of Computer Science, University of Birmingham, 2001.

[144] M. Kwiatkowska, G. Norman, R. Segala, and J. Spronston. Verifying quantitative properties of continuous probabilistic timed automata. In C. Palamadessi, editor, *Proc. CONCUR 2000*, pages 123–137. LNCS 1877, 2000.

[145] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. In J.-P. Katoen, editor, *Proc. ARTS'99*, pages 75–95. LNCS 1601, 1999.

[146] M. Kwiatkowska, G. Norman, and J. Spronston. Probabilistic model checking of deadline properties in the IEEE1394 firewire root contention protocol. In S. Maharaj, C. Shankland, and J.M.T. Romijn, editors, *Proc. International Workshop on Application of Formal Methods to IEEE 1394 Standard*, 2001.

[147] M.Z. Kwiatkowska and G.J. Norman. A fully abstract metric-space denotational semantics for reactive probabilistic processes. In *Proc. Express'98.* ENTCS 13, 1998.

[148] M.Z. Kwiatkowska and G.J. Norman. A testing equivalence for reactive probabilistic processes. In Ilaria Castellani and Catuscia Palamidessi, editors, *Electronic Notes in Theoretical Computer Science.* ENTCS 16(2), 2000.

[149] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94:1–28, 1991.

[150] K.G. Larsen and A. Skou. Compositional verification of probabilisitic processes. In *LNCS 630 (CONCUR'92)*, pages 456–471. Springer, 1992.

[151] D. Latella and P. Quaglia. Stochastic analysis via a probabilistic process algebra. In *Proc. PAPM'97*, pages 187–206. CTIT Technical Report no. 97–14, University of Twente, 1997.

[152] R. Loogen and U. Goltz. Modelling non-deterministic concurrent processes with event structures. *Fundamenta Informaticae*, 14:39–73, 1991.

[153] G. Lowe. *Probabilities and Priorities in Timed CSP.* PhD thesis, University of Oxford, 1993.

[154] G. Lowe. Representing nondeterminism and probabilistic behaviour in reactive processes. Technical Report PRG–TR–11–93, Oxford University Computing Laboratory, 1993.

[155] G. Lowe. Probabilistic and prioritized models of timed CSP. *Theoretical Computer Science*, 138:315–352, 1995. Special issue on the Mathematical Foundations of Programming Semantics conference, 1992.

[156] N. Lynch. *Distributed Algorithms.* The Morgan Kaufmann series in data management systems. Kaufmann, 1996.

[157] N.A. Lynch, R. Segala, and F.W. Vaandrager. Hybrid i/o automata revisited. In M.D. Di Benedetto and A.L. Sangiovanni-Vincetelli, editors, *Proc. HSCC'01.* LNCS 2034, 2001.

[158] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid i/o automata. Technical report, University of Nijmegen, April 1999.

[159] R. Milner. *A Calculus of Communicating Systems.* LNCS 92, 1980.

[160] R. Milner. *Communication and Concurrency*. Series in Computer Science. Prentice-Hall International, Englewood Cliffs, NJ, 1989.

[161] M. Mislove. Nondeterminism and probabilistic choice: Obeying the laws. In C. Palamadessi, editor, *Proc. CONCUR 2000*, pages 350–364. LNCS 1877, 2000.

[162] C. Morgan and A. McIver. pGCL: formal reasoning for random algorithms. *Proc. of WOFACS 1998, Special Issue of the South African Computer Journal*, 22:14–27, 1999.

[163] C. Morgan, A. McIver, and K. Seidel. Probabilistic predicate transformers. *ACM Transactions on Programming Languages and Systems*, 18(3):325–353, May 1996.

[164] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[165] M Nielsen, U. Engberg, and K.S. Larsen. Fully abstract models for a process language with refinement. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Proc. REX Workshop on Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 523–548. LNCS 354, 1988.

[166] M. Nivat. Infinite words, infinite trees, infinite computations. In J.W. de Bakker and J. van Leeuwen, editors, *Foundations of Computer Science III, part 2: Languages, Logic, Semantics*, volume 109 of *Mathematical Centre Tracts*, pages 3–52. Mathematical Centre, Amsterdam, 1979.

[167] G.J. Norman. *Metric Semantics for Reactive Probabilistic Systems*. PhD thesis, University of Birmingham, 1997.

[168] M. Núñez, D. de Frutos, and L. Llana. Acceptance trees for probabilistic processes. In I. Lee and S.A. Smolka, editors, *Proc. CONCUR'95*, pages 249–263. LNCS 962, 1995.

[169] P. Panangaden. Stochastic techniques in concurrency, 1997. Notes from Aarhus Fall 1996 and EATCS Summer School Udine 1997.

[170] P. Panangaden. Measure and probability for concurrency theorists. *Theoretical Computer Science*, 253(2), 2001.

[171] D.M.R. Park. Concurrency and automata on infinite sequences. In *Proc. 5th GI Conference on Theoretical Computer Science*, pages 167–183. LNCS 104, 1981.

[172] A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In C. Palamadessi, editor, *Proc. CONCUR 2000*, pages 334–349. LNCS 1877, 2000.

[173] G.D. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University, 1981.

[174] A. Pnueli and L.D. Zuck. Probabilistic verification. *Information and Computation*, 103:1–29, 1993.

[175] A. Rensink. *Models and Methods for Action Refinement.* PhD thesis, University of Twente, 1993.

[176] A. Rensink. An event-based SOS for a language with refinement. In J. Desel, editor, *Structures in Concurrency Theory*, Workshops in Computing, pages 294–309. Springer, 1995.

[177] J. Rutten and D.Turi. On the foundation of final semantics: non-standard sets, metric spaces, partial orders. In *LNCS 666 (Proceedings of the REX Workshop on Semantics: Foundations and Applications)*, pages 477–530. Springer, 1992.

[178] J.J.M.M. Rutten. Correctness and full abstraction of metric semantics for concurrency. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, pages 628–659. LNCS 354, 1989.

[179] J.J.M.M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000.

[180] J.J.M.M. Rutten and D. Turi. Initial algebra and final coalgebra semantics for concurrency. In J.W. de Bakker, W.-P. de Roever, and G. Rozenberg, editors, *Proc. REX Workshop "A Decade of Concurrency"*, pages 530–582. LNCS 803, 1994.

[181] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Massachusetts Institute of Technology, June 1995.

[182] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In B. Jonsson and J. Parrow, editors, *Proc. CONCUR'94*, pages 481–495. LNCS 836, 1994.

[183] K. Seidel. Probabilistic communicating processes. *Theoretical Computer Science*, 152:219–249, 1995.

[184] M.I.A. Stoelinga. Fun with FireWire: Experiments with verifying the IEEE1394 root contention protocol. In S. Maharaj, C. Shankland, and J.M.T. Romijn, editors, *Proceedings of the International Workshop on Application of Formal Methods to the IEEE1394 Standard*, pages 35–38, 2001. Also, Technical Rapport CSI-R0107, Computing Science Institute, University of Nijmegen, March 2001.

[185] M.I.A. Stoelinga. *Verification of Probabilistic, Real-Time and Parametric Systems.* PhD thesis, University of Nijmegen, 2002.

[186] V. Stoltenberg-Hansen and J.V. Tucker. *Effective algebras*, volume 4 of *Handbook of Logic in Computer Science.* Oxford science publications, 1995.

[187] J.E. Stoy. *Denotational Semantics—the Scott-Strachey approach to programming language theory.* MIT Press, 1977.

[188] E.P. de Vink. On a functor for probabilistic bisimulation and preservation of weak pullbacks. Technical Report IR–444, Vrije Universiteit, Amsterdam, 1998.

[189] E.P. de Vink. A note on the completeness of $\mathcal{M}_1(X)$. http://www.win.tue.nl/∼evink/note.ps, 2000.

[190] E.P. de Vink and J.J.M.M. Rutten. Bisimulation for probabilistic transition systems: a coalgebraic approach. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Automata, Languages and Programming*, pages 460–470. LNCS 1256, 1997. Proc. 24th ICALP'97, Bologna, Italy.

[191] E.P. de Vink and J.J.M.M. Rutten. Bisimulation for probabilistic transition systems: a coalgebraic approach. *Theoretical Computer Science*, 221:271–293, 1999.

[192] W. Vogler. Failures semantics based on interval semiwords is a congruence for refinement. *Distributed Computing*, 4:139–162, 1991.

[193] W. Vogler. *Modular Construction and Partial Order Semantics of Petri Nets.* LNCS 625, 1992.

[194] W. Vogler. Bisimulation and action refinement. *Theoretical Computer Science*, 114:173–200, 1993.

[195] W. Vogler. The limit of split$_n$-language equivalence. *Information and Computation*, 127:41–61, 1996.

[196] J.H.A. Warmerdam. Case studies in true concurrency and logic programming semantics. Master's thesis, Vrije Universiteit, October 1989.

[197] G. Winskel. *The Formal Semantics of Programming Languages. An Introduction.* The MIT Press, 1993.

[198] G. Winskel and M. Nielsen. *Models for Concurrency*, volume 4 of *Handbook of Logic in Computer Science*, pages 1–148. Oxford science publications, 1995.

[199] Sue-Hwey Wu, S.A. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. *Theoretical Computer Science*, 176:1–38, 1997.

[200] W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. In *Proc. PSTV XII*, pages 47–61. North-Holland, 1992.

# Samenvatting

## Probabilistische Uitbreidingen van Semantische Modellen

Het thema van dit proefschrift is het modelleren van en redeneren over probabiliteit in computer systemen. Kansen spelen een rol in computer systemen met inherent probabilistisch gedrag, zoals bijvoorbeeld een communicatiekanaal dat niet volledig betrouwbaar is, alsmede in probabilistische algoritmen waarin expliciet kansen worden geintroduceerd. Een probleem met probabilistische systemen is dat deze vaak lastig te testen zijn en de intuitie gemakkelijk misleiden, zie e.g. het 'drie-deuren-probleem' besproken in hoofdstukken 4 en 6, Het gebruik van wiskundige modellen en formeel redeneren helpt fouten te voorkomen.

Verscheidene semantische modellen voor procestalen met probabiliteit worden in dit proefschrift geïntroduceerd in een metrische setting. Ook wordt een Hoare-stijl logica voor het redeneren over programma's met probabilistische keuzes gegeven. De metrische eigenschappen van de domeinen die gebruikt worden voor de semantische modellen zijn nuttig in definities van semantiek en operaties alsmede voor het redeneren over (de betekenis) van programma's. De basis voor de modellering van probabilistische keuze wordt gelegd in hoofdstuk 3. In dit hoofdstuk wordt een eenvoudige taal $\mathcal{L}_p$ bestudeerd die naast probabiliteit als hoofdconstructies sequentiele compositie en recursie heeft. Een operationeel model, gebaseerd op een transitiesysteem en een compositioneel denotationeel model worden gegeven en vergeleken. In hoofdstukken 4, 5 en 7 worden verdere uitbreidingen van de taal $\mathcal{L}_p$ bestudeerd met, respectievelijk, de constructies niet-deterministische keuze en parallellisme, actieverfijning, en niet-discrete probabilistische keuze.

De uitbreiding met niet-deterministiche keuze en parallellisme in hoofdstuk 4 blijkt subtiel. Verschillende modellen zijn nodig te zijn voor verschillende mogelijke interpretaties van de niet-deterministiche keuze als mede de probabilistische keuze. Metrische modellen voor actieverfijning, zeker in een 'interleaving' setting, zijn minder bestudeerd dan modellen voor niet-deterministiche keuze. In hoofdstuk 5 wordt daarom eerst in sectie 5.2 een metrisch model ontwikkeld voor actieverfijning zonder probabilistische keuze in een interleaving setting. Vervolgens wordt in sectie 5.3 een model voor actieverfijning en probabilistische keuze geven door de technieken van hoofdstuk 3 en sectie 5.2 te combineren.

In hoofdstuk 6 wordt Hoare logica uitgebreid om het direct redeneren over probabilistische programma's en probabilistische programma toestanden mogelijk te maken. In een deterministische toestand is een eigenschap vervuld of niet. In een probabilistische toestand is een eigenschap vervuld met een bepaalde kans. Een gelaagde constructie wordt gebruikt voor probabilistische predikaten om over kansen te kunnen redeneren terwijl toch de waar of niet waar interpretatie van predikaten gehandhaafd blijft.

Hoofdstuk 7 introduceert een taal waarbij mogelijk een keuze gemaakt moet worden tussen oneindig veel mogelijkheden. Een metrische versie van stochastische kernen wordt ontwikkeld om voor deze taal een semantisch model te kunnen geven.

## Titles in the IPA Dissertation Series