Reconciling Nondeterministic and Probabilistic Choices

Copyright © 2006, Ling Cheung, Nijmegen
ISBN-10: 90-9020814-3
ISBN-13: 978-90-9020814-5
IPA Dissertation Series 2006-12

Typeset with LaTeX2e
Printed by PrintPartners Ipskamp, Enschede
Cover by Jesse Hughes

The work on this thesis has been carried out under the auspices of the research school IPA (Institute for Programming research and Algorithmics). The author has also received partial support from DFG/NWO bilateral cooperation project Validation of Stochastic Systems (VOSS,VOSS2).

Reconciling Nondeterministic and Probabilistic Choices

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

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de Rector Magnificus prof.dr. C.W.P.M. Blom,
volgens besluit van het College van Decanen
in het openbaar te verdedigen op maandag 18 september 2006
des namiddags om 3.30 uur precies

door

Ling Cheung
geboren op 12 juni 1978
te Canton, China

Promotor:

Prof. dr. Frits W.Vaandrager

Manuscriptcommissie:

Prof. dr. James Aspnes (Chair), Yale University
Prof. dr. Marta Kwiatkowska, University of Birmingham
Prof. dr. Nancy A. Lynch, Massachusetts Institute of Technology

Contents

Preface

I have come a long way to write this thesis.

My highest gratitude goes to my parents, whose influence and support have been indispensable throughout my education. This thesis is no less an accomplishment of theirs than it is my own. My gratitude also goes to all of my acquaintances at Carnegie Mellon University in the departments of Civil and Environmental Engineering, Mathematics and Philosophy. Without their warm support and encouragement, I would not have chosen to pursue a career in academics.

During my first (aimless) year in Nijmegen, I was very fortunate to be able to study with Henk Barendregt. Later I joined his Foundations group, where I also studied with Jan Willem Klop. Henk and Jan Willem sparked my interest in concurrency theory and process algebra, which eventually led to the research reported in this thesis. I thank them, as well as all (former) members of the Foundations group, for their hospitality and inspirations.

Then I must thank my promotor and supervisor Frits Vaandrager. Over the past three years, Frits introduced me to various topics in formal verification and taught me how to do research. All of my writings came under and benefited from his scrutiny. He also encouraged me to attend workshops, conferences and summer schools, which broadened my exposure and helped me establish contacts outside of Nijmegen. Last but not least, his humble and attentive personality made him a dear friend, even more so than a teacher.

This thesis is a collection (and an extension) of various papers I wrote in the past three years. I thank my coauthors for their contribution in these papers and for all the things I have learnt from them. They are: Ran Canetti, Martijn Hendriks, Jesse Hughes, Dilsun Kaynar, Moses Liskov, Nancy Lynch, Olivier Pereira, Roberto Segala, Mariëlle Stoelinga and Frits Vaandrager. The same goes to the members of my manuscript committee, James Aspnes, Marta Kwiatkowska and Nancy Lynch, for the time and effort they committed to reading this thesis.

I am also grateful to all of my friends and colleagues in Holland, especially those in Nijmegen on the sixth floor. They are always supportive, both academically and personally. Without their kindness it would have been much more difficult to survive in a country as exotic as Holland.

Finally, I want to thank Jesse and Quincy.

To Quincy, I apologize for the numerous times that I said "Mama is busy." Thank you for being a very sweet, loving and understanding boy. I hope I have struck a reasonable balance between family and work, and you will grow up to remember a warm and happy childhood.

To Jesse, I am grateful for the many roles you play in my life: lover, companion, babysitter, chauffeur, system administrator, TeX support, ... Thank you for being with me.

June 2006, Boston

1

Introduction

> **Formal Methods**: mathematically based techniques for the specification, development and verification of software and hardware systems.
>
> *FOLDOC* [FOL]

This thesis is written in the broad context of formal methods, which emerged as a branch of computer science aimed at improving the quality of computer systems, especially those in safety-critical applications. As we have experienced in the past few decades, the complexity of these systems grows very rapidly, owing to increased demand as well as advances in computing capabilities. Formal methods research takes on the challenge of improving such systems, by providing a wide range of tools and techniques to manage complexity.

Formal methods have been studied for all stages of system development: design, implementation and analysis. Arguably, the most effective application of formal methods occurs in the design stage, where formal languages are used to give a precise description of the desired system. Much work has also been done to streamline the transition from specification to implementation. In both operational and denotational settings, for example, formal specifications can be refined in an incremental fashion, until they reach a level of detail suitable for realization in actual software or hardware.

Mathematical rigor receives the greatest attention during the final stage, analysis and verification. The promise here is that mathematical proofs guarantee correct behavior in real life. Therefore we try to provide clear, mathematical semantics for both formal specifications and actual implementations. To facilitate reasoning, inference rules are developed and proven sound with respect to the proposed semantics, allowing us to move between different levels of abstraction.

Most of this Ph.D. work is carried out in the context described in the preceding paragraph. More specifically, we study semantic models for randomized, distributed computation. The basic approach is to extend well-establish theories of distributed computation with probabilistic elements. Aside from technical complications, the involvement of randomization poses an additional challenge: justifying, somehow, the leap from mathematics to the physical reality that we claim to model.

Probabilities and Nondeterminism

A large number of modeling frameworks have been proposed or adapted for the purpose of analyzing stochastic behavior in computer systems. For example, the traditional theory of (discrete- and continuous-time) *Markov chains* finds many applications in the area of performance and reliability analysis [KS76, Ste94, Hav98, Tri02].

Such applications often follow a common modeling paradigm: probability distributions are used to model uncertainties in the computation environment, for example, the arrival rate of jobs and the size and complexity of each job (measured in terms of processing time). We are then interested in estimating parameters such as expected waiting time and percentage of missed deadlines over a given period of time. To do so, one often needs to limit the types of probability distributions to a few well-behaved families, such as Poisson and exponential. Without these restrictions, the availability of solution techniques becomes very limited.

The models considered in this thesis are developed in a different tradition, namely, the analysis of distributed algorithms. Here randomness is used by the processes themselves to achieve certain goals. For instance, processes cast randomly generated votes to reach consensus, or they choose a neighbor at random to propagate information without flooding the network. In this setting, the computation environment is extremely unpredictable and it does not always makes sense to assume a fixed pattern of events.

For example, we may not have a good reason to believe that message delays experienced on a network can be faithfully modeled by an exponential distribution with parameter λ. Or that each of m pending messages are equally likely to be the first one delivered. Furthermore, when we do make such underlying assumptions, it is often impractical to perform experiments on an actual system so that our assumptions can be validated. Without this last step, it is difficult to assess what contributions we have actually made by performing a formal analysis.

Thus, nondeterminism is often considered a "safer" option, when we have no information over certain aspects of the object system and/or the external environment. Moreover, nondeterministic choices allow us to model systems at higher levels of abstraction, leaving many implementation details unspecified. These reasons motivate the development of many models that combine nondeterministic and probabilistic behaviors [Var85, PZ86, CY90, YL92, PZ93, SL95, BdA95, DEP02], as alternatives to purely probabilistic models like Markov chains. Many of these newer models are based on the theory of *Markov decision processes (MDPs)*, which is used extensively in planning and artificial intelligence [Put94, Ber95]. The models studied in the thesis also fall into this category.

While the presence of nondeterminism is desirable for modeling purposes, it

leads to complications in semantic definitions and analysis. Namely, in order to obtain well-defined probability distributions from a specification that contains both nondeterministic and probabilistic choices, one must somehow "untangle" these two types of choices. This is usually done by means of *adversaries/schedulers*[1], which resolve all nondeterministic choices in a specification.

In our view, the resolution of nondeterministic choices (also called "scheduling") is a fundamental issue in semantic studies, because different scheduling mechanisms associate different sets of probability distributions to a particular specification. That implies the same specification may have different properties depending on how nondeterministic choices are resolved. Therefore, we feel the need to devote more attention to the notion of adversaries. (Hence the title "Reconciling Nondeterministic and Probabilistic Choices".)

In particular, we study mathematical properties of adversaries, which are formalized as functions mapping execution histories to available next transitions. Moreover, we try to understand how different definitions of parallel composition translate into different assumptions on the behavior of adversaries. This allows us to identify some key properties of adversaries (e.g., history-dependence) that affect compositionality of trace-style semantics. Last but not least, we try to make a connection between the notions of adversaries captured by our formal definitions and those that are actually used in distributed computing, for example, in the areas of security protocols and randomized consensus.

Main Topics and Contributions

This thesis is organized into three parts. In Part I, we work with Segala's (simple) *Probabilistic Automata (PA)* model [Seg95b] and prove many technical theorems regarding adversaries and their induced probability distributions. These results are then used to extend the testing semantics proposed by Stoelinga and Vaandrager [SV03]. In Part II, we introduce our own variant of *Probabilistic Input/Output Automata (PIOA)* and use that as a basis of two specialized models, both of which come with a compositional trace-style semantics. Finally, Part III presents a randomized consensus algorithm, together with a manual correctness proof and a mechanized analysis using the probabilistic model checker PRISM [PRI].

These three parts are technically independent and can be read in any order. Below we summarize each of them in greater detail.

Part I

The PA model is a straightforward extension of the familiar notion of labeled transition systems. In particular, every automaton has a discrete state space and

[1] These are referred to as *policies* in the setting of MDPs.

discrete transitions from each state. If multiple transitions are available from the same state, then the choice among them is nondeterministic. Probabilities are introduced within transitions; namely, the target of each transition is a discrete probability distribution on next states, as opposed to a single next state.

In this setting, an adversary is a function that maps finite computation paths to available next transitions. Such a function induces a probability distribution on computation paths (here called a *probabilistic execution*) and hence a probability distribution on traces (here called a *trace distribution*). These probability distributions can be viewed very naturally as trees with probabilistic branching, and they are the main objects of study in Part I.

In Chapter 4, we give an explicit characterization of probabilistic executions, which allows us to manipulate probabilistic executions directly, without reference to the adversaries that induce them. This result is used in two useful constructions: convex combinations and limits.

Chapter 5 is concerned with finite approximation of infinite behavior, which is a main theme of Part I. Specifically, we define order structures on three levels: adversaries, probabilistic executions and trace distributions. The orderings we use are quite unusual, in that they are *not* based on the $\leq$ relation on real numbers. Instead, they are "flat" in nature, resembling the prefix relation on sequences and the subtree relation based on truncation of branches. This gives rise to algebraic order structures, whose compact elements can be characterized in a very natural way. (In contrast, orderings induced by the usual $\leq$ relation on real numbers are not algebraic.)

The limit construction of Chapter 4 is used in Chapter 5 to prove existence of least upper bounds. Another application of the same construction is found in Chapter 6, where trace distributions are viewed as points in an appropriate metric space and we show that the limit of a convergent sequence of trace distributions is again a trace distribution.

Finally, Chapter 7 extends the testing scenario of [SV03] with a notion of finite tests. We prove that, for all image finite processes, our testing equivalence coincides with the trace distribution equivalence of [Seg95b]. This is an improvement over [SV03], which requires that processes are finitely branching. A good portion of the technical machineries developed in previous chapters are applied here, showing that any infinite behavior can be reduced to its finite "sub-behaviors" and hence finite tests have sufficient distinguishing power over infinite processes.

Although Part I is written for the PA model, our technical developments actually take place in the more fundamental settings of ordered sets and metric spaces. Therefore, we believe that the same ideas and results can be adapted to other settings, where semantic objects of interest are probabilistic trees similar to our probabilistic executions and trace distributions.

Acknowledgment Chapter 7 is joint work with Mariëlle Stoelinga and Frits Vaandrager. The other chapters in Part I have also benefited from their com-

ments and suggestions. We thank them for many interesting conversations and for their help in preparing the joint paper [CSV06].

Part II

The trace distribution equivalence and testing equivalence discussed in Part I are examples of *semantic relations*. These relations provide a formal way of specifying which aspects in the behavior of a system are considered relevant and which others can be safely ignored. For instance, trace distribution equivalence focuses on correlations between observable actions and ignores internal branching structures, so long as they do not affect action correlations.

In Part II, we turn our attention to parallel composition of probabilistic processes. This operator captures the idea of running several components simultaneously, with possible interactions among them.

The combination of semantic relation and parallel composition forms a basis of many useful methods in formal analysis and verification. For example, a large system can be specified as the parallel composition of smaller and more tangible subsystems. Each subsystems can then be specified at different levels of abstraction, provided these levels fall under a particular semantic relation. Thus, it is important to ensure that properties of the composed system are preserved under substitution of subsystems. This is typically formalized as a *compositionality* theorem.

In a probabilistic setting, parallel composition can be defined in many different ways, depending on which type of adversaries are used to resolve choices between parallel components. We argue that, in order to achieve compositionality in a trace-style setting, adversaries should not have access to local information of individual components. This idea turns out to be difficult to formalize, because it requires a clear distinction between information that is globally available and information that is completely internal to a component.

We propose two frameworks in which local information can be hidden from the adversary. Both of these are based on the PIOA model presented in Chapter 9. Although PIOAs have appeared in many other papers [WSS94, PSL00, BPW04b], we introduce here a new formulation based on reactive and generative transition structures (cf. [vGSS95]). This yields a clear and concise model, which encompasses most existing versions of PIOAs. In addition, we define the notions of input/output (I/O) schedulers and their induced execution trees, generalizing respectively the notions of adversaries and probabilistic executions discussed in Part I. A basic parallel composition operator is also given, which is based on action synchronization and does not attempt to resolve nondeterministic choices between parallel components.

In Chapter 10, we present the *Switched PIOA* model, which uses a token structure to eliminate global nondeterministic choices. This token structure ensures that, at any point in time, there is at most one active component in a system

and this unique component determines the next active component. Thus, global scheduling is performed jointly by all local schedulers, which have access to local information only. We then define the notion of *switched probabilistic systems*, which are switched PIOAs paired with sets of "acceptable" I/O schedulers.

We give a trace-style semantics for switched probabilistic systems, using the notion of likelihood assignments. Likelihood assignments are generalizations of trace distributions of Part I. This semantics is shown to be compositional with respect to a parallel operator that combines local I/O schedulers into a joint I/O scheduler. Thus, the approach taken in Chapter 10 can be characterized as "schedule-and-compose", where local nondeterministic choices are resolved before the components are placed in parallel.

Chapter 11 follows a similar strategy, but without the token structure. Instead, several axioms are imposed on the reactive and generative transition structures, so that branching only occurs when it is meant to be globally visible (i.e., the branches carry different visible action labels). These axioms capture a *local-oblivious* assumption on adversaries, which is well-known in the area of randomized consensus [CIL94, AB04].

For the model of Chapter 11, we also define a trace-style semantics based on likelihood assignments. It is proven to be compositional with respect to a "schedule-and-compose" operator similar to that in Chapter 10. This proof is more involved because the absence of a token structure gives the adversary more freedom in selecting the next transition.

Acknowledgment Chapter 10 is joint work with Roberto Segala, Nancy Lynch and Frits Vaandrager [CLSV04a, CLSV06]. We are greatly indebted to their inputs and encouragements. We also thank Martijn Hendriks for discussions regarding Chapter 11. He is a coauthor of [CH05], which is closely related to Chapter 11.

Part III

Finally, we turn to more practical matters and present a randomized algorithm for asynchronous wait-free consensus using multi-writer multi-reader (MWMR) shared registers. This algorithm is based on earlier work by Chor, Israeli and Li (CIL) and is correct under the assumption that processes can perform a random choice and a write operation in one atomic step. This is a restricted form of the local-oblivious assumption adopted in Chapter 11.

The expected total work for our algorithm is shown to be $O(N \log(\log N))$, compared with $O(N^2)$ for the CIL algorithm, and $O(N \log N)$ for the best known weak adversary algorithm. The efficiency results mainly from the use of MWMR registers, which also allows us to achieve consensus with high probability using $O(\log N)$ space.

In addition to giving a manual correctness proof, we model check instances of our algorithm using the probabilistic model checking tool PRISM.

Acknowledgment Part III is essentially taken from [Che05b]. We thank James Aspnes for his inspiring article [Asp03] and many helpful comments. Also we thank David Parker for support in using PRISM, as well as Jaap-Henk Hoepman and the anonymous referees at OPODIS 05 for their comments and suggestions.

Part I

Basic Theory of Probabilistic Automata

2

Preliminaries

This section provides a summary of basic mathematical notions necessary for our development. In particular, we review materials from real analysis [KF70, Rud87], probability theory [Coh80, Rud87], statistics [CB90, Tri02] and order theory [DP90].

Basic Notations

Given two sets X and Y, we write $X + Y$ for the *disjoint union* of X and Y and $X \times Y$ for the *Cartesian product*, where the projection maps are denoted proj_1 and proj_2, respectively. The product of an indexed family $\{X_i \mid i \in \mathcal{I}\}$ of sets is denoted $\prod_{i \in \mathcal{I}} X_i$, with projection maps proj_i. When $\mathcal{I}$ is clear from context, we write $\vec{x}$ for a typical member of $\prod_{i \in \mathcal{I}} X_i$ and x_i for $\mathsf{proj}_i(\vec{x})$.

Given an equivalence relation $\equiv$ on X, the *quotient* of X under $\equiv$ (written $X/\equiv$) is the partition of X induced by $\equiv$. The *coset* $[x]$ in this quotient is the equivalence class containing x.

The set of all *partial functions* from X to Y is denoted $X \rightharpoonup Y$. The symbol $\emptyset$ denotes the empty function (as well as the empty set). For each $f \in X \rightharpoonup Y$, we write $\mathsf{Dom}(f)$, $\mathsf{Cod}(f)$ and $\mathsf{Range}(f)$ for the *domain, codomain and range* of f, respectively. If $x \notin \mathsf{Dom}(f)$, we sometimes write $f(x) = \bot$. We say that f is *total* if $X = \mathsf{Dom}(f)$ and we use $X \to Y$ to denote the set of all total functions from X to Y.

Metric Spaces

We encounter many times in this thesis the notion of "limits". They come in two flavors: (i) limit of a sequence of points in some metric space and (ii) limit of an increasing sequence in a partially ordered set. We now recall the former, while the latter is treated in Section 2.

Let $\mathbb{P}$ denote the set of non-negative real numbers. A *metric space* is a pair $\langle X, \mathsf{dist} \rangle$ where X is a set and the function $\mathsf{dist} : X \times X \to \mathbb{P}$ satisfies the following: for all $x, y \in X$,

(1) *identity*: $\mathsf{dist}(x, y) = 0$ if and only if $x = y$;

(2) *symmetry*: $\mathsf{dist}(x, y) = \mathsf{dist}(y, x)$; and

(3) *triangle inequality*: $\mathsf{dist}(x, z) \leq \mathsf{dist}(x, y) + \mathsf{dist}(y, z)$.

We give two familiar examples of metric spaces.

Example 2.0.1. The n-dimensional space $\mathbb{R}^n$ ($n \in \mathbb{N}$) together with the Euclidean distance function:

$$\mathsf{dist}(\vec{x}, \vec{y}) := \sqrt{\sum_{i=0}^{n} (x_i - y_i)^2}.$$

Example 2.0.2. The infinite dimensional space $[l, u]^\omega$ ($l, u \in \mathbb{R}$ with $l < u$) together with the distance function:

$$\mathsf{dist}(\vec{x}, \vec{y}) := \sup_{i \in \mathbb{N}} |x_i - y_i|.$$

Given an arbitrary metric space $\langle X, \mathsf{dist} \rangle$, we define the usual notion of an (open) *ε-ball* around a point x:

$$\mathbf{B}_\varepsilon(x) := \{y \in X \mid \mathsf{dist}(x, y) < \varepsilon\}.$$

A sequence of points $\{x_i \mid i \in \mathbb{N}\}$ in X *converges* to a *limit* $x \in X$ if, for every $\varepsilon > 0$, there is $N_\varepsilon \in \mathbb{N}$ such that $x_i \in \mathbf{B}_\varepsilon(x)$ for all $i \geq N_\varepsilon$. Equivalently, we may require $\lim_{i \to \infty} \mathsf{dist}(x, x_i) = 0$. It is trivial to check that limits must be unique and that all subsequences converge to the same limit.

The following is a special case of the famous Bolzano-Weierstraß Theorem.

Theorem 2.0.1. *Every bounded infinite sequence over $\mathbb{R}$ has a convergent subsequence.*

Probability Spaces

Let Ω be a set. A collection $\mathcal{F}$ of subsets of Ω is said to be a *σ-field* over Ω if $\mathcal{F}$ satisfies the following properties:

(1) $\Omega \in \mathcal{F}$;

(2) if $X \in \mathcal{F}$, then $\Omega \setminus X$ is also in $\mathcal{F}$ (closure under complement); and

(3) if $\{X_i \mid i \in \mathbb{N}\} \subseteq \mathcal{F}$, then $\bigcup_{i \in \mathbb{N}} X_i$ is also in $\mathcal{F}$ (closure under countable union).

We have the following familiar theorem about σ-fields.

Theorem 2.0.2. *Let $\mathcal{S}$ be any family of subsets of Ω. There exists a smallest σ-field $\mathcal{F}$ over Ω such that $\mathcal{S} \subseteq \mathcal{F}$. In that case, we say that $\mathcal{F}$ is* generated *by $\mathcal{S}$.*

A *probability measure* on a σ-field $\mathcal{F}$ is a countably additive function $\mathbf{m} : \mathcal{F} \to [0,1]$ such that $\mathbf{m}(\Omega) = 1$. *Countable additivity* says, given any disjoint family $\{X_i \mid i \in \mathbb{N}\} \subseteq \mathcal{F}$, it must be the case that

$$\mathbf{m}(\bigcup_{i\in\mathbb{N}} X_i) = \sum_{i\in\mathbb{N}} \mathbf{m}(X_i).$$

If $\mathbf{m}$ is a probability measure, the triple $\langle \Omega, \mathcal{F}, \mathbf{m}\rangle$ is said to form a *probability space*. The set Ω is called the *sample space* and members of $\mathcal{F}$ are called *events*.

Example 2.0.3. The powerset of Ω, $\mathcal{P}(\Omega)$, is a σ-field over Ω. Consider a function $\mu : \Omega \to [0,1]$ such that $\sum_{s\in\Omega} \mu(s) = 1$. Then μ induces a function $\mathbf{m} : \mathcal{P}(\Omega) \to [0,1]$ as follows:

$$\mathbf{m}(X) := \sum_{s\in X} \mu(s).$$

It is easy to check that $\mathbf{m}$ is countably additive, hence a probability measure on $\mathcal{P}(\Omega)$.

Such a function μ is often called a *discrete probability distribution* over the set Ω. The *support* of μ is defined to be the set $\mathsf{Supp}(\mu) := \{s \in \Omega \mid \mu(s) \neq 0\}$. Note that the support of a discrete probability distribution is a countable set. If $\mathsf{Supp}(\mu)$ is a singleton $\{s\}$, then μ is called a *Dirac distribution* and is often written as $\mathsf{Dirac}(s)$. If Ω is finite, then the distribution $\mathsf{Unif}(\Omega)$ assigns probability $\frac{1}{|\Omega|}$ to every s in Ω. The set of all discrete probability distributions over Ω is denoted by $\mathsf{Disc}(\Omega)$.

Similarly, we define a *sub-probability measure* to be a countably additive function $\mathbf{m} : \mathcal{F} \to [0,1]$ such that $\mathbf{m}(\Omega) \leq 1$. Thus a *discrete sub-distribution* is a function $\mu : \Omega \to [0,1]$ such that $\sum_{s\in\Omega} \mu(s) \leq 1$. The set of all such sub-distributions is denoted $\mathsf{SubDisc}(\Omega)$.

Example 2.0.4. Let Ω be the two element set $\{0,1\}$ and let μ be a discrete probability distribution over Ω. Write p for $\mu(1)$. This describes a *Bernoulli distribution* with parameter p. The two possible outcomes 1 and 0 are often referred to as *success* and *failure*, respectively.

If μ is a discrete distribution on a set X and Y is a superset of X, we shall freely regard μ as a discrete distribution on Y, where $\mu(y) := 0$ for all $y \in Y \setminus X$. Similarly for sub-distributions.

Let $\{X_i \mid i \in \mathcal{I}\}$ be an indexed family of sets and suppose we have $\{\mu_i \mid i \in \mathcal{I}\}$, where each μ_i is a discrete distribution on X_i. We form the *product distribution*,

denoted $\prod_{i\in\mathcal{I}} \mu_i$, as follows:

$$(\prod_{i\in\mathcal{I}} \mu_i)(\vec{s}) := \prod_{i\in\mathcal{I}} \mu_i(s_i).$$

This is easily shown to be a discrete distribution on $\prod_{i\in\mathcal{I}} X_i$. Conversely, given any distribution μ on $\prod_{i\in\mathcal{I}} X_i$ and $i \in \mathcal{I}$, one can form the *ith-projection* of μ, denoted $\mathsf{proj}_i(\mu)$, by:

$$\mathsf{proj}_i(\mu)(s) := \sum_{\vec{t}:t_i=s} \mu(\vec{t}).$$

We have the obvious identity: $\mathsf{proj}_i(\prod_{j\in\mathcal{I}} \mu_j) = \mu_i$.

Statistics

Let $\langle \Omega, \mathcal{F}, \mathbf{m}\rangle$ be a discrete probability space generated by the function $\mu : \Omega \to [0, 1]$. A *random variable* is a function $X : \Omega \to \mathbb{R}$. Intuitively, it is a rule that assigns a numerical value to each possible outcome of an experiment. Given $x \in \mathbb{R}$, let $[X = x]$ denote the event $\{s \in \Omega \mid X(s) = x\}$. The *probability mass function (pmf)* associated with X is defined by

$$p_X(x) := \mathbf{m}([X = x]) = \sum_{s\in[X=x]} \mu(s).$$

Often we write $\mathbf{P}[X = x]$ for $p_X(x)$. Similarly, we let $[X \geq x]$ denote the event $\{s \in \Omega \mid X(s) \geq x\}$ and write $\mathbf{P}[X \geq x]$ for $\sum_{s\in[X\geq x]} \mu(s)$.

The *expectation* (or *expected value*) of X, denoted $\mathbf{E}[X]$, is given by the sum

$$\mathbf{E}[X] := \sum_{x\in\mathsf{Range}(X)} x \ \ \mathbf{P}[X = x].$$

The *variance* of X, denoted $\mathbf{Var}[X]$, is defined as

$$\mathbf{Var}[X] := \mathbf{E}[(X - \mathbf{E}[X])^2] = \sum_{x\in\mathsf{Range}(X)} (x - \mathbf{E}[X])^2 \ \ \mathbf{P}[X = x].$$

Example 2.0.5. A *Bernoulli variable* is a random variable X with range $\{0, 1\}$. Intuitively, it classifies each outcome of an experiment as either success or failure. The value $\mathbf{P}[X = 1] = p$ is called the parameter of the Bernoulli variable. It is routine to derive $\mathbf{E}[X] = p$ and $\mathbf{Var}[X] = p(1 - p)$.

We have the following important inequality.

Theorem 2.0.3. *(Chebyshev's inequality). For every random variable X and $t > 0$,*

$$\mathbf{P}[|X - \mathbf{E}[X]| \geq t] \leq \frac{\mathbf{Var}[X]}{t^2}.$$

Next we consider hypothesis testing. This is a common method of *statistical inference*, which refers broadly to the practice of estimating characteristics of an entire population based on evidence produced by a sample drawn from that population. The starting point is a pair of complementary hypotheses: the *null* hypothesis and the *alternative* hypothesis. These are complementary statements about the probability distribution in question. A *hypothesis test* is a rule that specifies which sample values lead to the decision that the null hypothesis is accepted (thus the alternative hypothesis is rejected). This subset of the sample space is called the *acceptance region*, while its complement is called the *rejection region*. We say that a *false negative* (or *false rejection, type I*) error is committed if the null hypothesis is true but the test procedure concludes otherwise. Dually, a *false positive* (or *false acceptance*, *type II*) error is committed if the null hypothesis is false but is accepted by the test procedure. A test is said to be of *level* α ($\alpha \in [0,1]$) if the probability of committing a type I error is at most α.

Partial Orders

A *partially ordered set* (or *poset*) is a set P endowed with a binary relation $\leq$, which is reflexive, (weakly) antisymmetric and transitive. Given a subset $X \subseteq P$, we write $\bigvee X$ for the least upperbound of X, if it exists.

A non-empty subset D of P is *directed* if every finite subset D' of D has an upperbound in D. The least upperbound of a directed set (if it exists) is often called a *directed limit*. The poset P forms a *complete partial order (CPO)* if it has a bottom element $\bot$ and all directed limits. A function $f : P \to Q$ between CPOs P and Q is *monotone* if, for all $p, p' \in P$, $p \leq p'$ implies $f(p) \leq f(p')$. We say that f is *continuous* if it is monotone and, for every directed $D \subseteq P$, we have $f(\bigvee D) = \bigvee f(D)$. (Note that, given any monotone f, the image $f(D)$ of a directed set D is again directed.)

An increasing sequence of elements $p_0 \leq p_1 \leq p_2 \leq \ldots$ in P is called a *chain*. Chains are typical examples of directed sets and we write $\lim C$ for the least upperbound of a chain C. In fact, any directed limit can be converted to the limit of a chain with the same cardinality.

Theorem 2.0.4. *A poset P with $\bot$ is a CPO if and only $\lim C$ exists for every non-empty chain C.*

Finally, an element $c \in P$ is *compact* if, for every directed set D such that $c \leq \bigvee D$, there exists $p \in D$ with $c \leq p$. A CPO P is said to be *algebraic* if for all p, the set $\{c \mid c \leq p \text{ and } c \text{ compact}\}$ is directed and p is in fact the limit of this set.

Example 2.0.6. Let $X^{<\omega}$ (resp., X^{ω}) denote the set of finite (resp., infinite) sequences over a set X. Then the union of these two sets, denoted $X^{\leq\omega}$, forms an algebraic CPO under the prefix ordering $\sqsubseteq$. The compact elements are precisely the finite sequences.

Example 2.0.7. Consider $X \rightharpoonup Y$, the set of partial functions from X to Y. We define the *information ordering* on $X \rightharpoonup Y$ as follows: $f \subseteq g$ if and only if (i) $\mathsf{Dom}(f) \subseteq \mathsf{Dom}(g)$ and (ii) for all $x \in \mathsf{Dom}(f)$, $f(x) = g(x)$. In other words, the graph of f is a subset of the graph of g, hence the relation is also called the *subset ordering*. This gives rise to an algebraic CPO whose compact elements are partial functions with finite domain.

Infinite Sequences over $[0,1]$

We define a *flat* ordering on $[0,1]^\omega$ as follows: $\sigma \leq_\flat \sigma'$ if and only if for all $i \in \mathbb{N}$, $\sigma_i \neq 0$ implies $\sigma_i = \sigma'_i$. This ordering is very much analogous to the subset ordering in Example 2.0.7, since infinite sequences over $[0,1]$ can be viewed as functions from $\mathbb{N}$ to $[0,1]$ and we can interpret $\sigma_i = 0$ as "σ undefined at i". Given an arbitrary directed limit in this poset, we can always convert it to the limit of an ω-chain. This is a strengthening of Theorem 2.0.4 for the special case of $[0,1]^\omega$.

Lemma 2.0.5. *Let $\mathcal{D}$ be an arbitrary (not necessarily countable) directed subset of $[0,1]^\omega$. There is an ω-chain $\{\sigma_0, \sigma_1, \ldots\} \subseteq \mathcal{D}$ such that $\lim_{k\in\mathbb{N}} \sigma_k = \bigvee \mathcal{D}$.*

Proof. First we construct a sequence $\sigma'_0, \sigma'_1, \ldots$ as follows: for each $i \in \mathbb{N}$, choose $\sigma'_i \in \mathcal{D}$ such that $\sigma'_i(i) = (\bigvee \mathcal{D})(i)$. This is possible due to the definition of $\leq_\flat$. Then

- set σ_0 to be σ'_0;
- for $i+1$, set σ_{i+1} to be any upperbound of $\{\sigma_0, \ldots, \sigma_i, \sigma'_{i+1}\}$ in $\mathcal{D}$.

Since $\mathcal{D}$ is directed, this ω-chain is well-defined. One can easily check that its limit in fact equals the least upperbound of $\mathcal{D}$. □

Lemma 2.0.5 is used to prove Theorem 2.0.6 about infinite sums. Let $\mathcal{I}$ be an arbitrary index set and let $\{\{c_{i,j}\}_{j\in\mathbb{N}} \mid i \in \mathcal{I}\}$ be a set of ω-sequences over $[0,1]$. Assuming the infinite sums converge, it is true in general that

$$\bigvee_{i\in\mathcal{I}} \sum_{j\in\mathbb{N}} c_{i,j} \leq \sum_{j\in\mathbb{N}} \bigvee_{i\in\mathcal{I}} c_{i,j}.$$

We claim that equality holds under the assumption that $\{\{c_{i,j}\}_{j\in\mathbb{N}} \mid i \in \mathcal{I}\}$ is directed with respect to $\leq_\flat$. This can be seen as a special form of the well-known Monotone Convergence Theorem.

Theorem 2.0.6. *Assume that $\{\{c_{i,j} \mid j \in \mathbb{N}\} \mid i \in \mathcal{I}\}$ is a directed subset of $[0,1]^\omega$ and for all i, $\sum_{j\in\mathbb{N}} c_{i,j}$ converges to a limit in $[0,1]$. Then the sum $\sum_{j\in\mathbb{N}} \bigvee_{i\in\mathcal{I}} c_{i,j}$ converges and*

$$\bigvee_{i\in\mathcal{I}} \sum_{j\in\mathbb{N}} c_{i,j} = \sum_{j\in\mathbb{N}} \bigvee_{i\in\mathcal{I}} c_{i,j}.$$

Proof. For convergence, we show that all finite partial sums $\sum_{0\leq j\leq N}\bigvee_{i\in\mathcal{I}} c_{i,j}$ are bounded above by $\bigvee_{i\in\mathcal{I}}\sum_{j\in\mathbb{N}} c_{i,j}$. (In that case equality must also hold.)

By Lemma 2.0.5, we can choose an ω-chain $\{D_k \mid k\in\mathbb{N}\}\subseteq\{\{c_{i,j}\mid j\in\mathbb{N}\}\mid i\in\mathcal{I}\}$ such that $\bigvee_{i\in\mathcal{I}}\{c_{i,j}\mid j\in\mathbb{N}\}=\lim_{k\in\mathbb{N}} D_k$. Notice $(\lim_{k\in\mathbb{N}} D_k)(j)=\lim_{k\in\mathbb{N}} D_k(j)$, thus

$$\begin{aligned}\sum_{0\leq j\leq N}\bigvee_{i\in\mathcal{I}} c_{i,j} &= \sum_{0\leq j\leq N}(\lim_{k\in\mathbb{N}} D_k)(j) = \sum_{0\leq j\leq N}\lim_{k\in\mathbb{N}} D_k(j)\\ &= \lim_{k\in\mathbb{N}}\sum_{0\leq j\leq N} D_k(j) = \bigvee_{k\in\mathbb{N}}\sum_{0\leq j\leq N} D_k(j).\end{aligned}$$

This is below $\bigvee_{i\in\mathcal{I}}\sum_{j\in\mathbb{N}} c_{i,j}$, because each D_k equals $\{c_{i,j}\mid j\in\mathbb{N}\}$ for some $i\in\mathcal{I}$. □

An obvious corollary of Theorem 2.0.6 concerns the set of discrete probabilistic sub-distributions.

Corollary 2.0.7. *Let S be a countable set. The set $\mathsf{SubDisc}(S)$ of discrete probabilistic sub-distributions over S is a CPO with respect to the flat ordering.*

Proof. Via an enumeration of S, we view $\mathsf{SubDisc}(S)$ as a subset of $[0,1]^\omega$. Clearly the everywhere-0 distribution is a bottom element. Given any directed subset Δ, we apply Theorem 2.0.6 to

$$\{\{D(j)\mid j\in\mathbb{N}\}\mid D\in\Delta\}$$

and conclude that the (pointwise) join of Δ is also a sub-distribution. □

3

Probabilistic Automata

Throughout Part I of this thesis, a probabilistic process is a *(simple) probabilistic automaton* as introduced by Segala and Lynch [Seg95b, SL95]. This extends the usual nondeterministic automata model by allowing probabilistic information at the target of each transition. More precisely, every transition leads to a probability distribution over possible next states, rather than a single state. This section presents relevant definitions and basic results concerning semantic objects associated with probabilistic automata. In Part II, we will consider variations of the probabilistic automata model, for example, with input/output distinction and bundle transitions.

For simplicity, we consider systems with no internal or hidden actions. All external actions are taken from a countable set Act, which has a fixed enumeration $\{b_i \mid i \in \mathbb{N}\}$ throughout this thesis. Given $l \in \mathbb{N}$, we write Act_l for the list $b_0, \ldots, b_{l-1}$. The set of finite (resp. infinite) traces is denoted $\mathsf{Act}^{<\omega}$ (resp. Act^{ω}), while the set of all traces is $\mathsf{Act}^{\leq\omega}$. Also, we write ϵ for the empty trace.

Definition 3.0.1. A *probabilistic automaton (PA)* is a triple $\mathcal{A} = (S, s^0, \Delta)$ where

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

We write $s \xrightarrow{a} \mu$ for $(s, a, \mu) \in \Delta$. Also, we write $s \overset{a,\mu}{\rightsquigarrow} t$ whenever $s \xrightarrow{a} \mu$ and $\mu(t) > 0$. To avoid confusion, we sometimes refer to the components of $\mathcal{A}$ as $S_{\mathcal{A}}$, $s^0_{\mathcal{A}}$ and $\Delta_{\mathcal{A}}$.

Intuitively, we can view target distributions in the transition relation Δ as a form of probabilistic branching; that is, we think of $s \overset{a,\mu}{\rightsquigarrow} t$ as a nondeterministic transition $s \xrightarrow{a} \mu$ followed by a probabilistic transition $\mu \xrightarrow{\mu(t)} t$. In this way, we obtain an informal notion of the *underlying nondeterministic automaton* of $\mathcal{A}$, where we "forget" probabilistic information (i.e., $\mu(t)$) at each probabilistic transition. Thus inspired, we define paths in a PA $\mathcal{A}$ as follows.

Definition 3.0.2. A *path* π of $\mathcal{A}$ is a (finite or infinite) sequence of the form $s_0 a_1 \mu_1 s_1 a_2 \mu_2 s_2 \ldots$ such that:

- each s_i (resp., a_i, μ_i) denotes a state (resp., action, distribution over states);
- s_0 is the initial state[1];
- if π is finite, then it ends with a state;
- $s_i \overset{a_{i+1},\mu_{i+1}}{\rightsquigarrow} s_{i+1}$, for each non-final i.

The *length* of a finite path π, denoted $|\pi|$, is the number of action symbols occurring in it.

The set of all paths (finite and infinite) of $\mathcal{A}$ is denoted $\mathsf{Path}(\mathcal{A})$, while the set of finite paths is denoted $\mathsf{Path}^{<\omega}(\mathcal{A})$. We write $\mathsf{Path}^{\leq k}(\mathcal{A})$ for the set of paths with length at most k. The last state of a finite path π is written $\mathsf{last}(\pi)$. The *trace* of π, denoted by $\mathsf{tr}(\pi)$, is defined to be the sequence of actions appearing along π: $a_1 a_2 a_3 \ldots$. Given $F \subseteq \mathsf{Path}^{<\omega}\,\mathcal{A}$ and $a \in \mathsf{Act}$, we write $\mathsf{Succ}(F, a)$ for the set of paths π' of the form $\pi a \mu s$ with $\pi \in F$. Similarly for $\mathsf{Succ}(F, \beta)$ with $\beta \in \mathsf{Act}^{<\omega}$.

As in the case of nondeterministic automata, we are interested in certain finiteness properties in the branching structure.

Definition 3.0.3. A PA $\mathcal{A}$ is *finitely (resp. countably) branching* if, for each state s, the set $\{\langle a, \mu\rangle \mid s \xrightarrow{a} \mu\}$ is finite (resp. countable). It is *image finite* if for each state s and action a, the set $\{\mu \mid s \xrightarrow{a} \mu\}$ is finite.

Thus, each state in a finitely branching PA has finitely many outgoing transitions, while a state in an image finite PA may have infinitely many. In both cases, the set $\{t \mid s \overset{a,\mu}{\rightsquigarrow} t \text{ for some } a, \mu\}$ maybe infinite, since a target distribution μ may have infinite support. As a result, given a finite trace $\beta \in \mathsf{Act}^{<\omega}$, a finitely branching (or image finite) PA may have infinitely many paths with trace β. This is different from the case of nondeterministic automata.

Since Act is countable by assumption, image finiteness is a stronger condition than countable branching. Throughout Part I, we take countable branching as our basic assumption. However, image finiteness becomes necessary in some place, when we wish to prove convergence properties.

Note that every transition in a PA leads to a discrete distribution on states, which has a countable support. Therefore, the set of finite paths of a countably branching PA is countable. We often take advantage of this fact by imposing an enumeration.

[1] In other terminology, paths may start from non-initial states.

3.1 Adversaries and Probabilistic Executions

We now turn to behaviors of PAs. In the non-probabilistic case, an execution (or path) is obtained by resolving all nondeterministic choices in a deterministic fashion. For a PA, we resolve nondeterministic choices by means of an *adversary* (sometimes called a *scheduler*). Given any finite history leading to the current state, an adversary returns a discrete sub-distribution over the set of available next transitions. Therefore, our adversaries are (i) randomized, (ii) history-dependent, and (iii) partial, in the sense that they may choose to halt the execution at any time.

Definition 3.1.1. A *(randomized, history-dependent and partial) adversary* E of $\mathcal{A}$ is a function

$$E : \mathsf{Path}^{<\omega}(\mathcal{A}) \to \mathsf{SubDisc}(\mathsf{Act} \times \mathsf{Disc}(S_{\mathcal{A}}))$$

satisfying, for each finite path π, $E(\pi)(a, \mu) > 0$ implies $\mathsf{last}(\pi) \xrightarrow{a} \mu$.

We write $\mathsf{Adv}(\mathcal{A})$ for the set of all adversaries of $\mathcal{A}$. Intuitively, an adversary E tosses a coin to choose the next transition at every step of the computation of $\mathcal{A}$. Thus E induces a purely probabilistic "computation tree". This idea is captured by the notion of a probabilistic execution.

Definition 3.1.2. Let E be an adversary of $\mathcal{A}$. The *probabilistic execution* induced by E, denoted $\mathbf{Q}_E$, is the function from $\mathsf{Path}^{<\omega}(\mathcal{A})$ to $[0, 1]$ defined recursively by

$$\begin{aligned} \mathbf{Q}_E(s_0) &= 1, \\ \mathbf{Q}_E(\pi a \mu s) &= \mathbf{Q}_E(\pi) \cdot E(\pi)(a, \mu) \cdot \mu(s). \end{aligned}$$

The set of all probabilistic executions of $\mathcal{A}$ is written as $\mathsf{PExec}(\mathcal{A})$. Essentially, the function $\mathbf{Q}_E$ assigns probabilities to finite paths according to decisions made by the adversary E. We shall interpret "$\mathbf{Q}_E(\pi) = p$" as: under the control of adversary E, the automaton $\mathcal{A}$ follows path π with probability p. Notice, it need *not* be the case that $\mathcal{A}$ halts after π. Moreover, if $\pi \sqsubseteq \pi'$, then the event "$\mathcal{A}$ follows π'" implies the event "$\mathcal{A}$ follows π". Therefore $\mathbf{Q}_E$ is *not* a discrete distribution on the set of finite paths. However, $\mathbf{Q}_E$ does induce a probability space over the sample space $\mathsf{Path}(\mathcal{A})$ as follows.

Definition 3.1.3. Let $\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$ be given. The *cone* generated by π is the following set of paths: $\mathbf{C}_\pi := \{\pi' \in \mathsf{Path}(\mathcal{A}) \mid \pi \sqsubseteq \pi'\}$.

Let $\Omega_{\mathcal{A}} := \mathsf{Path}(\mathcal{A})$ be the sample space and let $\mathcal{F}_{\mathcal{A}}$ be the smallest σ-field generated by the collection $\{\mathbf{C}_\pi \mid \pi \in \mathsf{Path}^{<\omega}(\mathcal{A})\}$. The following theorem states that $\mathbf{Q}_E$ induces a unique probability measure on $\mathcal{F}_A$ [Seg95b].

Theorem 3.1.1. *Let E be an adversary of $\mathcal{A}$. There exists a unique probability measure $\mathbf{m}_E$ on $\mathcal{F}_{\mathcal{A}}$ such that $\mathbf{m}_E(\mathbf{C}_\pi) = \mathbf{Q}_E(\pi)$ for all $\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$.*

The measure $\mathbf{m}_E$ of the preceding theorem gives rise to a probability space $(\Omega_{\mathcal{A}}, \mathcal{F}_{\mathcal{A}}, \mathbf{m}_E)$. In the literature, many authors define probabilistic executions to be such probability spaces. For our development, it is more natural to reason with the function $\mathbf{Q}_E$, rather than the induced probability space. By virtue of Theorem 3.1.1, these two definitions are equivalent.

3.2 Trace Distributions

External behaviors of a PA $\mathcal{A}$ are obtained by removing the non-visible elements from probabilistic executions. Since we do not deal with internal actions, we remove states and distributions of states. This yields a trace distribution of $\mathcal{A}$, which assigns probabilities to certain sets of traces.

We define trace distributions via a lifting of the trace operator $\mathsf{tr} : \mathsf{Path}^{<\omega}(\mathcal{A}) \to \mathsf{Act}^{<\omega}$.

Definition 3.2.1. Let an adversary E of $\mathcal{A}$ be given and consider the probabilistic execution $\mathbf{Q}_E : \mathsf{Path}^{<\omega}(\mathcal{A}) \to [0,1]$. The *trace distribution* induced by E is the function $\mathsf{tr}(\mathbf{Q}_E) : \mathsf{Act}^{<\omega} \to [0,1]$ given by

$$\mathsf{tr}(\mathbf{Q}_E)(\beta) := \sum_{\pi \in \mathsf{tr}^{-1}(\beta)} \mathbf{Q}_E(\pi).$$

Notice, because we have no internal actions, the cones $\mathbf{C}_{\pi_1}$ and $\mathbf{C}_{\pi_2}$ are disjoint whenever π_1 and π_2 are distinct paths in $\mathsf{tr}^{-1}(\beta)$. Therefore, we have

$$\sum_{\pi \in \mathsf{tr}^{-1}(\beta)} \mathbf{Q}_E(\pi) = \sum_{\pi \in \mathsf{tr}^{-1}(\beta)} \mathbf{m}_E(\mathbf{C}_\pi) = \mathbf{m}_E\Big(\bigcup_{\pi \in \mathsf{tr}^{-1}(\beta)} \mathbf{C}_\pi \Big) \in [0,1].$$

This ensures that $\mathsf{tr}(\mathbf{Q}_E)$ is well-defined. We usually write $\mathbf{D}_E$ for $\mathsf{tr}(\mathbf{Q}_E)$ and, when it is desirable to leave the adversary E implicit, we use variables D, K, etc. The set of trace distributions of $\mathcal{A}$ is denoted by $\mathsf{TrDist}(\mathcal{A})$, and we define trace distribution inclusion as follows: $\mathcal{A} \leq_{\mathsf{td}} \mathcal{B}$ if and only if $\mathsf{TrDist}(\mathcal{A}) \subseteq \mathsf{TrDist}(\mathcal{B})$. Trace distribution equivalence is thus: $\mathcal{A} \equiv_{\mathsf{td}} \mathcal{B}$ if and only if $\mathsf{TrDist}(\mathcal{A}) = \mathsf{TrDist}(\mathcal{B})$.

Similar to the case of probability executions, each $\mathbf{D}_E$ induces a probability measure on the sample space $\Omega := \mathsf{Act}^{\leq\omega}$. Here the σ-field $\mathcal{F}$ is generated by the collection $\{\mathbf{C}_\beta \mid \beta \in \mathsf{Act}^{<\omega}\}$, where $\mathbf{C}_\beta := \{\beta' \in \Omega \mid \beta \sqsubseteq \beta'\}$.

Theorem 3.2.1. *Let E be an adversary of $\mathcal{A}$. There exists a unique probability measure $\mathbf{m}^E$ on $\mathcal{F}$ such that $\mathbf{m}^E(\mathbf{C}_\beta) = \mathbf{D}_E(\beta)$.*

Again, $\mathbf{m}^E$ gives rise to a probability space $\langle \Omega, \mathcal{F}, \mathbf{m}^E \rangle$, which is elsewhere called the trace distribution induced by E. We refer to [Seg95b] for these alternative definitions and the proofs of Theorems 3.1.1 and 3.2.1.

In general, $\mathbf{m}^E$ is not a discrete measure. However, for every $k \in \mathbb{N}$, $\mathbf{m}^E$ induces a discrete probability distribution on $\mathsf{Act}^{\leq k}$. We now describe how this is done. First, note that every singleton set $\{\beta\}$, with $\beta \in \mathsf{Act}^{<\omega}$, is a member of $\mathcal{F}$. This is because $\{\beta\}$ can be rewritten as $\mathbf{C}_\beta \setminus \bigcup_{a \in \mathsf{Act}} \mathbf{C}_{\beta a}$, which is measurable since Act is countable. Now, given $\beta \in \mathsf{Act}^{\leq k}$, we define $\mathbf{P}_{E,k}[\beta]$ to be:

- $\mathbf{m}^E(\mathbf{C}_\beta)$, if the length of β is exactly k;
- $\mathbf{m}^E(\{\beta\})$, otherwise.

The following proposition confirms that $\mathbf{P}_{E,k}$ is in fact a discrete probability distribution on $\mathsf{Act}^{\leq k}$.

Proposition 3.2.2. *For every $k \in \mathbb{N}$ and every adversary E, the function $\mathbf{P}_{E,k} : \mathsf{Act}^{\leq k} \to [0,1]$ is a discrete probability distribution over $\mathsf{Act}^{\leq k}$.*

Proof. We need to verify that $\sum_{\beta \in \mathsf{Act}^{\leq k}} \mathbf{P}_{E,k}[\beta] = 1$.

By Theorem 3.2.1, we know that $\mathbf{m}^E(\mathbf{C}_\beta) = \mathbf{D}_E(\beta)$ for every $\beta \in \mathsf{Act}^{<\omega}$. In case $k = 0$, it is sufficient to note that $\mathbf{P}_{E,k}[\epsilon] = \mathbf{m}^E(\mathbf{C}_\epsilon) = \mathbf{D}_E(\epsilon) = 1$. Assume $k \geq 1$. The following holds for every $\beta \in \mathsf{Act}^{<k}$:

$$\mathbf{P}_{E,k}[\beta] = \mathbf{m}^E(\{\beta\}) = \mathbf{m}^E(\mathbf{C}_\beta) - \sum_{a \in \mathsf{Act}} \mathbf{m}^E(\mathbf{C}_{\beta a}) = \mathbf{D}_E(\beta) - \sum_{a \in \mathsf{Act}} \mathbf{D}_E(\beta a).$$

On the other hand, we have:

$$\sum_{\beta \in \mathsf{Act}^{\leq k}} \mathbf{P}_{E,k}[\beta] = \sum_{i=0}^{k} \sum_{\beta \in \mathsf{Act}^i} \mathbf{P}_{E,k}[\beta] = (\sum_{i=0}^{k-1} \sum_{\beta \in \mathsf{Act}^i} \mathbf{P}_{E,k}[\beta]) + \sum_{\beta \in \mathsf{Act}^k} \mathbf{P}_{E,k}[\beta].$$

According to the length of β, we make appropriate substitutions for $\mathbf{P}_{E,k}[\beta]$ and obtain:

$$\begin{aligned}
\sum_{\beta \in \mathsf{Act}^{\leq k}} \mathbf{P}_{E,k}[\beta] &= (\sum_{i=0}^{k-1} \sum_{\beta \in \mathsf{Act}^i} (\mathbf{D}_E(\beta) - \sum_{a \in \mathsf{Act}} \mathbf{D}_E(\beta a))) + \sum_{\beta \in \mathsf{Act}^k} \mathbf{D}_E(\beta) \\
&= (\sum_{i=0}^{k-1} (\sum_{\beta \in \mathsf{Act}^i} \mathbf{D}_E(\beta) - \sum_{\beta \in \mathsf{Act}^i} \sum_{a \in \mathsf{Act}} \mathbf{D}_E(\beta a))) + \sum_{\beta \in \mathsf{Act}^k} \mathbf{D}_E(\beta) \\
&= (\sum_{i=0}^{k-1} (\sum_{\beta \in \mathsf{Act}^i} \mathbf{D}_E(\beta) - \sum_{\beta \in \mathsf{Act}^{i+1}} \mathbf{D}_E(\beta))) + \sum_{\beta \in \mathsf{Act}^k} \mathbf{D}_E(\beta) \\
&= \sum_{\beta \in \mathsf{Act}^0} \mathbf{D}_E(\beta) = \mathbf{D}_E(\epsilon) = 1
\end{aligned}$$

□

Observe the slight difference in meaning between $\mathbf{D}_E(\beta)$ and $\mathbf{P}_{E,k}[\beta]$ for $\beta \in \mathsf{Act}^{<k}$. The former is the probability that "$\mathcal{A}$ performs the trace β", without specifying what happens after β. The latter is the probability that "$\mathcal{A}$ performs *exactly* the trace β;" that is, $\mathcal{A}$ performs β and halts immediately afterwards. Moreover, given traces $\beta_0 \neq \beta_1$, the two events "$\mathcal{A}$ performs exactly β_0" and "$\mathcal{A}$ performs exactly β_1" are mutually exclusive. This holds even when β_0 is a prefix of β_1.

Since we restrict our attention to traces with length at most k, the function $\mathbf{P}_{E,k}$ gives a full description of the behavior of $\mathcal{A}$ while under the control of adversary E. Notice that $\mathbf{P}_{E,k}$ depends only on $\mathbf{D}_E$, therefore we often leave E implicit and write $\mathbf{P}_{D,k}$ when $D = \mathbf{D}_E$ for some E.

3.3 Finite Adversaries

Let E be an adversary of a PA $\mathcal{A}$. Given a finite path π, we say that π is *E-reachable* if $\mathbf{Q}_E(\pi) \neq 0$. Recall that adversaries may choose to halt an execution at any point. This is reflected by the fact that $E(\pi)$ is a sub-distribution on the set of possible next transitions, so the probability of E halting after π is

$$1 - \sum_{\langle a,\, \mu\rangle \in \mathsf{Supp}(E(\pi))} E(\pi)\langle a,\, \mu\rangle.$$

If $E(\pi)$ has empty support, then we say E *halts* after path π. In that case, $\mathbf{Q}_E(\pi') = 0$ for any proper extension π' of π. We say that E has *depth* (at most) k if E halts after every path of length k. This implies that every E-reachable path has length at most k.

The notion of depth gives a bound on how far an adversary follows each path. We also wish to talk about the degree of branching in an adversary. A typical approach is to give a bound on the cardinality of $\mathsf{Supp}(E(\pi))$ for all π. Here we propose a different definition: E has *breadth* (at most) l if, for all paths π and pairs $\langle a,\, \mu\rangle$ in $\mathsf{Act} \times \mathsf{Disc}(S_{\mathcal{A}})$, $E(\pi)(a, \mu) > 0$ implies $a \in \mathsf{Act}_l$. Given such E, an easy inductive argument shows that $\mathsf{tr}(\pi) \in (\mathsf{Act}_l)^{<\omega}$ for every E-reachable finite path π.

For all $k, l \in \mathbb{N}$, let $\mathsf{Adv}(\mathcal{A}, k, l)$ denote the set of adversaries of depth k and breadth l. We say that E is a *finite* adversary if there exists $k, l \in \mathbb{N}$ such that $E \in \mathsf{Adv}(\mathcal{A}, k, l)$. In other words, E is finite if it has both finite depth and finite breadth. The following proposition follows immediately from the relevant definitions.

Proposition 3.3.1. *Let $E \in \mathsf{Adv}(\mathcal{A}, k, l)$ and $\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$ be given. If π is E-reachable then $\mathsf{tr}(\pi) \in (\mathsf{Act}_l)^{\leq k}$.*

Finite adversaries are extremely important in our development, because we rely on reduction of infinite behavior to its finite approximations. This idea will

become clear in Chapters 5 through 7. In the meantime, we make some simple observations.

Proposition 3.3.2. *If $\mathcal{A}$ is an image finite PA and E is an adversary of $\mathcal{A}$ with finite breadth, then $\mathsf{Supp}(E(\pi))$ is finite for every E-reachable π.*

Proof. Suppose π is an E-reachable path in $\mathcal{A}$. By finite breadth of E, there are only finitely many $a \in \mathsf{Act}$ such that E assigns non-zero probability to transitions labeled a. Let such a be given. By image finiteness, there are only finitely many a-transitions available at π. Therefore, $\mathsf{Supp}(E(\pi))$ must be finite. □

Proposition 3.3.3. *There exist image finite PA $\mathcal{A}$ and adversary E of $\mathcal{A}$ such that $\mathsf{Supp}(E(\pi))$ is finite for all π but E has infinite breadth.*

Proof. Consider a single-state automaton with countably many loops such that no two loops carry the same label. Let E be an adversary that always chooses (with probability 1) a transition carrying a fresh label . Then $\mathsf{Supp}(E(\pi))$ is a singleton for all π and yet E has infinite breadth. □

Propositions 3.3.2 and 3.3.3 tell us that the finite breadth condition is strictly stronger than the requirement that $\mathsf{Supp}(E(\pi))$ is finite for all E-reachable π. As we show in Chapters 5 and 6, the conjunction of finite breadth and depth captures the "correct" notion of finiteness, which gives rise to an algebraic CPO structure on $\mathsf{TrDist}(\mathcal{A})$ and allows us to prove metric convergence properties.

We extend the notion of finiteness to probabilistic executions: $\mathbf{Q}_E$ is finite if there is an E' such that E' is finite and $\mathbf{Q}_E = \mathbf{Q}_{E'}$. The set of probabilistic executions induced by adversaries from $\mathsf{Adv}(\mathcal{A}, k, l)$ is denoted $\mathsf{PExec}(\mathcal{A}, k, l)$.

Finite trace distributions are defined analogously: $\mathbf{D}_E$ is finite just in case there is a finite E' such that $\mathbf{D}_E = \mathbf{D}_{E'}$. The set of trace distributions induced by adversaries from $\mathsf{Adv}(\mathcal{A}, k, l)$ is denoted $\mathsf{TrDist}(\mathcal{A}, k, l)$ and we write $\mathcal{A} \leq^{k,l}_{\mathsf{td}} \mathcal{B}$ whenever $\mathsf{TrDist}(\mathcal{A}, k, l) \subseteq \mathsf{TrDist}(\mathcal{B}, k, l)$. Also, we use $\mathsf{Adv}(\mathcal{A}, k, -)$ to denote the set of all adversaries with depth k (and arbitrary breadth). The same convention applies also to $\mathsf{Adv}(\mathcal{A}, -, l)$, $\mathsf{PExec}(\mathcal{A}, k, -)$, etc.

We end this section with some observations regarding discrete probability distributions $\mathbf{P}_{E,k}$ with $E \in \mathsf{Adv}(\mathcal{A}, k, l)$. The first remark is a corollary of Proposition 3.2.2.

Corollary 3.3.4. *Let $k, l \in \mathbb{N}$ and $E \in \mathsf{Adv}(\mathcal{A}, k, l)$ be given. Let $\mathbf{P}_{E,k,l}$ denote the restriction of $\mathbf{P}_{E,k}$ to $(\mathsf{Act}_l)^{\leq k}$. Then $\mathbf{P}_{E,k,l}$ is a discrete probability distribution over $(\mathsf{Act}_l)^{\leq k}$.*

Proof. This follows from the definition of $\mathbf{P}_{E,k}$ and Propositions 3.2.2 and 3.3.1. □

Again, when we wish to leave the adversary E implicit, we write $\mathbf{P}_{D,k,l}$ for $\mathbf{P}_{E,k,l}$ where $D = \mathbf{D}_E$. Also, the subscript l is often omitted when it is clear from context.

The second remark is, if two trace distributions from $\mathsf{TrDist}(\mathcal{A}, k, l)$ induce the same discrete distribution on $(\mathsf{Act}_l)^{\leq k}$, then they must be identical.

Proposition 3.3.5. *The function* $\mathbf{P}_{-,k} : \mathsf{TrDist}(\mathcal{A}, k, l) \to \mathsf{Disc}((\mathsf{Act}_l)^{\leq k})$ *is one-to-one.*

Proof. We will give a left inverse of $\mathbf{P}_{-,k}$. Let $D \in \mathsf{TrDist}(\mathcal{A}, k, l)$ be given. Define a function $D' : (\mathsf{Act}_l)^{\leq k} \to [0, 1]$ as follows:

$$D'(\beta) = \sum_{\beta \sqsubseteq \beta'; \beta' \in (\mathsf{Act}_l)^{\leq k}} \mathbf{P}_{D,k}[\beta'].$$

With a (backwards) inductive argument on the length of $\beta \in (\mathsf{Act}_l)^{\leq k}$, it is easy to check that $D = D'$. □

Using essentially the same proof, we obtain the analogous result for trace distributions in $\mathsf{TrDist}(\mathcal{A}, k, -)$.

Proposition 3.3.6. *The function* $\mathbf{P}_{-,k} : \mathsf{TrDist}(\mathcal{A}, k, -) \to \mathsf{Disc}(\mathsf{Act}^{\leq k})$ *is one-to-one.*

4

Characterizing Probabilistic Executions

This chapter presents some useful results on probabilistic executions and trace distributions. First we give an explicit characterization for the set of probabilistic executions of a given PA $\mathcal{A}$. This characterization is then used to prove that the set of trace distributions of $\mathcal{A}$ is closed under convex combinations. Finally, we describe a method of constructing an adversary from an infinite sequence of adversaries.

4.1 Characterization Theorem

By definition, a probabilistic execution $\mathbf{Q}_E$ is a mapping from $\mathsf{Path}^{<\omega}(\mathcal{A})$ to $[0,1]$, induced by some adversary E of a PA $\mathcal{A}$. Hence we can view $\mathbf{Q}$ as an operator from the set of adversaries of $\mathcal{A}$ to the function space $\mathsf{Path}^{<\omega}(\mathcal{A}) \to [0,1]$. This section provides an explicit characterization of the range of $\mathbf{Q}$. In other words, given an arbitrary function $Q : \mathsf{Path}^{<\omega}(\mathcal{A}) \to [0,1]$, we determine whether $Q = \mathbf{Q}_E$ for some adversary E of $\mathcal{A}$.

Clearly, if Q is induced by some E, it must satisfy the following properties.

(1) $Q(s^0) = 1$ and, whenever π is a prefix of π', we have $Q(\pi) \geq Q(\pi')$ (i.e., Q is antitone with respect to the prefix ordering).

(2) Given π, a, μ, s_0, s_1 such that $\mathsf{last}(\pi) \xrightarrow{a} \mu$ and $s_0, s_1 \in \mathsf{Supp}(\mu)$, we have

$$\frac{Q(\pi a \mu s_0)}{\mu(s_0)} = \frac{Q(\pi a \mu s_1)}{\mu(s_1)}.$$

We call this property the *consistency* of Q.

(3) Given $\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$ with $Q(\pi) \neq 0$, let S_π denote the set of (a,μ) such that $\mathsf{last}(\pi) \xrightarrow{a} \mu$. For each $(a,\mu) \in S_\pi$, fix any $s_{a,\mu} \in \mathsf{Supp}(\mu)$. Then

$$\sum_{(a,\mu)\in S_\pi} \frac{Q(\pi a \mu s_{a,\mu})}{Q(\pi) \cdot \mu(s_{a,\mu})} \leq 1.$$

(Notice, if Q is consistent, the choice of $s_{a,\mu}$ does not affect the summand.)

To see that these conditions are not only necessary but also sufficient to characterize the set of probabilistic executions, we note the following. Condition (1) expresses that, if $\pi \sqsubseteq \pi'$, then the event "$\mathcal{A}$ follows π'" is included in the event "$\mathcal{A}$ follows π". Also, any probabilistic execution begins at the start state s^0 with probability 1. Condition (2) is more subtle. Recall that $\mathbf{Q}_E(\pi a \mu s) = \mathbf{Q}_E(\pi) \cdot E(\pi)(a,\mu) \cdot \mu(s)$. If $Q(\pi) > 0$, we can recover the value $E(\pi)(a,\mu)$ from Q by taking the quotient $\frac{Q(\pi a \mu s)}{Q(\pi)\cdot\mu(s)}$ for some state $s \in \mathsf{Supp}(\mu)$, provided any choice of s yields the same quotient. This is precisely Condition (2). Condition (3) then says the sum of $E(\pi)(a,\mu)$ over all possible transitions $\mathsf{last}(\pi) \xrightarrow{a} \mu$ must be under 1 (i.e., $E(\pi)$ is a discrete sub-distribution on S_π).

Given a function Q with these properties, we construct an adversary E_Q as follows: for π, a and μ, define $E_Q(\pi)(a,\mu)$ to be

- 0, in case $Q(\pi) = 0$ or $\mathsf{last}(\pi) \xrightarrow{a} \mu$ is not a transition in $\mathcal{A}$;
- $\frac{Q(\pi a \mu s)}{Q(\pi)\cdot\mu(s)}$ otherwise, where s is any state in $\mathsf{Supp}(\mu)$.

By Conditions (2) and (3), E_Q is well-defined and $E_Q(\pi)$ is a discrete sub-distribution for every π. Moreover, $E_Q(\pi)(a,\mu) \neq 0$ only if $\mathsf{last}(\pi) \xrightarrow{a} \mu$ is a transition in $\mathcal{A}$, therefore E_Q is an adversary for $\mathcal{A}$. It remains to prove $Q = \mathbf{Q}_{E_Q}$ (so that we have a right inverse of the operation $\mathbf{Q}$).

Lemma 4.1.1. *For all $\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$, we have $Q(\pi) = \mathbf{Q}_{E_Q}(\pi)$.*

Proof. By induction on the length of π. If π consists of just the initial state, then $Q(\pi) = 1 = \mathbf{Q}_{E_Q}(\pi)$.

Now consider π' of the form $\pi a \mu s$. If $Q(\pi) = 0$, then $Q(\pi') = 0$ by Condition 1. Also by induction hypothesis, $\mathbf{Q}_{E_Q}(\pi) = Q(\pi) = 0$. Hence $\mathbf{Q}_{E_Q}(\pi') = \mathbf{Q}_{E_Q}(\pi) \cdot E_Q(\pi)(a,\mu)\cdot\mu(s) = 0 = Q(\pi')$, regardless of the values of $E_Q(\pi)(a,\mu)$ and $\mu(s)$.

Otherwise, we may choose π'' as in the definition of $E_Q(\pi)(a,\mu)$. Let s' denote $\mathsf{last}(\pi'')$. Then

$$\begin{aligned}
\mathbf{Q}_{E_Q}(\pi') &= \mathbf{Q}_{E_Q}(\pi) \cdot E_Q(\pi)(a,\mu) \cdot \mu(s) && \text{definition } \mathbf{Q}_{E_Q} \\
&= Q(\pi) \cdot \frac{Q(\pi'')}{Q(\pi)\cdot\mu(s')} \cdot \mu(s) && \text{I.H. and definition of } E_Q(\pi')(a,\mu) \\
&= \frac{Q(\pi'')\cdot\mu(s)}{\mu(s')} \\
&= Q(\pi'). && \text{consistency of } Q
\end{aligned}$$

□

This completes the proof of the following characterization theorem.

Theorem 4.1.2 (Characterization of Probabilistic Executions). *For all $Q : \mathsf{Path}^{<\omega}(\mathcal{A}) \to [0,1]$, Q is the probabilistic execution induced by some adversary E of $\mathcal{A}$ if and only if Q satisfies Conditions (1), (2) and (3).*

4.2 Convex Combinations

We now use Theorem 4.1.2 to show that both $\mathsf{PExec}(\mathcal{A})$ and $\mathsf{TrDist}(\mathcal{A})$ are closed under convex combinations.

Lemma 4.2.1. *Let $p \in [0,1]$ be given and let E_0 and E_1 be adversaries of $\mathcal{A}$. There exists an adversary E of $\mathcal{A}$ such that $\mathbf{Q}_E = p \cdot \mathbf{Q}_{E_0} + (1-p) \cdot \mathbf{Q}_{E_1}$.*

Proof. Define $Q := p \cdot \mathbf{Q}_{E_0} + (1-p) \cdot \mathbf{Q}_{E_1}$. By Theorem 4.1.2, it suffices to verify Conditions (1), (2) and (3) of Section 4.1. The first two are straightforward. For Condition (3), let π, S_π and $\{s_{a,\mu} \mid \langle a, \mu\rangle \in S_\pi\}$ be given as stated. Then

$$
\begin{aligned}
&\sum_{(a,\mu)\in S_\pi} \frac{Q(\pi a \mu s_{a,\mu})}{Q(\pi)\cdot\mu(s_{a,\mu})} \\
&= \sum_{(a,\mu)\in S_\pi} \frac{p\cdot\mathbf{Q}_{E_0}(\pi a\mu s_{a,\mu}) + (1-p)\cdot\mathbf{Q}_{E_1}(\pi a\mu s_{a,\mu})}{Q(\pi)\cdot\mu(s_{a,\mu})} \\
&= \sum_{(a,\mu)\in S_\pi} \frac{p\cdot\mathbf{Q}_{E_0}(\pi)\cdot E_0(\pi)(a,\mu) + (1-p)\cdot\mathbf{Q}_{E_1}(\pi)\cdot E_1(\pi)(a,\mu)}{Q(\pi)} \\
&= \frac{p\cdot\mathbf{Q}_{E_0}(\pi)\cdot(\sum_{(a,\mu)\in S_\pi} E_0(\pi)(a,\mu))}{Q(\pi)} \\
&\qquad + \frac{(1-p)\cdot\mathbf{Q}_{E_1}(\pi)\cdot(\sum_{(a,\mu)\in S_\pi} E_1(\pi)(a,\mu))}{Q(\pi)} \\
&\leq \frac{p\cdot\mathbf{Q}_{E_0}(\pi) + (1-p)\cdot\mathbf{Q}_{E_1}(\pi)}{Q(\pi)} = 1
\end{aligned}
$$

□

The next lemma says that tr preserves convex combinations. It follows immediately from the definition of $\mathsf{tr} : \mathsf{PExec}(\mathcal{A}) \to (\mathsf{Act}^{<\omega} \to [0,1])$ (cf. Section 3.2).

Lemma 4.2.2. *Let $p \in [0,1]$ be given and let E_0 and E_1 be adversaries of $\mathcal{A}$. Then*

$$\mathsf{tr}(p\cdot\mathbf{Q}_{E_0} + (1-p)\cdot\mathbf{Q}_{E_1}) = p\cdot\mathsf{tr}(\mathbf{Q}_{E_0}) + (1-p)\cdot\mathsf{tr}(\mathbf{Q}_{E_1}).$$

Theorem 4.2.3. *The set of trace distributions of $\mathcal{A}$ is closed under convex combinations.*

Proof. By Lemmas 4.2.1 and 4.2.2. □

We have a corollary concerning the discrete probability distributions $\mathbf{P}_{D,k}$ and $\mathbf{P}_{D,k,l}$ (cf. Sections 3.2 and 3.3).

Corollary 4.2.4. *For all $k, l \in \mathbb{N}$, the sets $\{\mathbf{P}_{D,k} \mid D \in \mathsf{TrDist}(\mathcal{A})\}$ and $\{\mathbf{P}_{D,k,l} \mid D \in \mathsf{TrDist}(\mathcal{A}, k, l)\}$ are closed under convex combinations.*

Proof. By Theorem 4.2.3 and the definition of $\mathbf{P}_{D,k}$. □

4.3 Limit Construction

In this section, we give a construction that yields an adversary E from any infinite sequence $\{E_i \mid i \in \mathbb{N}\}$ of adversaries. This is done roughly as follows. First, we construct an infinite decreasing sequence of sequences:

(i) set the initial sequence $\{E_j^0 \mid j \in \mathbb{N}\}$ to be $\{E_i \mid i \in \mathbb{N}\}$;

(ii) for each $n \in \mathbb{N}$, define a subsequence $\{E_j^{n+1} \mid j \in \mathbb{N}\}$ of $\{E_j^n \mid j \in \mathbb{N}\}$.

While choosing the appropriate subsequences, we obtain a function

$$Q : \mathsf{Path}^{<\omega}(\mathcal{A}) \to [0, 1]$$

such that Q is the probabilistic execution induced by some adversary E. Once we specify our notion of convergence, such E is an obvious candidate for the limit of $\{E_i \mid i \in \mathbb{N}\}$.

By assumption, $\mathcal{A}$ is countably branching, hence $\mathsf{Path}^{<\omega}(\mathcal{A})$ is countable. Let $\{\pi_n \mid n \in \mathbb{N}\}$ be an enumeration of that set. Given $n \in \mathbb{N}$, the sequence $\{\mathbf{Q}_{E_j^n}(\pi_n) \mid j \in \mathbb{N}\}$ is an infinite sequence in $[0, 1]$. By Theorem 2.0.1, there is a convergent subsequence. Let $\{E_j^{n+1} \mid j \in \mathbb{N}\}$ be a subsequence of $\{E_j^n \mid j \in \mathbb{N}\}$ such that $\{\mathbf{Q}_{E_j^{n+1}}(\pi_n) \mid j \in \mathbb{N}\}$ converges. Define

$$Q(\pi_n) := \lim_{j \in \mathbb{N}} \mathbf{Q}_{E_j^{n+1}}(\pi_n).$$

Given an adversary E_j^n as above, let $\mathsf{index}(E_j^n)$ denote the index of E_j^n in the original sequence $\{E_i \mid i \in \mathbb{N}\}$. In other words, $\mathsf{index}(E_j^n) = f_1(f_2(\ldots f_n(j)\ldots))$, where each $f_n : \mathbb{N} \to \mathbb{N}$ specifies the subsequence chosen in stage n.

The idea here is, at each stage n, we decide the value of Q at path π_n. Moreover, we remove those adversaries whose probabilistic executions (evaluated at π_n) fail to converge to $Q(\pi_n)$, taking care that we still have infinitely many adversaries left. As a consequence, at every stage after n, the probabilistic executions of remaining adversaries converge to the same limit at π_n. This claim is formalized in the following lemma.

Lemma 4.3.1. *For all $n < n'$, $\{\mathbf{Q}_{E_j^{n'}}(\pi_n) \mid j \in \mathbb{N}\}$ converges to $Q(\pi_n)$.*

Proof. For all $n < n'$, $\{E_j^{n'} \mid j \in \mathbb{N}\}$ is a subsequence of $\{E_j^n \mid j \in \mathbb{N}\}$. Hence sequence $\{\mathbf{Q}_{E_j^{n'}}(\pi_n)\}_{j\in\mathbb{N}}$ converges to the same limit as $\{\mathbf{Q}_{E_j^n}(\pi_n)\}_{j\in\mathbb{N}}$, namely, to $Q(\pi_n)$. □

Corollary 4.3.2. *Let $S \subseteq \mathbb{N}$ be finite. For all $n \in S$, $\{\mathbf{Q}_{E_j^{\max(S)+1}}(\pi_n) \mid j \in \mathbb{N}\}$ converges to $Q(\pi_n)$.*

The meaning of Corollary 4.3.2 is best explained by: "finitely many is the same as just one." Instead of taking the defining sequence of $Q(\pi_n)$ for each n, we can simply go to a much later stage in the construction where, for each $n \in S$, the weight on π_n is guaranteed to converge to the right value. Notice it is essential that S is finite. With this idea in mind, we prove that Q satisfies Conditions (1), (2) and (3) as in Section 4.1; then we apply Theorem 4.1.2 to conclude there is an adversary E with $\mathbf{Q}_E = Q$.

Lemma 4.3.3 (Condition (1)). *Let $\pi, \pi' \in \mathsf{Path}^{<\omega}(\mathcal{A})$ be given. Suppose π is a prefix of π', then $Q(\pi) \geq Q(\pi')$. Moreover, $Q(s^0) = 1$.*

Proof. The second claim follows from the definition of Q. For the first, we choose $n, n' \in \mathbb{N}$ such that $\pi = \pi_n$ and $\pi' = \pi_{n'}$. Let $N := \max(n, n')$. Since every E_j^{N+1} is an adversary of $\mathcal{A}$, we have $\mathbf{Q}_{E_j^{N+1}}(\pi) \geq \mathbf{Q}_{E_j^{N+1}}(\pi')$. Therefore, by Corollary 4.3.2,

$$Q(\pi) = \lim_{j\in\mathbb{N}} \mathbf{Q}_{E_j^{N+1}}(\pi) \geq \lim_{j\in\mathbb{N}} \mathbf{Q}_{E_j^{N+1}}(\pi') = Q(\pi').$$

□

The following lemmas verify Conditions (2) and (3).

Lemma 4.3.4 (Condition (2)). *Let $n, n_1, n_2 \in \mathbb{N}$ be given. Suppose $\pi_{n_1} = \pi_n a \mu s_1$, $\pi_{n_2} = \pi_n a \mu s_2$, $\mathsf{last}\,\pi_n \xrightarrow{a} \mu$ and $s_1, s_2 \in \mathsf{Supp}\,\mu$. Then*

$$\frac{Q(\pi_{n_1})}{\mu(s_1)} = \frac{Q(\pi_{n_2})}{\mu(s_2)}.$$

Proof. Let $N := \max(n_1, n_2)$. By Corollary 4.3.2 and the consistency of $\mathbf{Q}_{E_j^{N+1}}$, we have

$$\frac{Q(\pi_{n_1})}{\mu(s_1)} = \lim_{j\in\mathbb{N}} \frac{\mathbf{Q}_{E_j^{N+1}}(\pi_{n_1})}{\mu(s_1)} = \lim_{j\in\mathbb{N}} \frac{\mathbf{Q}_{E_j^{N+1}}(\pi_{n_2})}{\mu(s_2)} = \frac{Q(\pi_{n_2})}{\mu(s_2)}.$$

□

Lemma 4.3.5 (Condition (3)). *Let π be a path in $\mathsf{Path}^{<\omega}(\mathcal{A})$ such that $Q(\pi) \neq 0$. Recall that S_π denotes the set $\{(a,\mu) \mid \mathsf{last}\,\pi \xrightarrow{a} \mu\}$. For each $(a,\mu) \in S_\pi$, let $s_{a,\mu} \in \mathsf{Supp}\,\mu$ be given. Then*

$$\sum_{(a,\mu)\in S_\pi} \frac{Q(\pi a \mu s_{a,\mu})}{Q(\pi)\cdot\mu(s_{a,\mu})} \leq 1.$$

Proof. Let $\{(a_k,\mu_k)\}_{k\in\mathbb{N}}$ be a (possibly finite) enumeration of S_π. It suffices to show that all finite partial sums are below 1. Let $K \in \mathbb{N}$ be given. For each $0 \leq k \leq K$, let n_k be the index of $\pi a_k \mu_k s_{a_k,\mu_k}$ in the enumeration $\{\pi_n \mid n \in \mathbb{N}\}$. Similarly, let n be the index of π. Define N to be $\max\{n_0,\ldots,n_K,n\}+1$. Then by Corollary 4.3.2 we have

$$\sum_{k=0}^{K} \frac{Q(\pi_{n_k})}{Q(\pi)\cdot\mu_k(s_{a_k,\mu_k})} = \sum_{k=0}^{K} \frac{\lim_{j\in\mathbb{N}} \mathbf{Q}_{E_j^N}(\pi_{n_k})}{Q(\pi)\cdot\mu_k(s_{a_k,\mu_k})}$$

By the definition of $\mathbf{Q}_{E_j^N}$, this becomes

$$\begin{aligned}
&\sum_{k=0}^{K} \lim_{j\in\mathbb{N}} \frac{\mathbf{Q}_{E_j^N}(\pi)\cdot E_j^N(\pi)(a_k,\mu_k)\cdot\mu_k(s_{a_k,\mu_k})}{Q(\pi)\cdot\mu_k(s_{a_k,\mu_k})} \\
&= \sum_{k=0}^{K} \lim_{j\in\mathbb{N}} \frac{\mathbf{Q}_{E_j^N}(\pi)\cdot E_j^N(\pi)(a_k,\mu_k)}{Q(\pi)} \\
&= \lim_{j\in\mathbb{N}} \frac{\mathbf{Q}_{E_j^N}(\pi)}{Q(\pi)} \sum_{k=0}^{K} E_j^N(\pi)(a_k,\mu_k) && \text{finite sum} \\
&\leq \lim_{j\in\mathbb{N}} \frac{\mathbf{Q}_{E_j^N}(\pi)}{Q(\pi)} && E_j^N(\pi) \text{ subdistribution} \\
&= 1. && \text{Corollary 4.3.2}
\end{aligned}$$

□

Proposition 4.3.6. *Suppose $\mathcal{A}$ is countably branching. Let $\{E_i \mid i \in \mathbb{N}\}$ be a sequence of adversaries of $\mathcal{A}$ and let $Q : \mathsf{Path}^{<\omega}(\mathcal{A}) \to [0,1]$ be constructed as in the present section. Then there exists $E \in \mathsf{Adv}(\mathcal{A})$ such that $Q = \mathbf{Q}_E$.*

Proof. By Theorem 4.1.2 and Lemmas 4.3.3, 4.3.4 and 4.3.5. □

So far we have presented a construction that yields an adversary from any given countable sequence of adversaries. Let us now consider two examples in which this construction fails to provide a sensible "limit".

Example 4.3.1. Consider the infinitely branching automaton $\mathcal{A}$ drawn in Figure 4.1, where all transitions are labeled with symbol a and all target distributions are Dirac distributions. Consider this sequence $\{E_k \mid k \in \mathbb{N}\}$ of adversaries:

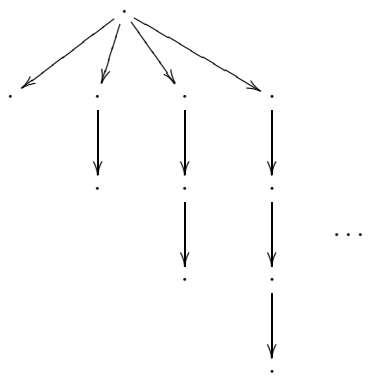

Figure 4.1: An automaton that is *not* image finite (cf. Example 4.3.1).

each E_k follows the kth branch of $\mathcal{A}$ with probability 1 and halts at the end of that branch. Thus, the trace distribution of E_k assigns 1 to every prefix of a^k, which denotes the length-k trace containing all a's. Intuitively, the limit of this sequence of trace distributions should assign 1 to the infinite trace a^ω; yet such an adversary cannot be constructed, simply because $\mathcal{A}$ has no infinite paths. In fact, our limit construction yields the everywhere-0 adversary.

Example 4.3.2. Consider automaton $\mathcal{A}$ as in Example 4.3.1 and Figure 4.1. Take the following sequence $\{E_k \mid k \in \mathbb{N}\}$ of adversaries: (i) at the start state, each E_k schedules the k-th transition with probability $\frac{2^k-1}{2^k}$ and halts with probability $\frac{1}{2^k}$; (ii) every E_k halts completely after one step. This sequence of adversaries induce the following sequence of trace distributions: for every $k \in \mathbb{N}$, $\mathbf{D}_{E_k}(a) = \frac{2^k-1}{2^k}$ and $\mathbf{D}_{E_k}(\beta) = 0$ for every other non-empty trace β. Intuitively, this is a converging sequence whose limit assigns 1 to the trace a. However, the limit of $\{E_k \mid k \in \mathbb{N}\}$, as constructed in the present section, is again the everywhere-0 adversary.

In Chapter 5, we will prove CPO properties of $\mathsf{PExec}(\mathcal{A})$ and $\mathsf{TrDist}(\mathcal{A})$ for image finite $\mathcal{A}$. In particular, that shows image finiteness is sufficient to remove Counterexample 4.3.1. In Chapter 6, we prove that image finiteness implies $\mathsf{TrDist}(\mathcal{A}, k, l)$ forms a closed set in the metric space $[0,1]^{\mathsf{Act}^{<\omega}}$ via the limit construction of the present section. Thus Counterexample 4.3.2 is also removed.

5

Order Structures

The concept of approximations is ubiquitous in the studies of mathematics and computer science. A fundamental example is the real numbers, where every member can be approximated by a sequence of rational numbers. Moreover, every continuous function on the reals is completely specified by its valuation on the rationals. This is extremely important because the rationals have finite representations and therefore can be manipulated by a machine in a straightforward fashion.

In this chapter, we consider approximations of a more "discrete" character. Recall the prefix ordering on sequences (Example 2.0.6). This order structure induces a very simple notion of approximation: any sequence β is "approximated" by its finite prefixes. Similarly, in the case of subset ordering on partial functions (Example 2.0.7), a function g is "approximated" by its sub-functions, namely, functions f such that, if f is defined on x, then g is also defined on x and $f(x) = g(x)$.

Generalizing these ideas, one can define a "subtree" ordering on the set of computation trees of a nondeterministic automaton. Each computation tree can be viewed as a prefix-closed set of paths of the automaton. This in turn can be viewed as a function from the universal set of paths to $\{0, 1\}$ (namely, the characteristic function of a subset). The subtree ordering is then induced pointwise by the obvious ordering on $\{0, 1\}$. Pictorially, a tree T_1 is a subtree of another tree T_2 if and only if T_1 can be obtained from T_2 by pruning away certain branches.

Our notions of approximation for probabilistic processes are extensions of the subtree ordering to a probabilistic setting. As we shall see, the various semantic objects (i.e., adversaries, probabilistic executions and trace distributions) associated with a PA $\mathcal{A}$ can be viewed as functions with codomain $[0, 1]$. Therefore, we can define orderings pointwise, based on the *flat* ordering on $[0, 1]$: $p \leq_\flat p'$ if and only if, $p \neq 0$ implies $p = p'$.

More precisely, we use $\leq_\flat$ to define partial orders on these three sets: $\mathsf{Adv}(\mathcal{A})$, $\mathsf{PExec}(\mathcal{A})$ and $\mathsf{TrDist}(\mathcal{A})$. We show that in the first two cases we obtain algebraic CPOs. The same holds in the third case, if we assume in addition that $\mathcal{A}$ is image finite. The compact elements of $\mathsf{Adv}(\mathcal{A})$ are those adversaries that return 0 on all but a finite number of triples $\langle \pi, a, \mu \rangle$. The compact elements of $\mathsf{PExec}(\mathcal{A})$

are those generated by compact adversaries. Finally, the compact elements of $\mathsf{TrDist}(\mathcal{A})$ are precisely the finite trace distributions defined in Section 3.3.

We also prove that the operator $\mathbf{Q} : \mathsf{Adv}(\mathcal{A}) \to \mathsf{PExec}(\mathcal{A})$ is continuous and bottom preserving. On the other hand, the operator $\mathsf{tr} : \mathsf{PExec}(\mathcal{A}) \to \mathsf{TrDist}(\mathcal{A})$ is *not* continuous, as illustrated by a counterexample.

Related Work

In a process algebraic setting, one can prove an *Approximation Induction Principle (AIP)*, which says that inclusion of finite traces implies inclusion of all traces [BK86, BBK87]. This holds because every automaton (or transition system) induced by an algebraic term is image finite. As illustrated in Example 5.0.3 below, the AIP fails when we include automata that are not image finite.

Example 5.0.3. Consider the automaton $\mathcal{A}$ in Figure 4.1 (cf. Example 4.3.1). Construct $\mathcal{A}'$ by adding an infinite a-path to $\mathcal{A}$. Then every finite trace of $\mathcal{A}'$ is a trace of $\mathcal{A}$, but clearly the infinite trace a^ω is not a trace of $\mathcal{A}$.

In the PhD thesis [Sto02a], Stoelinga proved the following AIP for probabilistic processes, giving a useful tool for proving trace distribution inclusion. A very similar result was observed by Segala [Seg96], who presented an informal proof sketch. The proof in [Sto02a] involves an explicit construction of the desired adversary of $\mathcal{B}$, relying on the fact that $\mathcal{B}$ is finitely branching.

Theorem 5.0.7. *Let $\mathcal{A}$ and $\mathcal{B}$ be PAs and let $\mathcal{B}$ be finitely branching. If $\mathcal{A} \leq_{\mathsf{td}}^{k,-} \mathcal{B}$ for all $k \in \mathbb{N}$, then $\mathcal{A} \leq_{\mathsf{td}} \mathcal{B}$.*

The present chapter recasts and extends the development of [Sto02a] in an order theoretic setting, relaxing the finite branching requirement to image finiteness. (Note that Theorem 5.0.7 is a corollary of the fact that $\mathsf{TrDist}(\mathcal{A})$ forms a CPO.) Since nondeterministic automata are degenerate cases of probabilistic automata, Example 5.0.3 suggests that image finiteness is a necessary assumption for the claim that $\mathsf{TrDist}(\mathcal{A})$ forms a CPO.

We now point out some fundamental differences between our treatment of approximations and that found in [DGJP03] for labeled Markov processes (LMPs). To begin, we note that the theory of LMPs is based on a reactive interpretation of discrete events. Namely, the action labels represent various input events that may be received from the environment and the transitions describe how the system "reacts" to these inputs. Because of this interpretation, their notion of behavioral equivalence is inherently branching: two systems are "bisimilar" if they react to every input in "bisimilar" ways. Their finite-state approximations are then obtained from the fixed-point computation of bisimulation, by refining the state partition at each step and adding more transitions.

In contrast, we work with a linear notion of behavioral equivalence, based on probability distributions on computation paths and traces. Our finite approximations are obtained by truncating infinite behavior. In the case of trace distributions, we truncate both in depth and in breadth. Although our formal model is time-abstract, truncating in depth can be thought of as halting the system after a finite amount of time. On the other hand, truncating in breadth can be thought of as disabling irrelevant actions; that is, we choose l large enough so that all interesting actions are contained in Act_l.

Our order-theoretic definitions are similar to those in [Vat01], which studies properties of distribution functions (a generalized notion of language) generated by finite-state probabilistic automata. However, since our model is intended for verification, we allow infinitely many states, actions and transitions. This renders our development more complicated than [Vat01].

5.1 Adversaries

We define the *flat* ordering on $\mathsf{Adv}(\mathcal{A})$: $E \leq_\flat E'$ if, for all finite executions π, action symbols a and state distributions μ, $E(\pi)(a,\mu) \neq 0$ implies $E(\pi)(a,\mu) = E'(\pi)(a,\mu)$. As the name suggests, this is essentially the same ordering on $[0,1]^\omega$ defined in Section 2.

First we show that $\mathsf{Adv}(\mathcal{A})$, ordered by $\leq_\flat$, is closed under directed joins. Let $\mathcal{D}$ be a directed subset of $\mathsf{Adv}(\mathcal{A})$. Given $\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$, $a \in \mathsf{Act}$ and $\mu \in \mathsf{Disc}(S_\mathcal{A})$, define $\widehat{E}(\pi)(a,\mu) := \bigvee_{E\in\mathcal{D}} E(\pi)(a,\mu)$. In other words, $\widehat{E}$ is the pointwise join of $\mathcal{D}$ in the function space $\mathsf{Path}^{<\omega}(\mathcal{A}) \times \mathsf{Act} \times \mathsf{Disc}(S_\mathcal{A}) \to [0,1]$. Our task is to show that $\widehat{E}$ is an adversary.

Fix $\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$. Notice that $\widehat{E}(\pi)$ assigns non-zero probability to $\langle a, \mu\rangle$ if and only if some E in $\mathcal{D}$ does. Hence

$$\langle a, \mu\rangle \in \mathsf{Supp}(\widehat{E}(\pi)) \Rightarrow \exists E \in \mathcal{D}, \langle a, \mu\rangle \in \mathsf{Supp}(E(\pi)) \Rightarrow \mathsf{last}(\pi) \xrightarrow{a} \mu.$$

It remains to show $\widehat{E}(\pi)$ is a sub-distribution.

Lemma 5.1.1. *For all finite executions π, the function $\widehat{E}(\pi)$ is a probabilistic sub-distribution over $\mathsf{Act} \times \mathsf{Disc}(S_\mathcal{A})$.*

Proof. Let X_π denote the set $\{\langle a, \mu\rangle \mid \mathsf{last}(\pi) \xrightarrow{a} \mu\} \subseteq \mathsf{Act} \times \mathsf{Disc}(S_\mathcal{A})$. Since Act is countable and $\mathcal{A}$ is countably branching assumption, X_π is also countable. Moreover, $E(\pi)$ is a sub-distribution over X_π for every adversary E.

Since $\mathcal{D}$ is directed, the set $\{E(\pi) \mid E \in \mathcal{D}\}$ is also directed. We can now apply Corollary 2.0.7 to conclude that $\widehat{E}(\pi)$ is also a sub-distribution over X_π. □

Theorem 5.1.2. *Let $\mathcal{A}$ be a countably branching PA. The set of adversaries of $\mathcal{A}$, ordered by $\leq_\flat$, is a CPO.*

Proof. By Lemma 5.1.1, the set $\mathsf{Adv}(\mathcal{A})$ equipped with the flat ordering is closed under directed limits. Clearly, the everywhere-0 adversary is a bottom element in this structure. Therefore we have a CPO. □

Compact Elements in $\mathsf{Adv}(\mathcal{A})$

We now show that $\langle \mathsf{Adv}(\mathcal{A}), \leq_\flat \rangle$ is in fact an algebraic CPO structure. First we identify the compact elements.

Definition 5.1.1. Let an adversary E be given. We say that E is *compact* if the set $\{\langle \pi, a, \mu \rangle \mid E(\pi)(a, \mu) \neq 0\}$ is finite.

Lemmas 5.1.3 and 5.1.4 below show that Definition 5.1.1 indeed characterizes the compact elements of the CPO $\langle \mathsf{Adv}(\mathcal{A}), \leq_\flat \rangle$.

Lemma 5.1.3. *Let $E \in \mathsf{Adv}(\mathcal{A})$ be given and assume that E is compact in the sense of Definition 5.1.1. Let $\mathcal{D}$ be a directed set in $\langle \mathsf{Adv}(\mathcal{A}), \leq_\flat \rangle$ such that $E \leq_\flat \bigvee \mathcal{D}$. Then there exists adversary $E' \in \mathcal{D}$ with $E \leq_\flat E'$.*

Proof. For each $\langle \pi, a, \mu \rangle$ with $E(\pi)(a, \mu) \neq 0$, choose $E'_{\pi,a,\mu} \in \mathcal{D}$ such that $E'(\pi)(a, \mu) = E(\pi)(a, \mu)$. This is possible because, by assumption, $E \leq_\flat \bigvee \mathcal{D}$. Since E is compact, the set $\{E'_{\pi,a,\mu} \mid E(\pi)(a, \mu) \neq 0\}$ is finite. This is a subset of the directed set $\mathcal{D}$, hence it has an upper bound E' in $\mathcal{D}$. Clearly, $E \leq_\flat E'$. □

Lemma 5.1.4. *Let $E \in \mathsf{Adv}(\mathcal{A})$ be given and assume that E is* not *compact in the sense of Definition 5.1.1. There exists a directed set $\mathcal{D}$ in $\langle \mathsf{Adv}(\mathcal{A}), \leq_\flat \rangle$ such that $E \leq_\flat \bigvee \mathcal{D}$ and yet $E' \lneq_\flat E$ for every $E' \in \mathcal{D}$.*

Proof. By assumption, the set $X_E = \{\langle \pi, a, \mu \rangle \mid E(\pi)(a, \mu) \neq 0\}$ is infinite. It is countable since Act is countable and $\mathcal{A}$ is countably branching. Consider an enumeration $\{\langle \pi_i, a_i, \mu_i \rangle \mid i \in \mathbb{N}\}$ of X_E. For every $i \in \mathbb{N}$, define E_i as follows:

- $E_i(\pi)(a, \mu) = E(\pi)(a, \mu)$ whenever $\langle \pi, a, \mu \rangle = \langle \pi_j, a_j, \mu_j \rangle$ for some $j \leq i$;
- $E_i(\pi)(a, \mu) = 0$ otherwise.

Clearly, this is a chain with limit E. Yet, for every i, $E_i \lneq_\flat E$ because

$$E_i(\pi_{i+1})(a_{i+1}, \mu_{i+1}) = 0 \neq E(\pi_{i+1})(a_{i+1}, \mu_{i+1}).$$

□

It remains to prove that, for every adversary E, the set of compact elements below E is directed and the join is precisely E.

Lemma 5.1.5. *Let E be an adversary of $\mathcal{A}$. Let K_E denote the set of compact elements below E; that is,*

$$K_E := \{E' \mid E' \text{ compact in the sense of Definition 5.1.1 and } E' \leq_\flat E\}.$$

Then K_E is directed and $E = \bigvee K_E$.

Proof. As in the proof of Lemma 5.1.4, we enumerate the set

$$X_E = \{\langle \pi, a, \mu\rangle \mid E(\pi)(a,\mu) \neq 0\}$$

as $\{\langle \pi_i, a_i, \mu_i\rangle \mid i \in \mathbb{N}\}$ and define $\{E_i \mid i \in \mathbb{N}\}$ using this enumeration. Let F be a finite subset of K_E and let $E' \in F$ be given. Since $E' \leq_\flat E$, we know that

$$X_{E'} = \{\langle \pi, a, \mu\rangle \mid E'(\pi)(a,\mu) \neq 0\} \subseteq X_E.$$

Since E' is compact and F is finite, we may choose $M \in \mathbb{N}$ such that $\bigcup_{E' \in F} X_{E'}$ is included in $\{\langle \pi_i, a_i, \mu_i\rangle \mid 0 \leq i \leq M\}$. Then E_M is an upperbound of F in K_E. Therefore K_E is directed. Finally, we note that $\bigvee K_E = \vee_{i\in\mathbb{N}} E_i = E$. □

This completes the proof of the following theorem.

Theorem 5.1.6. *Given a countably branching PA$\mathcal{A}$, the structure $\langle \mathsf{Adv}(\mathcal{A}), \leq_\flat\rangle$ is an algebraic CPO.*

5.2 Probabilistic Executions

For $\mathsf{PExec}(\mathcal{A})$, we also consider a *flat* ordering: $Q_1 \leq_\flat Q_2$ if, for all $\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$, $Q_1(\pi) \neq 0$ implies $Q_1(\pi) = Q_2(\pi)$. As in the case of adversaries, we want to show that $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat\rangle$ forms a CPO.

Let $\mathcal{D}$ be a directed subset of $\mathsf{PExec}(\mathcal{A})$. We claim that the pointwise join of $\mathcal{D}$ in the function space $\mathsf{Path}^{<\omega}(\mathcal{A}) \to [0,1]$ is also a probabilistic execution. By Theorem 4.1.2, it suffices to show $\bigvee \mathcal{D}$ satisfies the three properties in Section 4.1. Conditions (1) and (2) follow directly from the definition of pointwise joins. Lemma 5.2.1 below verifies Condition (3).

Lemma 5.2.1. *Let $\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$ be given and suppose $(\bigvee \mathcal{D})(\pi) \neq 0$. Let S_π denote the set of pairs $\langle a, \mu\rangle$ such that $\mathsf{last}(\pi) \xrightarrow{a} \mu$. For each $\langle a, \mu\rangle \in S_\pi$, fix any $s_{a,\mu} \in \mathsf{Supp}(\mu)$. Then*

$$\sum_{\langle a,\mu\rangle \in S_\pi} \frac{(\bigvee \mathcal{D})(\pi a \mu s_{a,\mu})}{(\bigvee \mathcal{D})(\pi) \cdot \mu(s_{a,\mu})} \leq 1.$$

Proof. First we apply Lemma 2.0.5 to obtain an ω-chain $C = \{Q_i \mid i \in \mathbb{N}\} \subseteq \mathcal{D}$ such that $\bigvee C = \bigvee \mathcal{D}$. Since C is increasing, $\bigvee C(\pi) = \lim_{i\in\mathbb{N}} Q_i(\pi)$ for every

$\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$. Since C is increasing and $(\bigvee C)(\pi) \neq 0$, we may assume without loss of generality that $Q_i(\pi) \neq 0$ for all i. Then

$$\begin{aligned}
&\sum_{\langle a,\,\mu\rangle \in X_\pi} \frac{\bigvee C(\pi a \mu s_{a,\mu})}{\bigvee C(\pi) \cdot \mu(s_{a,\mu})} \\
&= \sum_{\langle a,\,\mu\rangle \in X_\pi} \frac{\lim_{i\in\mathbb{N}} Q_i(\pi a \mu s_{a,\mu})}{\lim_{i\in\mathbb{N}} Q_i(\pi) \cdot \mu(s_{a,\mu})} \\
&= \sum_{\langle a,\,\mu\rangle \in X_\pi} \lim_{i\in\mathbb{N}} \frac{Q_i(\pi a \mu s_{a,\mu})}{Q_i(\pi) \cdot \mu(s_{a,\mu})} && \text{non-zero denominator} \\
&= \bigvee_{F \in \mathcal{P}_{\mathsf{fin}}(X_\pi)} \sum_{\langle a,\,\mu\rangle \in F} \lim_{i\in\mathbb{N}} \frac{Q_i(\pi a \mu s_{a,\mu})}{Q_i(\pi) \cdot \mu(s_{a,\mu})} \\
&= \bigvee_{F \in \mathcal{P}_{\mathsf{fin}}(X_\pi)} \lim_{i\in\mathbb{N}} \sum_{\langle a,\,\mu\rangle \in F} \frac{Q_i(\pi a \mu s_{a,\mu})}{Q_i(\pi) \cdot \mu(s_{a,\mu})} && \text{finite sum} \\
&\leq 1 && \text{every } Q_i \text{ satisfies Condition (3)}
\end{aligned}$$

□

We may now conclude that the set $\mathsf{PExec}(\mathcal{A})$ equipped with the flat ordering is a CPO.

Theorem 5.2.2. *For a countably branching PA $\mathcal{A}$, the set of probabilistic executions of $\mathcal{A}$ forms a CPO under $\leq_\flat$. The bottom element is the probabilistic execution generated by the everywhere-0 adversary.*

Continuity of Operator Q

Recall that $\mathbf{Q}$ is an operator from $\mathsf{Adv}(\mathcal{A})$ to $\mathsf{PExec}(\mathcal{A})$. So far, both the domain and codomain of $\mathbf{Q}$ have been given a CPO structure. Naturally, we proceed with a proof that $\mathbf{Q}$ is continuous. It is trivial to note that $\mathbf{Q}$ is also *strict* (i.e., bottom preserving).

Lemma 5.2.3. *The operator $\mathbf{Q}$ is monotone.*

Proof. Let $E_1 \leq_\flat E_2$ be given. We show that $\mathbf{Q}_{E_1} \leq_\flat \mathbf{Q}_{E_2}$, by induction on the length of execution π. The base case is trivial. Take an execution π' of the form $\pi a \mu s$ and assume $\mathbf{Q}_{E_1}(\pi') \neq 0$. Then $\mathbf{Q}_{E_1}(\pi) \neq 0$; applying the inductive hypothesis, we have $\mathbf{Q}_{E_1}(\pi) = \mathbf{Q}_{E_2}(\pi)$. On the other hand, we have $E_1(\pi)(a,\mu) \neq 0$ and $E_1 \leq_\flat E_2$, thus $E_1(\pi)(a,\mu) = E_2(\pi)(a,\mu)$. Hence

$$\mathbf{Q}_{E_1}(\pi') = \mathbf{Q}_{E_1}(\pi) \cdot E_1(\pi)(a,\mu) \cdot \mu(s) = \mathbf{Q}_{E_2}(\pi) \cdot E_2(\pi)(a,\mu) \cdot \mu(s) = \mathbf{Q}_{E_2}(\pi').$$

□

An important property of monotone operators is the following: the image of any directed set is again directed. Therefore, it makes sense to talk about the join of a directed image. The following lemma confirms that $\mathbf{Q}$ indeed commutes with directed join.

Lemma 5.2.4. *Let $\mathcal{D}$ be a directed set of adversaries. We have $\bigvee_{E\in\mathcal{D}} \mathbf{Q}_E = \mathbf{Q}_{\bigvee \mathcal{D}}$.*

Proof. Induction on the length of execution π. Since $\mathbf{Q}_E(s^0) = 1$ for every adversary E, the base case is trivial. For the inductive step, take an execution of the form $\pi a\mu s$ and let $\widehat{E}$ denote $\bigvee \mathcal{D}$. The following holds:

$$\begin{aligned}
\mathbf{Q}_{\widehat{E}}(\pi a\mu s) &= \mathbf{Q}_{\widehat{E}}(\pi)\cdot \widehat{E}(\pi,a,\mu)\cdot\mu(s)\\
&= \bigvee_{E\in\mathcal{D}} \mathbf{Q}_E(\pi)\cdot \bigvee_{E'\in\mathcal{D}} E'(\pi,a,\mu)\cdot\mu(s) && \text{I.H. and definition } \widehat{E}\\
&= \bigvee_{E,E'\in\mathcal{D}} \mathbf{Q}_E(\pi)\cdot E'(\pi,a,\mu)\cdot\mu(s)\\
&= \bigvee_{E\in\mathcal{D}} \mathbf{Q}_E(\pi)\cdot E(\pi,a,\mu)\cdot\mu(s) && \mathcal{D} \text{ directed and Lemma 5.2.3}\\
&= \bigvee_{E\in\mathcal{D}} \mathbf{Q}_E(\pi a\mu s).
\end{aligned}$$

□

This yields the following theorem.

Theorem 5.2.5. *The operator $\mathbf{Q} : \mathsf{Adv}(\mathcal{A}) \to \mathsf{PExec}(\mathcal{A})$ is strictly continuous.*

Compact Elements in $\mathsf{PExec}(\mathcal{A})$

We proceed to characterize compact elements in the CPO $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat\rangle$. For motivation, let us first consider a concrete example.

Example 5.2.1. Consider an automaton $\mathcal{A}$ with state space $\{s_0, s_1, s_2, \ldots\}$, where s_0 is the unique start state. The transition relation consists of a single triple $\langle s_0, a, \mu\rangle$, where μ assigns probability $\frac{1}{2^i}$ to each s_i. Let E be the adversary that schedules the transition $s_0 \xrightarrow{a} \mu$ with probability 1. Consider the probabilistic execution $\mathbf{Q}_E$. The only probabilistic execution strictly below $\mathbf{Q}_E$ is the bottom element of $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat\rangle$, which assigns 0 to every non-empty finite path. Therefore $\mathbf{Q}_E$ is compact in $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat\rangle$, despite the fact that $\mathbf{Q}_E(s_0 a\mu s_i) \neq 0$ for every $i \in \mathbb{N}$.

In light of Example 5.2.1, we cannot characterize compact elements in $\mathsf{PExec}(\mathcal{A})$ by simply counting the number of finite paths with non-zero probability mass. Thus we explore another possibility, namely, images of compact elements in

$\langle \mathsf{Adv}(\mathcal{A}), \leq_\flat \rangle$. This seems very natural since $\mathsf{PExec}(\mathcal{A})$ is the image of $\mathsf{Adv}(\mathcal{A})$ under the continuous operator $\mathbf{Q}$.

Example 5.2.2. It is not true in general that a continuous function always maps compact elements to compact elements, even if the function is onto. This is because there may be more directed sets in the codomain than there are in the domain. Consider

$$\{\bot, \top_0, \top_1\} \cup \{\langle 0, n\rangle \mid n \in \mathbb{N}\} \cup \{\langle 1, n\rangle \mid n \in \mathbb{N}\},$$

where $\bot$ is a bottom element below two independent copies of $\mathbb{N}^\top$. We can map this onto

$$\{\bot, \top\} \cup \{\langle 0, n\rangle \mid n \in \mathbb{N}\} \cup \{\langle 1, n\rangle \mid n \in \mathbb{N}\}$$

with the lexicographic ordering. We do so by taking $\top_1$ to $\top$ and merging $\top_0$ with $\langle 1, 0\rangle$. This is a continuous map, but $\langle 1, 0\rangle$ is compact in the domain and not compact in the codomain.

As it turns out, the operator $\mathbf{Q}$ behaves more nicely than the continuous map in Example 5.2.2. In particular, every $Q \in \mathsf{PExec}(\mathcal{A})$ has a "canonical" inverse in $\mathsf{Adv}(\mathcal{A})$, namely, the adversary E_Q constructed in Section 4.1. These canonical inverses enjoy an important property: if $Q \leq_\flat Q'$, then $E_Q \leq_\flat E_{Q'}$. In other words, the canonical inverse operation is also monotone. As a consequence, it gives rise to a special preimage operation that preserves directedness. These claims are proven in Lemma 5.2.6 and Corollary 5.2.7 below.

Lemma 5.2.6. *Let $Q, Q' \in \mathsf{PExec}(\mathcal{A})$ be such that $Q \leq_\flat Q'$. Then $E_Q \leq_\flat E_{Q'}$.*

Proof. Suppose we have $\langle \pi, a, \mu\rangle$ with $E_Q(\pi)(a, \mu) \neq 0$. We will prove that $E_Q(\pi)(a, \mu) = E_{Q'}(\pi)(a, \mu)$.

Since $E_Q(\pi)(a, \mu) \neq 0$, it follows from the definition of E_Q that $Q(\pi) \neq 0$. Since $Q \leq_\flat Q'$, this implies $Q'(\pi) = Q(\pi) \neq 0$. Now take any $s \in \mathit{supp}(\mu)$. We have

$$\begin{aligned} Q(\pi a \mu s) &= \mathbf{Q}_{E_Q}(\pi a \mu s) \\ &= \mathbf{Q}_{E_Q}(\pi) \cdot E_Q(\pi)(a, \mu) \cdot \mu(s) = Q(\pi) \cdot E_Q(\pi)(a, \mu) \cdot \mu(s) \neq 0, \end{aligned}$$

therefore $Q'(\pi a \mu s) = Q(\pi a \mu s) \neq 0$. Finally, by the definitions of E_Q and $E_{Q'}$, we have

$$E_Q(\pi)(a, \mu) = \frac{Q(\pi a \mu s)}{Q(\pi) \cdot \mu(s)} = \frac{Q'(\pi a \mu s)}{Q'(\pi) \cdot \mu(s)} = E_{Q'}(\pi)(a, \mu).$$

□

Corollary 5.2.7. *Let $\mathcal{D}$ be a directed subset of $\mathsf{PExec}(\mathcal{A})$ and let $\mathbf{Q}^<(\mathcal{D})$ denote the set $\{E_Q \mid Q \in \mathcal{D}\}$, where E_Q is constructed from Q as in Section 4.1. Then $\mathbf{Q}^<(\mathcal{D})$ is a directed subset of $\mathsf{Adv}(\mathcal{A})$ and $\mathbf{Q}_{\bigvee \mathbf{Q}^<(\mathcal{D})} = \bigvee(\mathcal{D})$.*

Proof. Since $\mathcal{D}$ is directed and the canonical inverse operation is monotone (Lemma 5.2.6), it is immediate that $\mathbf{Q}^<(\mathcal{D})$ is directed. The second claim then follows from continuity of $\mathbf{Q}$ (Theorem 5.2.5). □

Intuitively, a canonical inverse E_Q always halts after a path π that is not reachable. Therefore, by restricting our attention to canonical inverses, we ignore all the "noise", and thus inconsistencies, at unreachable paths. This in fact yields an isomorphism between CPO's.

Lemma 5.2.8. *The map* $E_- : \mathsf{PExec}(\mathcal{A}) \to \mathbf{Q}^<(\mathsf{PExec}(\mathcal{A}))$ *is a set isomorphism with inverse* $\mathbf{Q}$.

Proof. By Lemma 4.1.1, we know that $Q = \mathbf{Q}_{E_Q}$ for all $Q \in \mathsf{PExec}(\mathcal{A})$. Equivalently, $E_- : \mathsf{PExec}(\mathcal{A}) \to \mathsf{Adv}(\mathcal{A})$ is one-to-one. It follows immediately that $E_- : \mathsf{PExec}(\mathcal{A}) \to \mathbf{Q}^<(\mathsf{PExec}(\mathcal{A}))$ is a set isomorphism with inverse $\mathbf{Q}$. □

Lemma 5.2.9. *The poset* $\langle \mathbf{Q}^<(\mathsf{PExec}(\mathcal{A})), \leq_\flat \rangle$ *is a sub-CPO of* $\langle \mathsf{Adv}(\mathcal{A}), \leq_\flat \rangle$.

Proof. It is easy to check that E_- takes the bottom element of $\mathsf{PExec}(\mathcal{A})$ to the everywhere-0 adversary. Therefore $\langle \mathbf{Q}^<(\mathsf{PExec}(\mathcal{A})), \leq_\flat \rangle$ has the same bottom element as $\langle \mathsf{Adv}(\mathcal{A}), \leq_\flat \rangle$.

Let $\mathcal{D} \subseteq \mathbf{Q}^<(\mathsf{PExec}(\mathcal{A}))$ be directed. Since $\mathbf{Q}$ is monotone, the image of $\mathcal{D}$ under $\mathbf{Q}$ is again directed and has a join. Let Q denote that join. We claim that E_Q is the join of $\mathcal{D}$ in $\mathsf{Adv}(\mathcal{A})$.

Let $E' \in \mathcal{D}$ be given. Then $\mathbf{Q}_{E'} \leq_\flat Q$. By Lemmas 5.2.6 and 5.2.8, we have $E' = E_{\mathbf{Q}_{E'}} \leq_\flat E_Q$.

Now let $\overline{E}$ be an upperbound of $\mathcal{D}$ in $\mathsf{Adv}(\mathcal{A})$. By monotonicity of $\mathbf{Q}$, we have that $\mathbf{Q}_{\overline{E}}$ is an upperbound of the image of $\mathcal{D}$ under $\mathbf{Q}$, thus $Q \leq_\flat \mathbf{Q}_{\overline{E}}$. Again by Lemmas 5.2.6 and 5.2.8, we have $E_Q \leq_\flat E_{\mathbf{Q}_{\overline{E}}} = \overline{E}$. This completes our proof that E_Q is the join of $\mathcal{D}$. Therefore $\mathbf{Q}^<(\mathsf{PExec}(\mathcal{A}))$ is closed under directed joins and these joins coincide with those in $\mathsf{Adv}(\mathcal{A})$. □

Proposition 5.2.10. *The CPOs* $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat \rangle$ *and* $\langle \mathbf{Q}^<(\mathsf{PExec}(\mathcal{A})), \leq_\flat \rangle$ *are isomorphic via* $\mathbf{Q}$ *and* E_-.

Proof. By Theorem 5.2.5 and Lemmas 5.2.8 and 5.2.9, it suffices to show that E_- is strictly continuous. In the proof of Lemma 5.2.9, we saw that E_- preserves bottom element and, by Lemma 5.2.6, E_- is monotone. Thus it remains to show E_- commutes with directed joins.

Let $\mathcal{D} \subseteq \mathsf{PExec}(\mathcal{A})$ be directed. By the proof of Lemma 5.2.9, we know that $\bigvee \mathbf{Q}^<(\mathcal{D}) = E_Q$, where Q is the join of the image of $\mathbf{Q}^<(\mathcal{D})$ under $\mathbf{Q}$. Since $\mathbf{Q}$ is a bijection, this image is precisely $\mathcal{D}$. Thus

$$E_{\bigvee \mathcal{D}} = E_Q = \bigvee \mathbf{Q}^<(\mathcal{D}) = \bigvee \{E_Q \mid Q \in \mathcal{D}\}.$$

□

We now return to the discussion of compact elements in $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat \rangle$. By virtue of Lemma 5.2.9 and Proposition 5.2.10, $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat \rangle$ is isomorphic to a sub-CPO of $\langle \mathsf{Adv}(\mathcal{A}), \leq_\flat \rangle$. It is easy to see that, for every $Q \in \mathsf{PExec}(\mathcal{A})$,

$$\begin{aligned} &E_Q \text{ compact in } \langle \mathsf{Adv}(\mathcal{A}), \leq_\flat \rangle \Rightarrow E_Q \text{ compact in } \langle \mathbf{Q}^<(\mathsf{PExec}(\mathcal{A})), \leq_\flat \rangle \\ \Leftrightarrow\ &Q \text{ compact in } \langle \mathsf{PExec}(\mathcal{A}), \leq_\flat \rangle. \end{aligned}$$

In general, a compact element of a sub-CPO is not necessarily compact in the original CPO. (Given any non-compact element, we can take the sub-CPO consisting of only that element and $\bot$.) In our case, however, we do have the other implication. To prove this, we need the following auxiliary lemma.

Lemma 5.2.11. *For every $E \in \mathsf{Adv}(\mathcal{A})$, we have $E_{\mathbf{Q}_E} \leq_\flat E$.*

Proof. Let $\langle \pi, a, \mu \rangle$ be given such that $E_{\mathbf{Q}_E}(\pi)(a, \mu) \neq 0$. By the definition of E_-, we know that $\mathbf{Q}_E(\pi) \neq 0$ and

$$E_{\mathbf{Q}_E}(\pi)(a, \mu) = \frac{\mathbf{Q}_E(\pi a \mu s)}{\mathbf{Q}_E(\pi) \cdot \mu(s)},$$

where s is any state in $\mathsf{Supp}(\mu)$. Therefore, by the definition of $\mathbf{Q}_E(\pi a \mu s)$, we can conclude that $E(\pi)(a, \mu) = E_{\mathbf{Q}_E}(\pi)(a, \mu)$. □

Lemma 5.2.12. *Let $Q \in \mathsf{PExec}(\mathcal{A})$ be given and assume that Q is compact in $\mathsf{PExec}(\mathcal{A})$. Then E_Q is compact in $\mathsf{Adv}(\mathcal{A})$.*

Proof. Let $\mathcal{D} \subseteq \mathsf{Adv}(\mathcal{A})$ be directed and assume that $E_Q \leq_\flat \bigvee \mathcal{D}$ (join taken in $\mathsf{Adv}(\mathcal{A})$). By Proposition 5.2.10 and continuity of $\mathbf{Q}$, we have

$$Q = \mathbf{Q}_{E_Q} \leq_\flat \mathbf{Q}_{\bigvee \mathcal{D}} = \bigvee \{ \mathbf{Q}_E \mid E \in \mathcal{D} \}.$$

By compactness of Q, we may choose $E \in \mathcal{D}$ such that $Q \leq_\flat \mathbf{Q}_E$. Using Lemma 5.2.11 and monotonicity of E_-, we have $E_Q \leq_\flat E_{\mathbf{Q}_E} \leq_\flat E$. □

This completes our characterization of compact elements in $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat \rangle$.

Proposition 5.2.13. *A probabilistic execution Q is compact in $\mathsf{PExec}(\mathcal{A})$ if and only if E_Q is compact in $\mathsf{Adv}(\mathcal{A})$.*

Corollary 5.2.14. *Let $E \in \mathsf{Adv}(\mathcal{A})$ be given. If E is compact in $\mathsf{Adv}(\mathcal{A})$, then $\mathbf{Q}_E$ is compact in $\mathsf{PExec}(\mathcal{A})$.*

Proof. By Lemma 5.2.11, $E_{\mathbf{Q}_E} \leq_\flat E$. Using the definition of compactness in $\mathsf{Adv}(\mathcal{A})$, we infer that $E_{\mathbf{Q}_E}$ is also compact in $\mathsf{Adv}(\mathcal{A})$. By Proposition 5.2.13, $\mathbf{Q}_E$ is compact in $\mathsf{PExec}(\mathcal{A})$. □

Finally, we show that, for every probabilistic execution Q, the set of compact elements below Q is directed and the join is precisely Q.

Lemma 5.2.15. *Let Q be a probabilistic execution of $\mathcal{A}$. Let K_Q denote the set of compact elements below Q; that is,*

$$K_Q := \{Q' \mid Q' \text{ compact in } \mathsf{PExec}(\mathcal{A}) \text{ and } Q' \leq_\flat Q\}.$$

Then K_Q is directed and $Q = \bigvee K_Q$.

Proof. Let F be a finite subset of K_Q and let $Q' \in F$ be given. By Proposition 5.2.13, $E_{Q'}$ is compact in $\mathsf{Adv}(\mathcal{A})$. Moreover, $E_{Q'} \leq_\flat E_Q$. Therefore $\{E_{Q'} \mid Q' \in F\}$ is a finite set of compact elements of $\mathsf{Adv}(\mathcal{A})$ below E_Q. By Theorem 5.1.6, there is compact E (in $\mathsf{Adv}(\mathcal{A})$) below E_Q such that $E_{Q'} \leq_\flat E$ for every $Q' \in F$.

By Corollary 5.2.14, $\mathbf{Q}_E$ is compact in $\mathsf{PExec}(\mathcal{A})$. Since $E \leq_\flat E_Q$, we have $\mathbf{Q}_E \leq_\flat \mathbf{Q}_{E_Q} = Q$. Moreover, $Q' = \mathbf{Q}_{E_{Q'}} \leq_\flat \mathbf{Q}_E$ for every $Q' \in F$. Thus $\mathbf{Q}_E$ is an upperbound of F in K_Q.

It remains to show $Q = \bigvee K_Q$. Let E' be a compact element of $\mathsf{Adv}(\mathcal{A})$ with $E' \leq_\flat E_Q$. By Corollary 5.2.14, $\mathbf{Q}_{E'}$ is compact. Moreover, $\mathbf{Q}_{E'} \leq_\flat \mathbf{Q}_{E_Q} = Q$. Therefore $\mathbf{Q}_{E'} \in K_Q$. By Theorem 5.1.6, we have

$$Q = \mathbf{Q}_{E_Q} = \mathbf{Q}_{\bigvee K_{E_Q}} = \bigvee\{\mathbf{Q}_{E'} \mid E' \text{ compact and } E' \leq_\flat E_Q\} \leq_\flat \bigvee K_Q.$$

The other inequality is trivial. □

This completes the proof of the following theorem.

Theorem 5.2.16. *Given a countably branching PA $\mathcal{A}$, the poset $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat \rangle$ is an algebraic CPO.*

5.3 Trace Distributions

Finally, we treat the case of trace distributions. Define $\leq_\flat$ in exactly the same way: given $D_1, D_2 \in \mathsf{TrDist}(\mathcal{A})$, we say that $D_1 \leq_\flat D_2$ if for all $\beta \in \mathsf{Act}^{<\omega}$, $D_1(\beta) \neq 0$ implies $D_1(\beta) = D_2(\beta)$.

We wish to prove that the structure $\langle \mathsf{TrDist}(\mathcal{A}), \leq_\flat \rangle$ is a CPO. Notice, the trace distribution generated by the everywhere-0 adversary is the bottom element in $\langle \mathsf{TrDist}(\mathcal{A}), \leq_\flat \rangle$. It assigns 1 to the empty trace ϵ and 0 to every other $\beta \in \mathsf{Act}^{<\omega}$. It remains to show $\langle \mathsf{TrDist}(\mathcal{A}), \leq_\flat \rangle$ is closed under directed joins.

This claim does not hold in general, as illustrated in Example 4.3.1 of Section 4.3. However, as we show in this section, it is sufficient to assume that $\mathcal{A}$ is image finite (cf. Definition 3.0.3). We begin by examining properties of adversaries of an image finite automaton.

Image Finite Automata and Bounded Adversaries

Every adversary E for an image finite automaton $\mathcal{A}$ is *bounded* in the following sense: given any finite trace β and a small, positive error ε, it is possible to find a finite set $F \subseteq \mathsf{tr}^{-1}(\beta)$ such that $\mathbf{Q}_E$ assigns probability at most ε outside of F (i.e., the probability masses concentrate on F). The finite set F is a *uniform* bound, in that it depends only on β and ε, but not on the choice of adversary E. Existence of such a uniform bound is the key to avoiding counterexamples such as that in Example 4.3.1.

We now give a formal proof of this boundedness claim. Lemma 5.3.1 is a simple measure-theoretic observation: the event "executing trace β after following one of the paths in F" is strictly included in the event "following one of the paths in F". Therefore the first event has smaller measure. Notice this claim does not require image finiteness.

Lemma 5.3.1. *For all $F \subset \mathsf{Path}^{<\omega}(\mathcal{A})$ and $\beta \in \mathsf{Act}^{<\omega}$, we have*

$$\sum_{\pi \in F} \mathbf{Q}_E(\pi) \geq \sum_{\pi' \in \mathsf{Succ}(F,\beta)} \mathbf{Q}_E(\pi'),$$

provided both sums converge.

Proof. By induction on the length of β. If β is the empty sequence, then $\mathsf{Succ}(F, \beta) = F$ and the inequality trivially holds. Consider βa and let $\pi' \in \mathsf{Succ}(F, \beta a)$ be given. By definition of $\mathbf{Q}_E$, we have the following.

$$\begin{aligned}
&\sum_{\pi' \in \mathsf{Succ}(F,\beta a)} \mathbf{Q}_E(\pi') \\
&= \sum_{\pi'' \in \mathsf{Succ}(F,\beta)} \sum_{\mu : \mathsf{last}(\pi'') \xrightarrow{a} \mu} \sum_{s \in \mathsf{Supp}\,\mu} \mathbf{Q}_E(\pi'') \cdot E(\pi'')(a,\mu) \cdot \mu(s) \\
&= \sum_{\pi'' \in \mathsf{Succ}(F,\beta)} \mathbf{Q}_E(\pi'') \cdot \Big(\sum_{\mu : \mathsf{last}(\pi'') \xrightarrow{a} \mu} E(\pi'')(a,\mu) \cdot \sum_{s \in \mathsf{Supp}\,\mu} \mu(s) \Big) \\
&= \sum_{\pi'' \in \mathsf{Succ}(F,\beta)} \mathbf{Q}_E(\pi'') \cdot \Big(\sum_{\mu : \mathsf{last}(\pi'') \xrightarrow{a} \mu} E(\pi'')(a,\mu) \Big)
\end{aligned}$$

Since E is a sub-distribution, the inner sum is at most 1 and the whole expression is at most $\sum_{\pi'' \in \mathsf{Succ}(F,\beta)} \mathbf{Q}_E(\pi'')$. Applying the induction hypothesis, this is at most $\sum_{\pi \in F} \mathbf{Q}_E(\pi)$. □

Lemma 5.3.2 below says, given a particular path π and action a, we can find a finite set of paths F such that:

- every path in F extends π with an a-transition;
- every probabilistic execution $\mathbf{Q}_E$ must "concentrate" on F.

The key assumption here is image finiteness of $\mathcal{A}$, which dictates that only finitely many a-transitions are available at $\mathsf{last}(\pi)$. For each such transition $\langle \mathsf{last}(\pi), a, \mu \rangle$, we pick out a finite set of points in the support of μ so that μ "concentrates" on that set. This yields a finite set of paths for each μ and, taking a union over μ's, we obtain the finite set F. The idea is, no matter how an adversary E distributes probabilities among the μ's, the probability of landing in F is high due to the effect of each μ.

Lemma 5.3.2. *Assume $\mathcal{A}$ is image finite. Let $\varepsilon > 0$ be given. For all finite path π and action symbol a, there exists finite $F \subseteq \mathsf{Succ}(\pi, a)$ such that for all adversary E, $\sum_{\pi' \in \mathsf{Succ}(\pi,a) \setminus F} \mathbf{Q}_E(\pi') \leq \varepsilon$.*

Proof. Since $\mathcal{A}$ is image finite, there are finitely many μ's such that $\mathsf{last}(\pi) \xrightarrow{a} \mu$. Call them $\mu_0, \ldots, \mu_{n-1}$. For each $0 \leq i \leq n-1$, choose a finite subset $F_i \subseteq \mathsf{Supp}(\mu_i)$ such that

$$\sum_{s \in \mathsf{Supp}(\mu_i) \setminus F_i} \mu_i(s) \leq \frac{\varepsilon}{n}.$$

Define F to be $\bigcup_{0 \leq i \leq n-1} \{\pi a \mu_i s \mid s \in F_i\}$. Clearly F is finite. For any adversary E, we have

$$\begin{aligned}
&\sum_{\pi' \in \mathsf{Succ}(\pi,a) \setminus F} \mathbf{Q}_E(\pi') \\
&= \sum_{i=0}^{n-1} \sum_{s \in \mathsf{Supp}(\mu_i) \setminus F_i} \mathbf{Q}_E(\pi) \cdot E(\pi)(a, \mu_i) \cdot \mu_i(s) \\
&\leq \sum_{i=0}^{n-1} \sum_{s \in \mathsf{Supp}(\mu_i) \setminus F_i} \mu_i(s) && \mathbf{Q}_E(\pi) \leq 1; E(\pi)(a, \mu_i) \leq 1 \\
&\leq n \cdot \frac{\varepsilon}{n} = \varepsilon.
\end{aligned}$$

□

Lemma 5.3.2 can be thought of as boundedness for a single step. Now we generalize it to finitely many steps.

Lemma 5.3.3. *Assume $\mathcal{A}$ is image finite. Let $\varepsilon > 0$ and $\beta \in \mathsf{Act}^{<\omega}$ be given. There exists finite $F_\beta \subseteq \mathsf{tr}^{-1}(\beta)$ such that for all adversaries E,*

$$\sum_{\pi \in \mathsf{tr}^{-1}(\beta) \setminus F_\beta} \mathbf{Q}_E(\pi) \leq \varepsilon.$$

Proof. We proceed by induction on the length of β. If β is the empty sequence, then take F_β to be the singleton $\{s^0\}$.

Consider a finite trace βa and assume the induction hypothesis holds for β. Choose finite F_β such that for all E, $d_E := \sum_{\pi \in \mathsf{tr}^{-1}(\beta) \setminus F_\beta} \mathbf{Q}_E(\pi) \leq \frac{\varepsilon}{2}$. By Lemma 5.3.1, we have for all E,

$$\sum_{\pi' \in \mathsf{Succ}(\mathsf{tr}^{-1}(\beta) \setminus F_\beta, a)} \mathbf{Q}_E(\pi') \leq d_E \leq \frac{\varepsilon}{2}.$$

If F_β is empty, then

$$\begin{aligned}
\sum_{\pi' \in \mathsf{tr}^{-1}(\beta a) \setminus \emptyset} \mathbf{Q}_E(\pi') &= \sum_{\pi' \in \mathsf{tr}^{-1}(\beta a)} \mathbf{Q}_E(\pi') \\
&= \sum_{\pi' \in \mathsf{Succ}(\mathsf{tr}^{-1}(\beta), a)} \mathbf{Q}_E(\pi') \\
&= \sum_{\pi' \in \mathsf{Succ}(\mathsf{tr}^{-1}(\beta) \setminus F_\beta, a)} \mathbf{Q}_E(\pi') \\
&\leq \frac{\varepsilon}{2} \leq \varepsilon.
\end{aligned}$$

Therefore we may set $F_{\beta a}$ to be $\emptyset$.

Otherwise, let $\pi_0, \ldots, \pi_{n-1}$ be an enumeration of F_β and let $0 \leq i \leq n-1$ be given. By Lemma 5.3.2, we may choose $F_i \subseteq \mathsf{Succ}(\pi_i, a)$ such that for all E,

$$c_{E,i} := \sum_{\pi' \in \mathsf{Succ}(\pi_i, a) \setminus F_i} \mathbf{Q}_E(\pi') \leq \frac{\varepsilon}{2n}.$$

Let F be $\bigcup_{0 \leq i \leq n-1} F_i$. We have for all E,

$$\begin{aligned}
&\sum_{\pi \in \mathsf{tr}^{-1}(\beta a) \setminus F} \mathbf{Q}_E(\pi) \\
&= \sum_{0 \leq i \leq n-1} \sum_{\pi' \in \mathsf{Succ}(\pi_i, a) \setminus F_i} \mathbf{Q}_E(\pi') + \sum_{\pi' \in \mathsf{Succ}(\mathsf{tr}^{-1}(\beta) \setminus F_\beta, a)} \mathbf{Q}_E(\pi') \\
&\leq (\sum_{0 \leq i \leq n-1} c_{E,i}) + d_E \\
&\leq n \cdot \frac{\varepsilon}{2n} + \frac{\varepsilon}{2} = \varepsilon.
\end{aligned}$$

□

Directed Joins

For the rest of this section, we assume that $\mathcal{A}$ is image finite. First we show the join of an ω-chain of trace distributions is again a trace distribution. Let $\{E_i \mid i \in \mathbb{N}\}$ be a sequence of adversaries for $\mathcal{A}$ such that the set $\mathcal{C} := \{\mathbf{D}_{E_i} \mid i \in \mathbb{N}\}$ forms a chain. We need to find an adversary E such that $\mathbf{D}_E = \bigvee \mathcal{C}$, where

$\bigvee \mathcal{C}$ is the pointwise join. For convenience, let D_i denote $\mathbf{D}_{E_i}$ and let $\widehat{D}$ denote $\bigvee \mathcal{C}$.

Let $\{\pi_n \mid n \in \mathbb{N}\}$ be an enumeration of $\mathsf{Path}^{<\omega}(\mathcal{A})$. We apply the construction of Section 4.3 to $\{E_i \mid i \in \mathbb{N}\}$ and $\{\pi_n \mid n \in \mathbb{N}\}$ to obtain a sequence $\{\{E_j^n \mid j \in \mathbb{N}\}\}_{n\in\mathbb{N}}$ of sequences of adversaries for $\mathcal{A}$ and $Q \in \mathsf{PExec}(\mathcal{A})$. We claim that the trace distribution associated with Q is precisely $\widehat{D}$, thus any adversary E inducing Q also induces $\widehat{D}$.

Lemma 5.3.4. *For all $\beta \in \mathsf{Act}^{<\omega}$, $\mathsf{tr}(Q)(\beta) = \sum_{\pi\in\mathsf{tr}^{-1}(\beta)} Q(\pi) \leq \widehat{D}(\beta)$.*

Proof. Let $\beta \in \mathsf{Act}^{<\omega}$ be given. Let S be the set of n such that $\mathsf{tr}(\pi_n) = \beta$. It suffices to prove for all finite $Y \subseteq S$, $\sum_{n\in Y} Q(\pi_n) \leq \widehat{D}(\beta)$.

Let $N := \max(Y)$. By definition of Q and Corollary 4.3.2, we have

$$Q(\pi_n) = \lim_{j\in\mathbb{N}} \mathbf{Q}_{E_j^{n+1}}(\pi_n) = \lim_{j\in\mathbb{N}} \mathbf{Q}_{E_j^{N+1}}(\pi_n).$$

Thus, moving the finite sum into the limit, we have

$$\sum_{n\in Y} Q(\pi_n) = \sum_{n\in Y} \lim_{j\in\mathbb{N}} \mathbf{Q}_{E_j^{N+1}}(\pi_n) = \lim_{j\in\mathbb{N}} \sum_{n\in Y} \mathbf{Q}_{E_j^{N+1}}(\pi_n).$$

For each $j \in \mathbb{N}$, we have $\sum_{n\in Y} \mathbf{Q}_{E_j^{N+1}}(\pi_n) \leq \mathbf{D}_{E_j^{N+1}}(\beta) \leq \widehat{D}(\beta)$, hence the limit is also below $\widehat{D}(\beta)$. □

Lemma 5.3.5. *For all $\beta \in \mathsf{Act}^{<\omega}$, $\mathsf{tr}(Q)(\beta) = \sum_{\pi\in\mathsf{tr}^{-1}(\beta)} Q(\pi) \geq \widehat{D}(\beta)$.*

Proof. Let $\beta \in \mathsf{Act}^{<\omega}$ be given. Without loss of generality, assume that $\widehat{D}(\beta) \neq 0$. It suffices to show, for arbitrary $0 < \varepsilon < \widehat{D}(\beta)$, $\sum_{\pi\in\mathsf{tr}^{-1}(\beta)} Q(\pi) \geq \widehat{D}(\beta) - \varepsilon$. Let such ε be given. By Lemma 5.3.3, choose finite $F \subseteq \mathsf{tr}^{-1}(\beta)$ such that for all $i \in \mathbb{N}$, $D_i(\beta) - \sum_{\pi\in F} \mathbf{Q}_{E_i}(\pi) \leq \varepsilon$.

Clearly, $\sum_{\pi\in\mathsf{tr}^{-1}(\beta)} Q(\pi) \geq \sum_{\pi\in F} Q(\pi)$. We will prove that the latter is greater than or equal to $\widehat{D}(\beta) - \varepsilon$. Since F is finite, we may choose $N \in \mathbb{N}$ such that $F \subseteq \{\pi_0, \ldots, \pi_N\}$. Now we have

$$\begin{aligned}
\sum_{\pi\in F} Q(\pi) = \sum_{\{n \mid \pi_n\in F\}} Q(\pi_n) &= \sum_{\{n \mid \pi_n\in F\}} \lim_{j\in\mathbb{N}} \mathbf{Q}_{E_j^{n+1}}(\pi_n) \\
&= \sum_{\{n \mid \pi_n\in F\}} \lim_{j\in\mathbb{N}} \mathbf{Q}_{E_j^{N+1}}(\pi_n) && \text{Lemma 4.3.1} \\
&= \lim_{j\in\mathbb{N}} \sum_{\{n \mid \pi_n\in F\}} \mathbf{Q}_{E_j^{N+1}}(\pi_n) && F \text{ finite} \\
&\geq \lim_{j\in\mathbb{N}} (D_{\mathsf{index}(E_j^{N+1})}(\beta) - \varepsilon) && \text{choice of } F \\
&= (\lim_{j\in\mathbb{N}} D_{\mathsf{index}(E_j^{N+1})}(\beta)) - \varepsilon \\
&= \widehat{D}(\beta) - \varepsilon && \mathcal{C} \text{ increasing chain}
\end{aligned}$$

□

Corollary 5.3.6. *For all* $\beta \in \mathsf{Act}^{<\omega}$, $\mathsf{tr}(Q)(\beta) = \sum_{\pi \in \mathsf{tr}^{-1}(\beta)} Q(\pi) = \widehat{D}(\beta)$.

The following proposition summarizes the results we have obtained so far.

Proposition 5.3.7. *Let* $\mathcal{A}$ *be an image finite PA and let* $\mathcal{C}$ *be an increasing* ω*-chain of trace distributions. Then* $\bigvee \mathcal{C}$ *is also a trace distribution of* $\mathcal{A}$.

Theorem 5.3.8. *Let* $\mathcal{A}$ *be an image finite PA and let* $\mathcal{D}$ *be an arbitrary directed subset of* $\mathsf{TrDist}(\mathcal{A})$. *Then* $\bigvee \mathcal{D}$ *is also a trace distribution of* $\mathcal{A}$.

Proof. By Proposition 5.3.7 and Lemma 2.0.5. □

Corollary 5.3.9. *Given an image finite PA* $\mathcal{A}$, $\mathsf{TrDist}(\mathcal{A})$ *is a CPO whose bottom element is generated by the everywhere-*0 *adversary.*

Compact Elements in $\mathsf{TrDist}(\mathcal{A})$

Next we try to characterize compact elements in $\mathsf{TrDist}(\mathcal{A})$. Recall that the trace function $\mathsf{tr} : \mathsf{Path}^{<\omega}(\mathcal{A}) \to \mathsf{Act}^{<\omega}$ can be lifted to a function $\mathsf{tr} : \mathsf{PExec}(\mathcal{A}) \to \mathsf{TrDist}(\mathcal{A})$ (Definition 3.2.1). The trace distribution function $\mathbf{D} : \mathsf{Adv}(\mathcal{A}) \to \mathsf{TrDist}(\mathcal{A})$ is simply the composition of the following:

$$\mathsf{Adv}(\mathcal{A}) \xrightarrow{\mathbf{Q}} \mathsf{PExec}(\mathcal{A}) \xrightarrow{\mathsf{tr}} \mathsf{TrDist}(\mathcal{A}).$$

As we saw in Section 5.2, the CPO $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat \rangle$ is isomorphic to a sub-CPO of $\langle \mathsf{PExec}(\mathcal{A}), \leq_\flat \rangle$ via the continuous function $\mathbf{Q}$ and its right inverse E_-. Therefore, the compact elements in $\mathsf{PExec}(\mathcal{A})$ are essentially given by those in $\mathsf{Adv}(\mathcal{A})$. Unfortunately, this nice property no longer holds when we move from $\mathsf{PExec}(\mathcal{A})$ to $\mathsf{TrDist}(\mathcal{A})$.

Example 5.3.1. Consider the automaton $\mathcal{A}$ in Example 5.2.1. We add the following loops to the transition relation:

$$\{\langle s_i, b, \mathsf{Dirac}(s_i)\rangle \mid i \in \mathbb{N} \text{ and } i \neq 0\}.$$

Let E be the adversary that

- first schedules the transition $s^0 \xrightarrow{a} \mu$ with probability 1;
- then, at each $s^0 a \mu s_i$, schedules $s_i \xrightarrow{b} \mathsf{Dirac}(s_i)$ with probability 1;
- halts after the b-loop.

Clearly, E is not compact in $\mathsf{Adv}(\mathcal{A})$. However, $\mathbf{D}_E$ is compact in $\mathsf{TrDist}(\mathcal{A})$, because it assigns 1 to the three traces $\{\epsilon, a, ab\}$ and 0 to all others.

Example 5.3.1 suggests that the order structures on $\mathsf{PExec}(\mathcal{A})$ and $\mathsf{TrDist}(\mathcal{A})$, respectively, do not correspond closely via the map tr. In fact, tr is not monotone (Example 5.3.2), although it does preserve directed joins (Proposition 5.3.10). It is also trivial to check that tr preserves bottom.

Example 5.3.2. Consider an automaton $\mathcal{A}$ with state space $\{s^0, s_1, s_2\}$ and two transitions $\langle s^0, a, \mathsf{Dirac}(s_1)\rangle$ and $\langle s^0, a, \mathsf{Dirac}(s_2)\rangle$. Let E_1 be the adversary that schedules $\langle s^0, a, \mathsf{Dirac}(s_1)\rangle$ with probability $\frac{1}{2}$ and halts otherwise. Let E_2 be the adversary that schedules each of the two transitions with probability $\frac{1}{2}$. Clearly, $\mathbf{Q}_{E_1} \leq_\flat \mathbf{Q}_{E_2}$. However, it is not the case that $\mathbf{D}_{E_1} \leq_\flat \mathbf{D}_{E_2}$, since $\mathbf{D}_{E_1}(a) = \frac{1}{2} \neq 1 = \mathbf{D}_{E_2}(a)$. Therefore tr is not monotone.

Proposition 5.3.10. *Let $\mathcal{D}$ be a directed subset of $\mathsf{PExec}(A)$. Given any $\beta \in \mathsf{Act}^{<\omega}$,*

$$\mathsf{tr}(\bigvee \mathcal{D})(\beta) = \bigvee_{Q \in \mathcal{D}} \mathsf{tr}(Q)(\beta).$$

Proof. By definition,

$$\mathsf{tr}(\bigvee \mathcal{D})(\beta) = \sum_{\pi \in \mathsf{tr}^{-1}(\beta)} (\bigvee \mathcal{D})(\pi) = \sum_{\pi \in \mathsf{tr}^{-1}(\beta)} \bigvee_{Q \in \mathcal{D}} Q(\pi).$$

Since the set of finite paths in $\mathcal{A}$ is countable and $\mathcal{D}$ is directed, we can view $\{\{Q(\pi) \mid \pi \in \mathsf{tr}^{-1}(\beta)\} \mid Q \in \mathcal{D}\}$ as a directed subset of $[0,1]^{\mathbb{N}}$. Moreover, each Q in $\mathcal{D}$ is a probabilistic execution, hence $\sum_{\pi \in \mathsf{tr}^{-1}(\beta)} Q(\pi)$ converges to a value in $[0,1]$. Therefore, we can apply Theorem 2.0.6 to conclude:

$$\sum_{\pi \in \mathsf{tr}^{-1}(\beta)} \bigvee_{Q \in \mathcal{D}} Q(\pi) = \bigvee_{Q \in \mathcal{D}} \sum_{\pi \in \mathsf{tr}^{-1}(\beta)} Q(\pi) = \bigvee_{Q \in \mathcal{D}} \mathsf{tr}(Q)(\beta).$$

□

These observations dictate that we must start afresh in characterizing compact elements of $\mathsf{TrDist}(\mathcal{A})$. This leads to the notions of finite adversaries introduced in Section 3.3. We now give a formal proof that finite trace distributions (i.e., those induced by finite adversaries) are precisely the compact elements in $\mathsf{TrDist}(\mathcal{A})$. Moreover, $\mathsf{TrDist}(\mathcal{A})$ forms an algebraic CPO structure under $\leq_\flat$.

Essentially, a trace distribution $\mathbf{D}_E$ is finite in the sense of Section 3.3 if it assigns zero probability to all but a finite number of traces. Lemma 5.3.11 below says that all finite trace distributions are compact in the order-theoretic sense. Lemma 5.3.12 is the converse.

Lemma 5.3.11. *Let $\mathbf{D}_E$ be a finite trace distribution and let $\mathcal{D}$ be a directed set of trace distributions such that $\mathbf{D}_E \leq_\flat \bigvee \mathcal{D}$. Then there exists adversary E' with $\mathbf{D}_{E'} \in \mathcal{D}$ and $\mathbf{D}_E \leq_\flat \mathbf{D}_{E'}$.*

Proof. Let F denote the set of traces $\{\beta \in \mathsf{Act}^{<\omega} \mid \mathbf{D}_E(\beta) \neq 0\}$. By virtue of Proposition 3.3.1, F is a subset of $(\mathsf{Act}_l)^k$ for some k, l in $\mathbb{N}$. Hence F is finite.

Since $\mathbf{D}_E \leq_\flat \bigvee \mathcal{D}$, we may choose, for each $\beta \in F$, an adversary E_β with $\mathbf{D}_{E_\beta} \in \mathcal{D}$ and $\mathbf{D}_{E_\beta}(\beta) = \mathbf{D}_E(\beta)$. Since $\mathcal{D}$ is directed and F is finite, we may choose E' such that $\mathbf{D}_{E'}$ is in $\mathcal{D}$ and is an upperbound of $\{\mathbf{D}_{E_\beta} \mid \beta \in F\}$. Clearly $\mathbf{D}_E \leq_\flat \mathbf{D}_{E'}$. □

Lemma 5.3.12. *Let E be a adversary of $\mathcal{A}$ with $\mathbf{D}_E$ not finite. There exists a directed set $\mathcal{D}$ of trace distributions of $\mathcal{A}$ such that $\mathbf{D}_E = \bigvee \mathcal{D}$ and yet $\mathbf{D}_{E'} <_\flat \mathbf{D}_E$ for all $\mathbf{D}_{E'} \in \mathcal{D}$.*

Proof. Let $\{\beta_0, \beta_1, \ldots\}$ be a prefix-preserving enumeration of $\mathsf{Act}^{<\omega}$. That is, if β_m is a prefix of β_n, then $m \leq n$. This is always possible for the set of finite words over a countable alphabet. For each $n \in \mathbb{N}$, construct a adversary E_n as follows: for all π, a and μ,

- $E_n(\pi)(a, \mu) = E(\pi)(a, \mu)$ if $\mathsf{tr}(\pi)a$ is in $\beta_0, \ldots, \beta_n$;
- $E_n(\pi)(a, \mu) = 0$ otherwise.

Informally, each E_n makes the same decisions as E until it reaches a trace not in $\beta_0, \ldots, \beta_n$, at which point it halts. Since $\{\beta_n \mid n \in \mathbb{N}\}$ preserves prefix, it is easy to verify that $\{\mathbf{D}_{E_n} \mid n \in \mathbb{N}\}$ satisfies these two conditions:

- for all $m \leq n$, $\mathbf{D}_{E_n}(\beta_m) = \mathbf{D}_E(\beta_m)$;
- for all $m > n$, $\mathbf{D}_{E_n}(\beta_m) = 0$.

Clearly, each $\mathbf{D}_{E_n}$ is finite. Since $\mathbf{D}_E$ is infinite, we have $\mathbf{D}_{E_n} <_\flat \mathbf{D}_E$ for all $n \in \mathbb{N}$. Also $\{\mathbf{D}_{E_n} \mid n \in \mathbb{N}\}$ is an increasing chain whose limit is precisely $\mathbf{D}_E$, hence $\mathbf{D}_E$ must not be compact. □

This completes the proof of the following proposition.

Proposition 5.3.13. *For every $D \in \mathsf{TrDist}(\mathcal{A})$, D is compact in $\langle \mathsf{TrDist}(\mathcal{A}), \leq_\flat \rangle$ if and only if D is finite in the sense of Section 3.3.*

To show that $\langle \mathsf{TrDist}(\mathcal{A}), \leq_\flat \rangle$ is algebraic, we need one more lemma. Namely, every element in $\mathsf{TrDist}(\mathcal{A})$ is the (directed) join of all compact elements below it.

Lemma 5.3.14. *Let E be a adversary of $\mathcal{A}$. Let K_E denote the set of compact elements below $\mathbf{D}_E$; that is,*

$$K_E := \{\mathbf{D}_{E'} \mid \mathbf{D}_{E'} \text{ finite and } \mathbf{D}_{E'} \leq_\flat \mathbf{D}_E\}.$$

Then K_E is directed and $\mathbf{D}_E = \bigvee K_E$.

Proof. Again we make use of the prefix-preserving enumeration $\{\beta_n \mid n \in \mathbb{N}\}$. Take $\{E_n \mid n \in \mathbb{N}\}$ as in the proof of Lemma 5.3.12. Given a finite subset F of K_E, we can find $N \in \mathbb{N}$ such that for all $\mathbf{D}_{E'} \in F$ and $n \geq N$, $\mathbf{D}_{E'}(\beta_n) = 0$. This is because F is finite and each $\mathbf{D}_{E'}$ is finite. Then $\mathbf{D}_{E_N}$ is an upperbound of F. Moreover, $\mathbf{D}_{E_N}$ is finite, hence in K_E. This shows K_E is directed.

Finally, by the definition of $\leq_\flat$, we have for all n: $\bigvee K_E(\beta_n) = \mathbf{D}_{E_n}(\beta_n) = \mathbf{D}_E(\beta_n)$. □

Theorem 5.3.15. *Given an image finite PA $\mathcal{A}$, the structure $\langle \mathsf{TrDist}(\mathcal{A}), \leq_\flat \rangle$ is an algebraic CPO and the compact elements are precisely the finite trace distributions.*

Proof. By Corollary 5.3.9, Proposition 5.3.13 and Lemma 5.3.14. □

We end this section with a comparison between our development and that by Segala [Seg96]. In his proposal of the Approximation Induction Principle, trace distributions are ordered pointwise by the usual ordering on $\mathbb{R}$, rather than our flat ordering. Since the real numbers are complete, this alternative also gives rise to a CPO on $\mathsf{TrDist}(\mathcal{A})$, provided $\mathcal{A}$ is image finite. It also enjoys the pleasant property that the operator $\mathsf{tr} : \mathsf{PExec}(\mathcal{A}) \to \mathsf{TrDist}(\mathcal{A})$ is monotone (cf. Example 5.3.2). However, there is a major disadvantage: the resulting structure on $\mathsf{TrDist}(\mathcal{A})$ is not algebraic. This is illustrated in the following example.

Example 5.3.3. Consider an automaton $\mathcal{A}$ with state space $\{s^0, s_1\}$ and a single transition $\langle s^0, a, \mathsf{Dirac}(s_1)\rangle$. Let E be the adversary E that assigns probability 1 to that transition. Consider the sequence $E_0, E_1, \ldots$ of adversaries where each E_k chooses the a-transition with probability $1 - 2^{-k}$ and halts with probability 2^{-k}. Clearly, this infinite sequence converges monotonically to E under Segala's ordering; yet $E \neq E_k$ for all k. Therefore E is not a compact element. Similarly, one can show that every non-trivial trace distribution is *not* compact.

6

Metric Convergence

In Chapter 5, we imposed an order structure on $\mathsf{TrDist}(\mathcal{A})$ and proved that it is closed under directed limits. Those limits are of a very "discrete" character, thanks to our definition of the flat ordering. In the present chapter, we study limits of trace distributions in a different setting. Namely, we view each trace distribution of $\mathcal{A}$ as a point in the metric space $[0,1]^{\mathsf{Act}^{<\omega}}$, where the distance between two points is given by

$$\mathsf{dist}(\vec{u},\vec{v}) := \sup_{\beta \in \mathsf{Act}^{<\omega}} |u_\beta - v_\beta|.$$

Recall from Section 4.3 the construction of a probabilistic execution Q from any infinite sequence $\{E_i \mid i \in \mathbb{N}\}$ of adversaries. This construction is a key ingredient in our proof that $\langle \mathsf{TrDist}(\mathcal{A}), \leq_\flat \rangle$ is closed under directed limits (cf. Section 5.3). In particular, it is applied to a sequence of adversaries whose trace distributions form an increasing chain $\mathcal{C}$. Assuming image finiteness, the trace distribution associated with Q is shown to be the join of $\mathcal{C}$ (cf. Corollary 5.3.6).

It turns out that the same construction can be used to obtain a limiting probabilistic execution in the metric setting. It is applied in a very similar fashion: given a sequence of adversaries whose trace distributions converges to a point $\vec{u}$ in $[0,1]^{\mathsf{Act}^{<\omega}}$, we construct Q as in Section 4.3 and show that the trace distribution associated with Q is precisely $\vec{u}$. In light of Example 4.3.2, this result relies on the assumption that $\mathcal{A}$ is image finite. Therefore, we assume image finiteness throughout this chapter.

In Section 6.1, we show that $\mathsf{TrDist}(\mathcal{A},k,l)$ forms a closed set in $[0,1]^{\mathsf{Act}^{<\omega}}$. This fact will be used in Chapter 7 to establish the equivalence between trace distribution semantics and our finite testing semantics. In Section 6.2, we prove the analogous claim for $\mathsf{TrDist}(\mathcal{A},k,-)$.

6.1 Finite Breadth

Let $\{E_i \mid i \in \mathbb{N}\}$ be a sequence of adversaries and, for each $i \in \mathbb{N}$, let D_i denote the trace distribution $\mathbf{D}_{E_i}$. Since every D_i can be viewed as a point in $[0,1]^{\mathsf{Act}^{<\omega}}$, it makes sense to speak of convergence of $\{D_i \mid i \in \mathbb{N}\}$ with respect

to the metric dist.

Definition 6.1.1. We say that $\{E_i \mid i \in \mathbb{N}\}$ is a *trace convergent* sequence of adversaries whenever $\{D_i \mid i \in \mathbb{N}\}$ is a convergent sequence in the space $\langle [0,1]^{\mathsf{Act}^{<\omega}}, \mathsf{dist}\rangle$. That is, there exists $D \in [0,1]^{\mathsf{Act}^{<\omega}}$ such that

$$\forall \varepsilon \; \exists N \; \forall i \geq N \;\; \mathsf{dist}(D_i, D) \leq \varepsilon.$$

Expanding the definition of dist, this becomes

$$\forall \varepsilon \; \exists N \; \forall i \geq N \; \forall \beta \in \mathsf{Act}^{<\omega} \;\; |D_i(\beta) - D(\beta)| \leq \varepsilon.$$

The goal of this section is to show that, given a trace convergent sequence $\{E_i \mid i \in \mathbb{N}\} \subseteq \mathsf{TrDist}(\mathcal{A}, k, l)$, the limit point D is also a trace distribution in $\mathsf{TrDist}(\mathcal{A}, k, l)$. We do so by first constructing a probabilistic execution Q from $\{E_i \mid i \in \mathbb{N}\}$ using the procedure described in Section 4.3. Then we consider the adversary E_Q, as defined in Section 4.1. We will prove that E_Q is in $\mathsf{Adv}(\mathcal{A}, k, l)$ and $\mathbf{D}_E$ is the limit of $\{D_i \mid i \in \mathbb{N}\}$.

We need a rather technical lemma, which is a modification of Lemma 5.3.3. Very roughly, Lemma 5.3.3 says: given any finite trace β, it is possible to find a finite set F of paths with trace β so that every probabilistic execution is "concentrated" on F. In the following modified version, we restrict our attention to adversaries of finite breadth (i.e., those from $\mathsf{TrDist}(\mathcal{A}, -, l)$). This allows us to strengthen the conclusion of Lemma 5.3.3 to the existence of a uniform bound for all $\beta \in \mathsf{Act}^{\leq k}$.

Lemma 6.1.1. *Suppose $\mathcal{A}$ is image finite. For every $\varepsilon > 0$, there exists finite, non-empty $P_{k,\varepsilon} \subseteq \mathsf{Path}^{\leq k}(\mathcal{A})$ such that for all $E \in \mathsf{Adv}(\mathcal{A}, -, l)$ and for all $\beta \in \mathsf{Act}^{\leq k}$, $\sum_{\pi \in \mathsf{tr}^{-1}(\beta) \setminus P_{k,\varepsilon}} \mathbf{Q}_E(\pi) \leq \varepsilon$.*

Proof. We proceed by induction on k. For every ε, take $P_{0,\varepsilon}$ to be the singleton $\{s^0\}$. Now suppose the claim holds for k. Let $\varepsilon > 0$ be given and choose a finite, nonempty set $P_{k,\frac{\varepsilon}{2}}$ as stated. Let $m > 0$ be its cardinality. Consider the set

$$S := \bigcup_{|\pi|=k, \pi \in P_{k,\frac{\varepsilon}{2}}} \{\langle \mathsf{last}(\pi),\, a,\, \mu\rangle \mid \mathsf{last}(\pi) \xrightarrow{a} \mu \text{ and } a \in \mathsf{Act}_l\}.$$

Since $\mathcal{A}$ is image finite, S is a finite union of finite sets, hence also finite. If S is empty, set $P_{k+1,\varepsilon}$ to be $P_{k,\frac{\varepsilon}{2}}$. Otherwise, let $n > 0$ be the cardinality of S. For each μ occurring in S, choose a finite set $X_\mu \subseteq \mathsf{Supp}(\mu)$ such that

$$\sum_{s \in \mathsf{Supp}(\mu) \setminus X_\mu} \mu(s) \leq \frac{\varepsilon}{2mn}.$$

Then set $P_{k+1,\varepsilon}$ to be

$$P_{k,\frac{\varepsilon}{2}} \cup \{\pi a \mu s \mid \pi \in P_{k,\frac{\varepsilon}{2}} \text{ and } |\pi| = k \text{ and } \langle \mathsf{last}(\pi),\, a,\, \mu\rangle \in S \text{ and } s \in X_\mu\}.$$

We will prove that $P_{k+1,\varepsilon}$ satisfies the desired condition.

Let $E \in \mathsf{Adv}(\mathcal{A}, -, l)$ and $\beta \in \mathsf{Act}^{\leq k+1}$ be given. Notice, if β contains a symbol not in Act_l, then by Proposition 3.3.1 $\mathbf{Q}_E(\pi) = 0$ for all $\pi \in \mathsf{tr}^{-1}(\beta)$. Thus we may assume that $\beta \in (\mathsf{Act}_l)^{\leq k+1}$. Moreover, if β has length at most k, then $\mathsf{tr}^{-1}(\beta) \setminus P_{k+1,\varepsilon} = \mathsf{tr}^{-1}(\beta) \setminus P_{k,\frac{\varepsilon}{2}}$. This is because every path $\pi \in P_{k+1,\varepsilon} \setminus P_{k,\frac{\varepsilon}{2}}$ (if it exists) must have length $k+1$. Therefore, we have

$$\sum_{\pi \in \mathsf{tr}^{-1}(\beta) \setminus P_{k+1,\varepsilon}} \mathbf{Q}_E(\pi) = \sum_{\pi \in \mathsf{tr}^{-1}(\beta) \setminus P_{k,\frac{\varepsilon}{2}}} \mathbf{Q}_E(\pi) \leq \frac{\varepsilon}{2} \leq \varepsilon.$$

Note that the equality $\mathsf{tr}^{-1}(\beta) \setminus P_{k+1,\varepsilon} = \mathsf{tr}^{-1}(\beta) \setminus P_{k,\frac{\varepsilon}{2}}$ also holds when we have $S = \emptyset$. Therefore the same reasoning applies.

We now focus on the case in which $\beta \in (\mathsf{Act}_l)^{k+1}$ and S is non-empty. Suppose β is of the form $\beta' a$. By the choice of $P_{k,\frac{\varepsilon}{2}}$, it contains paths with length at most k, thus $P_{k,\frac{\varepsilon}{2}} \cap \mathsf{tr}^{-1}(\beta) = \emptyset$. We can then partition $Y := \mathsf{tr}^{-1}(\beta) \setminus P_{k+1,\varepsilon}$ into two sets:

$$Y_0 := \{\pi a \mu s \in Y \mid \pi \notin P_{k,\frac{\varepsilon}{2}} \text{ or } |\pi| \neq k \text{ or } \langle \mathsf{last}(\pi),\, a,\, \mu \rangle \notin S\},$$

$$Y_1 := \{\pi a \mu s \in Y \mid \pi \in P_{k,\frac{\varepsilon}{2}} \text{ and } |\pi| = k \text{ and } \langle \mathsf{last}(\pi),\, a,\, \mu \rangle \in S \text{ and } s \notin X_\mu\}.$$

It is easy to check that $Y_0 = \{\pi a \mu s \in Y \mid \pi \notin P_{k,\frac{\varepsilon}{2}}\}$. Then by Lemma 5.3.1 and the induction hypothesis, we have

$$\sum_{\pi \in Y_0} \mathbf{Q}_E(\pi) \leq \sum_{\pi' \in \mathsf{tr}^{-1}(\beta') \setminus P_{k,\frac{\varepsilon}{2}}} \mathbf{Q}_E(\pi') \leq \frac{\varepsilon}{2}.$$

On the other hand,

$$\begin{aligned}
\sum_{\pi \in Y_1} \mathbf{Q}_E(\pi) &= \sum_{\pi a \mu s \in Y_1} \mathbf{Q}_E(\pi) \cdot E(\pi)(a, \mu) \cdot \mu(s) \\
&\leq \sum_{\pi a \mu s \in Y_1} \mu(s) \\
&\leq \sum_{\pi \in P_{k,\frac{\varepsilon}{2}}} \sum_{\langle \mathsf{last}(\pi),\, a,\, \mu \rangle \in S} \sum_{s \in \mathsf{Supp}(\mu) \setminus X_\mu} \mu(s) \\
&\leq m \cdot n \cdot \sum_{s \in \mathsf{Supp}(\mu) \setminus X_\mu} \mu(s) \\
&\leq m \cdot n \cdot \frac{\varepsilon}{2mn} = \frac{\varepsilon}{2}.
\end{aligned}$$

Therefore,

$$\sum_{\pi \in \mathsf{tr}^{-1}(\beta) \setminus P_{k+1,\varepsilon}} \mathbf{Q}_E(\pi) = \sum_{\pi \in Y_0} \mathbf{Q}_E(\pi) + \sum_{\pi \in Y_1} \mathbf{Q}_E(\pi) \leq \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon.$$

□

We are now ready to prove that every trace convergent sequence in $\mathsf{TrDist}(\mathcal{A}, k, l)$ has a limiting trace distribution.

Proposition 6.1.2. *Let $\mathcal{A}$ be an image finite PA and let $k, l \in \mathbb{N}$ be given. Let $\{E_i \mid i \in \mathbb{N}\}$ be a sequence of trace convergent adversaries from $\mathsf{Adv}(\mathcal{A}, k, l)$ and write D_i for $\mathbf{D}_{E_i}$. Let Q be constructed as in Section 4.3 and let E denote E_Q (cf. Section 4.1). Then $\mathbf{D}_E$ is the limit of $\{D_i \mid i \in \mathbb{N}\}$ in the space $[0,1]^{\mathsf{Act}^{<\omega}}$. That is,*

$$\forall \varepsilon \; \exists N \; \forall i > N \; \forall \beta \in \mathsf{Act}^{<\omega} \;\; |D_i(\beta) - \mathbf{D}_E(\beta)| \leq \varepsilon.$$

Also, $E \in \mathsf{Adv}(\mathcal{A}, k, l)$.

Proof. By Proposition 3.3.1, we have $D_i(\beta) = 0 = \mathbf{D}_E(\beta)$ for all $\beta \notin (\mathsf{Act}_l)^{\leq k}$ and $i \in \mathbb{N}$. Hence we may focus on traces in $(\mathsf{Act}_l)^{\leq k}$. Let $\varepsilon > 0$ be given. Choose finite, non-empty $P_{k,\frac{\varepsilon}{3}}$ as in Lemma 6.1.1 and let $m := |P_{k,\frac{\varepsilon}{3}}|$. Moreover, by trace convergence of $\{E_i \mid i \in \mathbb{N}\}$, we may choose M_0 such that for all $i, j > M_0$, $\mathsf{dist}(D_i, D_j) < \frac{\varepsilon}{3}$.

Recall from Section 4.3 that we have an enumeration $\{\pi_n \mid n \in \mathbb{N}\}$ of $\mathsf{Path}^{<\omega}(\mathcal{A})$. Let $M := \max\{n \mid \pi_n \in P_{k,\frac{\varepsilon}{3}}\} + 1$. Then by Corollary 4.3.2, we have

$$\forall \pi \in P_{k,\frac{\varepsilon}{3}} \; \lim_{j \in \mathbb{N}} \mathbf{Q}_{E_j^M}(\pi) = \mathbf{Q}_E(\pi).$$

For each $\pi \in P_{k,\frac{\varepsilon}{3}}$, choose j_π such that

$$\forall j > j_\pi \;\; |\mathbf{Q}_{E_j^M}(\pi) - \mathbf{Q}_E(\pi)| < \frac{\varepsilon}{3m}.$$

Let L be the least number such that $L > \max\{j_\pi \mid \pi \in P_{k,\frac{\varepsilon}{3}}\}$ and $\mathsf{index}(E_L^M) > M_0$. Let $N := \mathsf{index} E_L^M$. Write Y_0 for $\mathsf{tr}^{-1}(\beta) \cap P_{k,\frac{\varepsilon}{3}}$ and Y_1 for $\mathsf{tr}^{-1}(\beta) \setminus P_{k,\frac{\varepsilon}{3}}$. Then for all $i > N$ and $\beta \in (\mathsf{Act}_l)^{\leq k}$,

$$\begin{aligned}
&|D_i(\beta) - \mathbf{D}_E(\beta)| \\
&\leq |D_i(\beta) - D_N(\beta)| + |D_N(\beta) - \mathbf{D}_E(\beta)| \\
&\leq \frac{\varepsilon}{3} + |\sum_{\pi \in \mathsf{tr}^{-1}(\beta)} \mathbf{Q}_{E_L^M}(\pi) - \sum_{\pi \in \mathsf{tr}^{-1}(\beta)} \mathbf{Q}_E(\pi)| \\
&\leq \frac{\varepsilon}{3} + |\sum_{\pi \in Y_0} \mathbf{Q}_{E_L^M}(\pi) - \sum_{\pi \in Y_0} \mathbf{Q}_E(\pi) + \sum_{\pi \in Y_1} \mathbf{Q}_{E_L^M}(\pi) - \sum_{\pi \in Y_1} \mathbf{Q}_E(\pi)| \\
&\leq \frac{\varepsilon}{3} + \sum_{\pi \in Y_0} |\mathbf{Q}_{E_L^M}(\pi) - \mathbf{Q}_E(\pi)| + |\sum_{\pi \in Y_1} \mathbf{Q}_{E_L^M}(\pi) - \sum_{\pi \in Y_1} \mathbf{Q}_E(\pi)| \\
&\leq \frac{\varepsilon}{3} + m \cdot \frac{\varepsilon}{3m} + \frac{\varepsilon}{3} = \varepsilon,
\end{aligned}$$

where:

- the second inequality follows from the fact that $\mathsf{index}(E_L^M) > M_0$; and

– the last inequality follows from the fact that $L > \max\{j_\pi \mid \pi \in P_{k,\frac{\varepsilon}{3}}\}$ and the choice of $P_{k,\frac{\varepsilon}{3}}$.

Finally, we prove that $E \in \mathsf{Adv}(\mathcal{A}, k, l)$. By the definition of E, $E(\pi)(a,\mu) \neq 0$ implies $Q(\pi) \neq 0$. Therefore $E(\pi)(a,\mu) \neq 0$ also implies $\mathbf{D}_E(\mathsf{tr}(\pi)a) \neq 0$. At the beginning of this proof, we saw that $\mathbf{D}_E(\beta) = 0$ whenever $\beta \notin (\mathsf{Act}_l)^{\leq k}$. Therefore, $E(\pi)(a,\mu) \neq 0$ implies that $|\pi| < k$ and $a \in \mathsf{Act}_l$. This completes our proof. □

Corollary 6.1.3. *Let $\mathcal{A}$ be image finite. For all $k, l \in \mathbb{N}$, the set $\mathsf{TrDist}(\mathcal{A}, k, l)$ is a closed subset of $\langle [0,1]^{\mathsf{Act}^{<\omega}}, \mathsf{dist}\rangle$.*

Next we prove the analogous result for induced probability distributions (as defined in Section 3.2).

Lemma 6.1.4. *Suppose $\mathcal{A}$ is image finite. Let $\{P_i \mid i \in \mathbb{N}\} \subseteq \{\mathbf{P}_{D,k} \mid D \in \mathsf{TrDist}(\mathcal{A}, k, l)\}$ be a convergent sequence in $\mathsf{Act}^{<\omega}$ with limit point P. Then P is a discrete distribution on $\mathsf{Act}^{<\omega}$.*

Proof. Clearly, $P[\beta] = 0$ for all $\beta \notin (\mathsf{Act}_l)^{\leq k}$. On the other hand, since $(\mathsf{Act}_l)^{\leq k}$ is a finite set, we have

$$\sum_{\beta \in (\mathsf{Act}_l)^{\leq k}} P[\beta] = \sum_{\beta \in (\mathsf{Act}_l)^{\leq k}} \lim_{i \in \mathbb{N}} P_i[\beta] = \lim_{i \in \mathbb{N}} \sum_{\beta \in (\mathsf{Act}_l)^{\leq k}} P_i[\beta] = 1.$$

□

Lemma 6.1.5. *Let $\mathcal{A}$ be image finite. Let $k, l \in \mathbb{N}$ and $\{P_i \mid i \in \mathbb{N}\} \subseteq \{\mathbf{P}_{D,k} \mid D \in \mathsf{TrDist}(\mathcal{A}, k, l)\}$ be given. Suppose $\{P_i \mid i \in \mathbb{N}\}$ is a convergent sequence in $\mathsf{Act}^{<\omega}$ with limit point P. For each i, choose D_i so that $P_i = \mathbf{P}_{D_i,k}$. Then $\{D_i \mid i \in \mathbb{N}\}$ is also a convergent sequence in $\mathsf{Act}^{<\omega}$. Moreover, $P = \mathbf{P}_{D,k}$, where D is the limit of $\{D_i \mid i \in \mathbb{N}\}$.*

Proof. Recall from Proposition 3.3.5 that, for each $i \in \mathbb{N}$ and $\beta \in (\mathsf{Act}_l)^{\leq k}$, we have

$$D_i(\beta) = \sum_{\beta \sqsubseteq \beta'; \beta' \in (\mathsf{Act}_l)^{\leq k}} P_i[\beta'].$$

Define D from P with the same formula. Notice this is a finite sum, therefore D is the limit of $\{D_i \mid i \in \mathbb{N}\}$. □

Corollary 6.1.6. *Let $\mathcal{A}$ be image finite. For all $k, l \in \mathbb{N}$, the set $\{\mathbf{P}_{D,k} \mid D \in \mathsf{TrDist}(\mathcal{A}, k, l)\}$ is also a closed subset of $[0,1]^{\mathsf{Act}^{<\omega}}$.*

Proof. By Corollary 6.1.3 and Lemma 6.1.5. □

6.2 Infinite Breadth

In the previous section, we showed that $\mathsf{TrDist}(\mathcal{A}, k, l)$ is a closed subset of the metric space $[0,1]^{\mathsf{Act}^{<\omega}}$, where $\mathsf{dist}(\vec{u}, \vec{v})$ is given by $\sup_{\beta \in \mathsf{Act}^{<\omega}} |u_\beta - v_\beta|$. The present section concerns the analogous result for $\mathsf{TrDist}(\mathcal{A}, k, -)$.

We use the same broad strategy: given a trace convergent sequence $\{E_i \mid i \in \mathbb{N}\}$ of adversaries, we construct a probabilistic execution Q as in Section 4.3 and prove that $\mathsf{tr}(Q)$ is the appropriate limit. However, in the case of $\mathsf{TrDist}(\mathcal{A}, k, -)$, the trace distributions in question may assign nonzero probability to infinitely many traces in $\mathsf{Act}^{<\omega}$. This means we are working in an infinite-dimensional space, where convergence properties are very different from those in the finite-dimensional case. For instance, not every bounded sequence has a convergent subsequence.

Example 6.2.1. Let $\{\vec{u}_i \mid i \in \mathbb{N}\}$ be the following sequence in $[0,1]^\omega$: for all $i, j \in \mathbb{N}$,

$$u_{i,j} = \begin{cases} 0 & \text{if } j \leq i, \\ 1 & \text{otherwise.} \end{cases}$$

Clearly, $\{\vec{u}_i \mid i \in \mathbb{N}\}$ is bounded below by $\vec{0}$ and above by $\vec{1}$, but it has no convergent subsequence. In particular, give any $i \neq i'$, $\mathsf{dist}(\vec{u}_i, \vec{u}_{i'}) = 1$.

This suggests the proof in Section 6.1 will not go through for $\mathsf{Adv}(\mathcal{A}, k, -)$. Indeed, Lemma 6.1.1 fails when we try to generalize it to $\mathsf{Adv}(\mathcal{A})$ (i.e., removing the finite breadth condition).

Example 6.2.2. Let $\{b_i \mid i \in \mathbb{N}\}$ be an enumeration of Act. Consider an automaton $\mathcal{A}$ with state space $\{s_0, s_1, s_2, \ldots\}$, where s_0 is the unique start state. The transition relation consists of triples $\langle s_0, b_i, \mathsf{Dirac}(s_i)\rangle$. It is trivial to see that $\mathcal{A}$ is image finite. For each i, let E_i be the adversary that schedules the transition $s_0 \xrightarrow{b_i} \mathsf{Dirac}(s_i)$ with probability 1. Given any finite subset F of $\mathsf{Path}^{\leq 1}(\mathcal{A})$, we can choose i large enough so that the path $s_0 b_i \,\mathsf{Dirac}(s_i) s_i$ is not in F. Then $\sum_{\pi \in \mathsf{tr}^{-1}(b_i) \setminus F} \mathbf{Q}_{E_i}(\pi) = \mathbf{Q}_{E_i}(s_0 b_i \,\mathsf{Dirac}(s_i) s_i) = 1$.

Nonetheless, we are able to prove Proposition 6.2.2, which is an exact analog of Proposition 6.1.2. The key observation is, in the proof of Proposition 6.1.2, we never invoked the full assumption that $\{E_i \mid i \in \mathbb{N}\}$ is trace convergent. Instead, we used the fact that every convergent sequence is a Cauchy sequence. That is, if $\{D_i \mid i \in \mathbb{N}\}$ converges to a limit, then there exists for every ε a number N_ε such that $\mathsf{dist}(D_i, D_j) \leq \varepsilon$ for all $i, j > N_\varepsilon$. A typical proof of the converse requires the fact that very bounded sequence has a convergent subsequence, which, as we saw in Example 6.2.1, fails in our setting.

To use the full power of trace convergence, we proceed with a contradiction proof. Namely, we assume that $\{D_i \mid i \in \mathbb{N}\}$ converges to the "wrong" limit D, where $D \neq \mathsf{tr}(Q)$. This allows us to choose a particular $\beta \in \mathsf{Act}^{<\omega}$ such that

$D(\beta)$ and $\mathsf{tr}(Q)(\beta)$ are separated by a non-zero distance δ. Using this β, we are able to derive a contradiction based the construction of Q.

We need an auxiliary lemma, which strengthens Lemma 5.3.3.

Lemma 6.2.1. *Assume $\mathcal{A}$ is image finite. Let $\beta \in \mathsf{Act}^{<\omega}$ and $\varepsilon > 0$ be given. Suppose there is an adversary E_0 such that $\mathbf{D}_{E_0}(\beta) \geq \varepsilon$. Then for all $\varepsilon' < \varepsilon$, there exists non-empty finite set $F \subseteq \mathsf{tr}^{-1}(\beta)$ such that for all adversaries E, $\sum_{\pi \in \mathsf{tr}^{-1}(\beta) \setminus F} \mathbf{Q}_E(\pi) \leq \varepsilon'$.*

Proof. By Lemma 5.3.3, we can choose finite $F \subseteq \mathsf{tr}^{-1}(\beta)$ such that for all adversaries E, $\sum_{\pi \in \mathsf{tr}^{-1}(\beta) \setminus F} \mathbf{Q}_E(\pi) \leq \varepsilon'$. It remains to show F is non-empty. If F is empty, then

$$\mathbf{D}_{E_0}(\beta) = \sum_{\pi \in \mathsf{tr}^{-1}(\beta)} \mathbf{Q}_{E_0}(\pi) = \sum_{\pi \in \mathsf{tr}^{-1}(\beta) \setminus F} \mathbf{Q}_{E_0}(\pi) \leq \varepsilon' < \varepsilon.$$

This is a contradiction. □

Proposition 6.2.2. *Let $\mathcal{A}$ be an image finite PA and let $k \in \mathbb{N}$ be given. Let $\{E_i \mid i \in \mathbb{N}\}$ be a sequence of trace convergent adversaries from $\mathsf{Adv}(\mathcal{A}, k, -)$ and write D_i for $\mathbf{D}_{E_i}$. Let Q be constructed as in Section 4.3 and let E denote E_Q (cf. Section 4.1). Then $\mathbf{D}_E$ is the limit of $\{D_i \mid i \in \mathbb{N}\}$ in the space $[0,1]^{\mathsf{Act}^{<\omega}}$.*

Proof. Let D denote the limit of $\{D_i \mid i \in \mathbb{N}\}$. Such D exists because, by assumption, $\{E_i \mid i \in \mathbb{N}\}$ is trace convergent. For the sake of contradiction, suppose that $D \neq \mathbf{D}_E$. Then $\mathsf{dist}(D, \mathbf{D}_E) > 0$, which implies there exists β such that $|D(\beta) - \mathbf{D}_E(\beta)| > 0$. Choose such β and let δ denote $|D(\beta) - \mathbf{D}_E(\beta)|$.

We claim that there is adversary E_0 with $\mathbf{D}_{E_0}(\beta) \geq \frac{\delta}{2}$. If E satisfies this property, set E_0 to E. Otherwise, $\mathbf{D}_E(\beta) < \frac{\delta}{2}$; then it must be the case that $D(\delta) > \mathbf{D}_E(\beta) + \delta \geq \delta$. By the definition of convergence, we may choose i such that $|D(\beta) - D_i(\beta)| \leq \frac{\delta}{2}$. Then we can take E_0 to be E_i. Now we can apply Lemma 6.2.1 and choose non-empty finite set $F \subseteq \mathsf{tr}^{-1}(\beta)$ such that for all adversaries E, $\sum_{\pi \in \mathsf{tr}^{-1}(\beta) \setminus F} \mathbf{Q}_E(\pi) \leq \frac{\delta}{3}$. Let $m := |F|$.

Let $\{\pi_n \mid n \in \mathbb{N}\}$ be the enumeration of $\mathsf{Path}^{<\omega}(\mathcal{A})$ used in the construction of Q. Let $M := \max\{n \mid \pi_n \in F\} + 1$. Then by Corollary 4.3.2, we have

$$\forall \pi \in F \ \lim_{j \in \mathbb{N}} \mathbf{Q}_{E_j^M}(\pi) = \mathbf{Q}_E(\pi).$$

For each $\pi \in F$, choose j_π such that

$$\forall j > j_\pi \ \ |\mathbf{Q}_{E_j^M}(\pi) - \mathbf{Q}_E(\pi)| < \frac{\delta}{3m}.$$

Separately, since $\{D_i \mid i \in \mathbb{N}\}$ converges to D, we may choose N such that $\mathsf{dist}(D, D_i) \leq \frac{\delta}{3}$ for all $i > N$. Let L be the least number such that $L >$

$\max\{j_\pi \mid \pi \in P_{k,\frac{\varepsilon}{3}}\}$ and $\mathsf{index}(E_L^M) > N$. Set $\widehat{L} := \mathsf{index}(E_L^M)$. Now we have:

$$\begin{aligned}
&|D(\beta) - \mathbf{D}_E(\beta)| \\
&\leq |D(\beta) - D_{\widehat{L}}(\beta)| + |D_{\widehat{L}}(\beta) - \mathbf{D}_E(\beta)| \\
&\leq \frac{\delta}{3} + |\sum_{\pi \in \mathsf{tr}^{-1}(\beta)} \mathbf{Q}_{E_L^M}(\pi) - \sum_{\pi \in \mathsf{tr}^{-1}(\beta)} \mathbf{Q}_E(\pi)| \\
&\leq \frac{\delta}{3} + |\sum_{\pi \in \mathsf{tr}^{-1}(\beta)\cap F} \mathbf{Q}_{E_L^M}(\pi) - \sum_{\pi \in \mathsf{tr}^{-1}(\beta)\cap F} \mathbf{Q}_E(\pi) \\
&\qquad\qquad + \sum_{\pi \in \mathsf{tr}^{-1}(\beta)\setminus F} \mathbf{Q}_{E_L^M}(\pi) - \sum_{\pi \in \mathsf{tr}^{-1}(\beta)\setminus F} \mathbf{Q}_E(\pi)| \\
&\leq \frac{\delta}{3} + \sum_{\pi \in \mathsf{tr}^{-1}(\beta)\cap F} |\,\mathbf{Q}_{E_L^M}(\pi) - \mathbf{Q}_E(\pi)| \\
&\qquad + |\sum_{\pi \in \mathsf{tr}^{-1}(\beta)\setminus F} \mathbf{Q}_{E_L^M}(\pi) - \sum_{\pi \in \mathsf{tr}^{-1}(\beta)\setminus F} \mathbf{Q}_E(\pi)| \\
&\leq \frac{\delta}{3} + m \cdot \frac{\delta}{3m} + |\sum_{\pi \in \mathsf{tr}^{-1}(\beta)\setminus F} \mathbf{Q}_{E_L^M}(\pi) - \sum_{\pi \in \mathsf{tr}^{-1}(\beta)\setminus F} \mathbf{Q}_E(\pi)| \\
&\leq \frac{\delta}{3} + \frac{\delta}{3} + \frac{\delta}{3} = \delta.
\end{aligned}$$

The second inequality follows from the fact that $\mathsf{index}(E_L^M) > N$. The fifth follows from the fact that $L > \max\{j_\pi \mid \pi \in P_{k,\frac{\delta}{3}}\}$. The last is by Lemma 6.2.1. This yields a contradiction, therefore it must be the case that $D{=}\mathbf{D}_E$. □

Finally, we check that E_Q is in fact an adversary in $\mathsf{Adv}(\mathcal{A}, k, -)$.

Lemma 6.2.3. *Let $k \in \mathbb{N}$ be given and suppose $\{E_i \mid i \in \mathbb{N}\}$ is a sequence of adversaries from $\mathsf{Adv}(\mathcal{A}, k, -)$. Let Q be the probabilistic execution constructed from $\{E_i \mid i \in \mathbb{N}\}$ as in Section 4.3 and let E denote E_Q (cf. Section 4.1). Then for all $\beta \in \mathsf{Act}^{<\omega}$, $|\beta| > k$ implies $\mathsf{tr}(Q)(\beta) = 0$. Moreover, $E_Q \in \mathsf{Adv}(\mathcal{A}, k, -)$.*

Proof. Let $\pi \in \mathsf{Path}^{<\omega}(\mathcal{A})$ be given. By the construction of Q, $Q(\pi) > 0$ implies $\mathbf{Q}_{E_i}(\pi) > 0$ for some i, which in turn implies $|\,\mathsf{tr}(\pi)| \leq k$. Therefore, $\mathsf{tr}(Q)(\beta) = 0$ whenever $|\beta| > k$. This proves the first claim.

By the definition of E, $E(\pi)(a, \mu) \neq 0$ implies $Q(\pi) \neq 0$. Therefore

$$E(\pi)(a, \mu) \neq 0 \;\Rightarrow Q(\pi a \mu s) \neq 0 \Rightarrow \mathbf{D}_E(\mathsf{tr}(\pi)a) \neq 0,$$

where s is any state in $\mathsf{Supp}(\mu)$. Hence, by the first claim of the present lemma, $E(\pi)(a, \mu) \neq 0$ implies $|\pi| = |\,\mathsf{tr}(\pi)| < k$. This proves $E_Q \in \mathsf{Adv}(\mathcal{A}, k, -)$. □

Corollary 6.2.4. *Let $\mathcal{A}$ be image finite. For all $k \in \mathbb{N}$, the set $\mathsf{TrDist}(\mathcal{A}, k, -)$ is a closed subset of $\langle [0,1]^{\mathsf{Act}^{<\omega}}, \mathsf{dist}\rangle$.*

Proof. By Proposition 6.2.2 and Lemma 6.2.3. □

7

Testing Semantics

7.1 Introduction

A fundamental idea in concurrency theory is that two processes are deemed equivalent if they cannot be distinguished by external observation. Varying the power of the external observer, different notions of behavioral equivalence arise. For processes modeled as labeled transition systems (LTS's), this idea has been thoroughly explored: a large number of behavioral equivalences have been characterized via intuitive *testing scenarios*, also called *button-pushing experiments*.

In a typical button-pushing experiment, we envision a machine equipped with a display and a series of buttons. The process under observation resides within this machine and its activities, represented by action symbols, are shown on the display. An external observer may influence the execution of this process by pressing one or more buttons at various times. The simplest example of such an experiment is the *trace machine* in Figure 7.1, which has an action display but no buttons. It turns out to be sufficient for characterizing the well-known *trace equivalence* for LTS's.

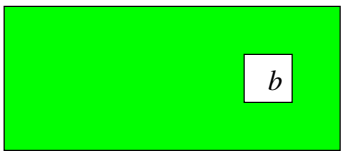

Figure 7.1: The trace machine.

Button-pushing experiments are desirable for a number of reasons. First, they provide a simple and intuitive way to understand behavioral equivalences that are defined more abstractly, e.g., via process algebras or in terms of satisfaction of logical formulas. Second, they provide a unified setting for comparing these behavioral equivalences. We refer to Van Glabbeek [vG01] for an excellent overview of results in this area of *comparative concurrency semantics*. Finally, in a button-pushing experiment, interactions between process and observer take place exclusively via the predefined interface, namely, display and buttons. This

is in keeping with the tradition of modular reasoning, which requires that processes evolve independently from their environments, aside from explicit inputs.

This chapter presents such a testing scenario for probabilistic processes. (For our purposes, a *probabilistic process* may make discrete random choices as well as nondeterministic choices.) This task calls for a nontrivial extension of existing testing scenarios for LTS's, because one must specify a means to "observe" probability distributions. For that end, we develop an extension of the *trace distribution machine* proposed in [SV03, Sto02a], where *null hypothesis tests* are used to provide a link between

- probability distributions derived in an abstract semantics and
- sample observations collected from the trace distribution machine.

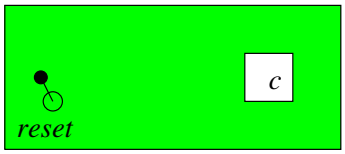

Figure 7.2: The trace distribution machine.

The distinguishing feature of the trace distribution machine (depicted in Figure 7.2) is a *reset* button, which restarts the machine from its initial state. This allows an observer to record traces from multiple runs of the machine. These runs are assumed to be independent; that is, random choices in one run are not correlated with those in another run. However, it is not assumed that nondeterministic choices are resolved in exactly the same way, therefore each run is governed by a possibly different probability distribution.

The semantics of this reset button poses a challenge in the design of hypothesis tests. Even though one can compute frequencies of traces from a sample of m runs, it is not immediately clear what information has been obtained about the m possibly distinct probability distributions. As it turns out, this frequency statistic provides a very natural estimator for the *average* of the m distributions. Thus these m distribution are treated collectively: a typical null hypothesis states that a sample consisting of m runs is generated by a particular sequence of m distributions. These hypothesis tests induce a notion of observational equivalence that coincides with the trace distribution equivalence of [Seg95b]. Therefore, this testing scenario can be viewed as an intuitive justification of the more abstract notion of trace distribution equivalence.

Another challenging issue is infinite behaviors of probabilistic processes. These may include non-terminating runs, as well as states with infinitely many outgoing transitions. In contrast, experiments on the trace distribution machine are of a finite character: an observer can record only finitely many symbols from a single run and can observe only finitely many runs. To overcome this

discrepancy, the authors of [SV03] impose the restriction that all probabilistic processes in question must be *finitely branching*. That is, every state enables at most finitely many transitions, regardless of action labels. Moreover, an Approximation Induction Principle is proved to justify the reduction of an infinite probabilistic behavior to its finite "sub-behaviors".

The present chapter builds upon the results of [SV03]. We introduce an *extended trace distribution machine*, shown in Figure 7.3 below. This machine allows the observer to suppress all but a finite number of actions, so that each experiment is associated with a fixed, finite sample space. The observational equivalence induced by this new testing scenario also coincides with Segala's trace distribution equivalence, but for a wider class of processes, namely, for all image finite processes. A process is said to be *image finite* if, for each state s and action a, only finitely many a-transitions are enabled in s. In particular, if the action alphabet Act is infinite, then a state s may enable infinitely many transitions (say, one for each action label a). Therefore, image finiteness is strictly weaker than finite branching.

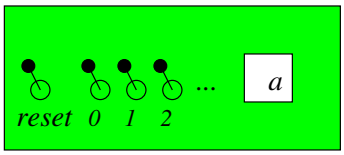

Figure 7.3: The extended trace distribution machine.

Following [SV03], our technical work is carried out in the framework of *probabilistic automata (PA)*, as discussed in Chapter 3. This framework has seen many applications in the analysis of distributed algorithms [Agg94, LSS94, PSL00, SV99]. More importantly, its relative simplicity frees us from particular features that may hamper the portability of our results. Indeed, we focus on semantic objects induced by PAs, as opposed to the automata themselves. These objects are probability distributions on computation paths (the so-called *probabilistic executions*) and probability distributions on traces (the so-called *trace distributions*).

As we saw in Chapters 5 and 6, probabilistic executions and trace distributions can be viewed very naturally as trees with probabilistic branching, so that our technical developments quickly migrated towards the more fundamental settings of ordered sets and metric spaces. We believe these developments can be easily adapted to other settings where the semantic objects of interest are such probabilistic trees, regardless of the particular framework under which these trees are induced.

Finally, we point out that the present chapter can be seen as an application of the extensive technical machinery we have developed so far in this thesis for probabilistic automata. Notably, the following facts are key ingredients in the proof that our testing semantics coincides with Segala's trace distribution

semantics (Theorem 7.4.5).

- For all $k, l \in \mathbb{N}$, the set $\{\mathbf{P}_{D,k,l} \mid D \in \mathsf{TrDist}(\mathcal{A}, k, l)\}$ of discrete probability distributions on $(\mathsf{Act}_l)^{\leq k}$ is closed under convex combinations (Corollary 4.2.4).
- If $\mathcal{A}$ is image finite, then, for all $k, l \in \mathbb{N}$, the set

$$\{\mathbf{P}_{D,k} \mid D \in \mathsf{TrDist}(\mathcal{A}, k, l)\}$$

  is a closed subset of $[0,1]^{\mathsf{Act}^{<\omega}}$ (Corollary 6.1.6).
- If $\mathcal{A}$ is image finite, $\mathsf{TrDist}(\mathcal{A})$ forms an algebraic CPO under $\leq_\flat$ and the set of compact elements is precisely $\bigcup_{k,l\in\mathbb{N}} \mathsf{TrDist}(\mathcal{A}, k, l)$ (Theorem 5.3.15).

Related Work

Aside from the trace distribution machine of [SV03], several testing preorders and equivalences for probabilistic processes have been proposed in the literature [Chr90, Seg96, GN98, CDSY99, JY02]. All these papers study testing relations in the style of De Nicola and Hennesy [NH84]. That is, a *test* is defined as a (probabilistic) process that interacts with a system via shared actions and reports either success or failure. The various testing relations are then obtained by comparing success probabilities. Unlike [SV03] or the present chapter, these papers do not describe how success probabilities can be observed from an external point of view. Therefore, in our opinion, these relations are not completely observational. In that sense, our work is more closely related to the seminal paper of Larsen and Skou [LS91], where probabilistic bisimulation is characterized by a testing scenario based on hypothesis testing. Technically, the setting in [LS91] is more restrictive than ours because of their minimal deviation assumption, which imposes a uniform lower bound on all transition probabilities and hence an upper bound on the probabilistic branching degree.

Also closely related is the fast emerging field of *statistical model checking* [YS02, YKNP04, SVA04, You05]. Traditionally, a probabilistic model checker does its job by exploring the state space and computing numerically all relevant probabilities. In statistical model checking, the idea is instead to collect sample runs from the model. Properties of interest are formulated as test hypotheses and, by increasing the number of sample runs, one can control the probability of producing an erroneous answer to the model checking question. So far, statistical model checking techniques have been developed for discrete and continuous time Markov chains [YKNP04, SVA04], semi-Markov processes [SVA04] and stochastic discrete event systems [YS02, You05]. In most of these models, the notions of delay and relative timing are treated explicitly, whereas in our approach nondeterminism is used to model timing uncertainty. Much of our effort goes to show that standard techniques in hypothesis testing can be used

to distinguish processes even in the presence of nondeterminism, as long as all nondeterministic choices are within a closed set.

Our development differs in another way from many other works on stochastic systems (e.g., [Eda95, BK98, DEP02]), which focus more on functional behaviors of these processes and hence probability distributions on the state space. These distributions are *conditional* upon occurrences of events, which are often interpreted as inputs to a system. In contrast, we focus on probability distributions on computation paths and traces, therefore we must take into account probability distributions on events, in addition to distributions on states. In this respect, our development is closer to [Vat01], which studies properties of distribution functions (a generalized notion of language) generated by finite-state probabilistic automata. One may argue this distinction between state-based and action-based reasonings is inconsequential, yet our experience suggests the slight difference in interpretation can lead to divergence in the methods of analysis and eventually in the types of application domains.

Organization

In Section 7.2, we explain on an informal level the design and motivation of our testing scenario. Section 7.3 presents the technical definition of acceptable observations. The main characterization theorem is proven in Section 7.4 and concluding remarks follow in Section 7.5. We refer our reader to Chapters 2 and 3 of this thesis for mathematical preliminaries and basic definitions of the probabilistic automata framework.

7.2 Hypothesis Tests

Before presenting our results on a technical level, we give an informal overview of the proposed testing scenario. To keep things simple and focused, we explain the design of hypothesis tests for the trace distribution machine. The extended version is treated in Section 7.2.4.

We view the ensuing discussion a contribution of the present chapter, although the trace distribution machine itself is not. In particular, we distinguish between two interpretations of nondeterministic choices: angelic vs. demonic. Such a clear and systematic treatment is absent in [SV03, Sto02a].

7.2.1 Button-Pushing Experiments

As described in Section 7.1, a typical button-pushing experiment consists of a process operating inside a black box. Given a process $\mathcal{A}$, such an experiment induces a set $\mathsf{Obs}(\mathcal{A})$ of all observations that are possible/acceptable under $\mathcal{A}$. This in turn yields an observational equivalence: two LTS's $\mathcal{A}_1$ and $\mathcal{A}_2$ are equivalent if and only if $\mathsf{Obs}(\mathcal{A}_1) = \mathsf{Obs}(\mathcal{A}_2)$.

Consider for instance the trace machine shown in Figure 7.1, which characterizes trace semantics for image finite LTS's [vG01]. This machine has no buttons at all, thus the observer cannot influence its execution. During a single experiment, the observer records the contents of the display over time, yielding a finite trace of the process inside the machine. Gathering all possible observations, we obtain a testing scenario that corresponds to trace equivalence.

Example 7.2.1. The LTS's $\mathcal{A}_1$ and $\mathcal{A}_2$ in Figure 7.4 below are trace equivalent and have the same observations with respect to the trace machine: ϵ (the empty trace), a, ab and ac.

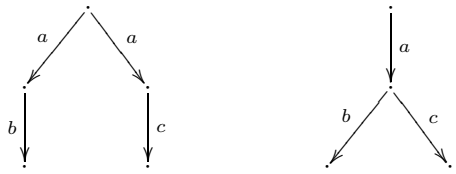

Figure 7.4: LTS's $\mathcal{A}_1$ and $\mathcal{A}_2$.

To obtain a testing scenario for probabilistic processes, a *reset* button is added to the trace machine, allowing the observer to undo all changes and restart from the initial state [SV03]. The resulting trace distribution machine is depicted in Figure 7.2. An experiment on the trace distribution machine is carried out as follows.

(1) First, the observer fixes the *type* of the experiment: two natural numbers k and m. The first specifies the maximum length of each run and is referred to as the *depth* of the experiment. The second specifies the number of runs to be executed and is referred to as the *width*.

(2) The observer then starts the machine by pushing the reset button.

(3) As the machine executes, the action symbols appearing on the display are recorded in succession.

(4) When the display becomes empty, or when the observer has recorded k actions, the machine is reset and recording starts in a fresh column.

(5) The experiment stops when m runs of the machine have been recorded.

Table 7.5 below illustrates a sample that *may* be obtained in a type-$\langle 2, 6\rangle$ experiment conducted on the process $\mathcal{A}_1$ from Figure 7.1. (In our setting, LTS's are degenerate probabilistic processes.)

So far, we have described how to collect a sample from the trace distribution machine. The next step is to use hypothesis testing to define the set of *type*-$\langle k, m\rangle$

1	2	3	4	5	6
a	a	a	a	a	a
c	b	c	b	c	c

Figure 7.5: A possible type-$\langle 2, 6\rangle$ sample from $\mathcal{A}_1$ of Figure 7.4.

acceptable observations of $\mathcal{A}$, denoted $\mathsf{Obs}(\mathcal{A}, k, m)$, for a given process $\mathcal{A}$ and sample type $\langle k, m\rangle$. Then $\mathsf{Obs}(\mathcal{A})$ is defined to be the union $\bigcup_{k,m} \mathsf{Obs}(\mathcal{A}, k, m)$. In this way, two processes $\mathcal{A}_1$ and $\mathcal{A}_2$ are distinguished in our semantics if and only if there exists sample type $\langle k, m\rangle$ such that $\mathsf{Obs}(\mathcal{A}_1, k, m) \neq \mathsf{Obs}(\mathcal{A}_2, k, m)$.

As we mentioned in Section 7.1, this task is complicated by the semantics of the reset button. Namely, nondeterministic choices may be resolved differently in the various runs of an experiment, so that the traces recorded from these runs need not be identically distributed. These nondeterministic choices are said to be *demonic*, because we have no control over them.

To facilitate understanding, we first consider hypothesis tests in the weaker setting of *angelic* nondeterministic choices, where we do assume control. In Section 7.2.3, we explain how these tests can be adapted to the original setting of demonic choices.

7.2.2 Hypothesis Testing: Angelic Nondeterminism

Consider a type-$\langle k, m\rangle$ experiment on a probabilistic process $\mathcal{A}$ with finite action alphabet[1] Act. Let $\mathsf{Act}^{\leq k}$ denote the set of traces with length at most k. Suppose we can make sure that nondeterministic choices are resolved in the same way in all m runs, so that every run is associated with the same discrete probability distribution D on $\mathsf{Act}^{\leq k}$.

Fix trace β in $\mathsf{Act}^{\leq k}$. We can view the m runs of this experiment as m independent *Bernoulli trials* as follows: during each run, a *success* occurs if the record for that run contains exactly β; otherwise, we have a *failure*. By our assumption of angelic nondeterminism, these trials are identically distributed and the common parameter θ is precisely $D(\beta)$.

It is well-known that the frequency of successes from a Bernoulli sample is a *sufficient statistic* for the parameter θ. Intuitively, the number of successes in a sample contains all the information about θ that is present in the sample. This suggests we define our hypothesis test in terms of the frequency of successes. In fact, sine $\mathsf{Act}^{\leq k}$ is finite, we can do so for all traces β simultaneously, by devising a test with this null hypothesis: "the underlying probability distribution is D." This hypothesis is *accepted* if, for every β, the frequency of successes in the

[1]This finiteness restriction on Act can be replaced by a *finite branching* condition on processes [SV03]. In Section 7.2.4, we discuss the extended trace distribution machine, which accommodates for image finite processes with countably infinite action alphabet.

actual outcome is in the interval $[D(\beta) - r, D(\beta) + r]$; otherwise, it is *rejected*. Here r is some appropriate real number between 0 and 1. To discuss how we choose r, we need to bring in some terminology.

Since hypothesis tests are concerned with yes/no questions, there are two possible types of errors: *false rejection* and *false acceptance*. A good test should guarantee that the probability of committing either error is low. However, it is often hard to control these errors independently. In some cases, it is proven to be impossible to control false acceptance uniformly among all alternative parameters, while conforming to a certain tolerance of false rejection (cf. Chapter 8 of [CB90].) Therefore one typically starts with tests that control false rejections, while keeping false acceptance small. We adopt the same strategy, namely, given any $\alpha \in [0, 1]$, we define tests with probability of false rejection at most α. These tests are said to have *level* α.

It may seem desirable to have tests that never commit false rejection errors (i.e., level 0). However, this strategy leads to rather uninteresting tests, because it forces acceptance whenever the actual outcome has nonzero probability under the null hypothesis. To avoid such triviality, one typically fixes a small but nonzero level, e.g., $\alpha = 0.05$. This quantity α determines the size of the *acceptance region*, which is the set of outcomes that lead to acceptance of the null hypothesis. In particular, an acceptance region should contain just enough possible outcomes so that the probability of false rejection is below α. A smaller acceptance region would violate the level-α requirement, while a larger one would lead to higher probability of false acceptance errors.

In our case, the size of the acceptance region depends on the value r and we choose the smallest r that give rise to a level-α test. Now we can define $\mathsf{Obs}(D, k, m)$ to be this acceptance region, namely, the set of possible outcomes such that the frequency of successes for every β is in the interval $[D(\beta) - r, D(\beta) + r]$. The set of acceptable type-$\langle k,\ m\rangle$ observations for $\mathcal{A}$ is in turn given as $\bigcup_D \mathsf{Obs}(D, k, m)$, where D ranges over all possible distributions induced by $\mathcal{A}$. The following example illustrates such hypothesis tests for a fair coin and a biased coin, respectively.

Example 7.2.2. Consider the two probabilistic processes $\mathcal{A}_1$ and $\mathcal{A}_2$ in Figure 7.6 below. We interpret the symbol a as the action of flipping a coin, while b and c announce on which side the coin lands. Then $\mathcal{A}_1$ models a fair coin, i.e., the uniform distribution on the set $\{ab, ac\}$. Similarly, $\mathcal{A}_2$ models a coin with bias $\frac{1}{3}$ for heads, i.e., a distribution assigning probability $\frac{1}{3}$ to the trace ab and $\frac{2}{3}$ to the trace ac.

Suppose α is set at 0.05 and we consider experiments of type $\langle 2,\ 100\rangle$. In other words, we observe 100 runs of length 2 each. The acceptance region for $\mathcal{A}_1$ consists of sequences in which the traces ab occurs between 41 and 59 times, while in the acceptance region for $\mathcal{A}_2$ the trace ab occurs between 24 and 42 times. If ab is actually observed 45 times, we answer "yes" in the test for $\mathcal{A}_1$ and "no" in the test for $\mathcal{A}_2$. Therefore, $\mathcal{A}_1$ and $\mathcal{A}_2$ are *distinguished* in the testing semantics.

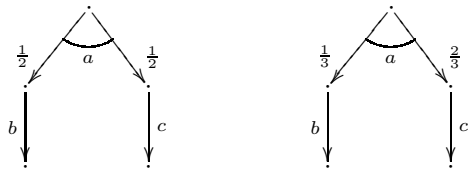

Figure 7.6: Fair coin vs. biased coin.

Intuitively, the distinguishing power of this testing scenario is a direct consequence of the well-known (weak) law of large numbers. Given any small ε, we can toss a coin sufficiently many times so that it is extremely unlikely to observe a sample mean that deviates from the true bias by more than ε. This allows us to "separate" the acceptance regions of two coins with different biases.

It is interesting to note that the observational equivalence, thus obtained, is independent of the choice of α, because we have the freedom to vary the number of runs. In general, as α decreases, we must enlarge the acceptance regions for the two processes in question, possibly increasing the overlap between them. Therefore more runs need to be performed so that we can find sample points residing in the difference of the two acceptance regions.

7.2.3 Hypothesis Testing: Demonic Nondeterminism

In the angelic case, a width-m experiment on the trace distribution machine can be likened to tossing the same coin m times. Our testing scenario thus boils down to the problem of distinguishing two coins with different biases. In the demonic case, a width-m experiment can be likened to tossing a sequence of m coins with possibly different biases, and our testing scenario reduces to the following (slightly more complicated) problem.

Suppose we have a sequence S of coins with biases $p_0, p_1, p_2, \ldots$ such that every p_i is in a closed interval $I \subseteq [0,1]$. Given any m, we devise a hypothesis test for the first m coins in S as follows: a length-m sequence of heads and tails leads to a "yes" answer if and only if the frequency of heads falls in the interval $[\overline{p} - r, \overline{p} + r]$. Here $\overline{p}$ is the average of $p_0, \ldots, p_{m-1}$ and r is chosen as before to guarantee a level-α test.

Suppose there is another coin with bias $q \notin I$ and, for each m, we construct a test for m tosses of the new coin in exactly the same way. (Here the midpoint of the interval is simply q.) The question we try to answer is: is there an m for which there exists a sample point that leads to a "yes" answer in the test for $p_0, \ldots, p_{m-1}$ but a "no" answer in the test for $q, \ldots, q$?

Again, we can appeal to the weak law of large numbers in the second test, with repeated tosses of the same coin. As it turns out, the same intuition also applies in the first test, despite the fact that the p_i's are possibly different. In

Section 4.1, we prove an analog of the weak law of large numbers for independent Bernoulli variables, replacing the bias of a single coin with the average bias of m different coins (Lemma 7.4.2). This key observation, together with the fact that $\overline{p}$ and q are separated by the closed interval I, allows us to separate two acceptance regions just as in the angelic case.

Using the same trick of treating all traces in $\mathsf{Act}^{\leq k}$ simultaneously, we generalize the above argument on coin tosses to trace distributions. It is therefore important that the set of all trace distributions of a probabilistic process forms a convex closed set.

7.2.4 Extension to Countably Infinite Action Alphabet

So far we have worked with processes with finite action alphabet, so that each length-k run has finitely many possible outcomes (namely, traces in $\mathsf{Act}^{\leq k}$). This is an important property because it allows us to generalize our separation argument from one particular trace to all possible traces. To preserve this property in the case of countably infinite action alphabet, we add buttons $0, 1, 2, \ldots$ to the trace distribution machine in Figure 7.2, yielding the *extended* trace distribution machine. This is depicted in Figure 7.3.

At the start of each experiment (i.e., Step (1) in Section 7.2.1), the observer fixes not only the depth and width of the experiment, but also the *breadth*. This is done by pressing exactly one of the buttons $l \in \mathbb{N}$, indicating that only the first l actions $\{b_0, b_1, \ldots, b_{l-1}\}$ of the alphabet[2] are enabled during the entire experiment. We then proceed exactly as before. This new feature of action switches can be thought of as a "finite testing policy": each experiment focuses on a finite number of possibilities. Since the observer may free an arbitrarily large number of actions, this is a sufficient method of exploring the entire structure.

Notice, the type of an experiment now has three arguments: k, l and m. Given a process $\mathcal{A}$, $\mathsf{Obs}(\mathcal{A})$ is defined as the union $\bigcup_{k,l,m} \mathsf{Obs}(\mathcal{A}, k, l, m)$, where $\mathsf{Obs}(\mathcal{A}, k, l, m)$ is the set of type-$\langle k, l, m\rangle$ acceptable outcomes of $\mathcal{A}$. This induces an observational equivalence that coincides with Segala's trace distribution equivalence, provided the processes are image finite. The image-finite requirement is necessary for the various convergence properties that are essential in our proofs. (This is very much analogous to the situation of LTS's and the trace machine.)

From now on, we studies exclusively image finite processes and the extended trace distribution machine. For brevity, we shall omit the word "extended".

[2] We assume a fixed enumeration of Act (cf. Chapter 3).

7.3 Observations

We begin this section by recalling the procedure of sample collection from a trace distribution machine. Then we identify samples that are *acceptable* if the trace distribution machine operates as specified by a probabilistic automaton $\mathcal{A}$. A sample O falls into this category just in case there exists a possible sequence of trace distributions $D_0, \ldots, D_{m-1}$ under which O is an acceptable outcome. Such samples will constitute the set of observations of $\mathcal{A}$. To save space, we use $\vec{D}$ to denote (syntactically) $D_0, \ldots, D_{m-1}$. Similarly for $\vec{D}'$, $\vec{K}$, etc.

7.3.1 Sampling

We associate with each experiment a triple $\langle k, l, m\rangle$ of natural numbers. We call this the *type* of the experiment, which specifies some parameters in the data collection procedure. More precisely, an observer conducts a *depth-k*, *breadth-l* and *width-m* experiment on a trace distribution machine as follows.

(1) First, the observer presses the button labeled by l, activating the actions in Act_l.

(2) The observer then starts the machine by pushing the reset button.

(3) As the machine executes, the action symbols appearing on the display are recorded in succession.

(4) When the display becomes empty, or when the observer has recorded k actions, the machine is reset and recording starts in a fresh column.

(5) The experiment stops when m runs of the machine have been recorded.

During such an experiment, an observer records a sequence $\beta_0, \ldots, \beta_{m-1}$, where each β_i is a sequence of actions symbols from Act_l and has length at most k. We call such a record O a *sample* of depth k, breadth l and width m (or simply a sample of *type* $\langle k, l, m\rangle$). A trace β is said to *appear* in $\beta_0, \ldots, \beta_{m-1}$ if $\beta = \beta_i$ for some i. When k, l and m are clear from context, we will write $\mathcal{U}$ for the universe of all possible samples of type $\langle k, l, m\rangle$; that is, $\mathcal{U} := ((\mathsf{Act}_l)^{\leq k})^m$.

We assume the trace distribution machine is governed by a PA $\mathcal{A}$. During each run, the trace distribution machine chooses a trace β according to some trace distribution D of $\mathcal{A}$. When the observer presses the reset button, the machine returns to the initial state of $\mathcal{A}$ and starts over with a possibly different trace distribution. Since all actions outside Act_l are blocked, and each time the machine is allowed to perform at most k steps, a run of the trace distribution machine is governed by a trace distribution from $\mathsf{TrDist}(\mathcal{A}, k, l)$. Thus, each sample O of width m is generated by a sequence of m trace distributions from $\mathsf{TrDist}(\mathcal{A}, k, l)$.

Let us focus for a moment on a single run. Recall from Chapter 3 that every trace distribution $D \in \mathsf{TrDist}(\mathcal{A}, k, l)$ induces a discrete probability distribution $\mathbf{P}_{D,k}$ on $(\mathsf{Act}_l)^{\leq k}$. For every $\beta \in (\mathsf{Act}_l)^{\leq k}$, $\mathbf{P}_{D,k}[\beta]$ equals:

- $D(\beta)$, if the length of β is exactly k;
- $D(\beta) - \sum_{a \in \mathsf{Act}_l} D(\beta a)$, otherwise.

In other words, $\mathbf{P}_{D,k}[\beta]$ gives the probability that the observer records *exactly* the trace β during the current run. The first clause corresponds to the case in which the observer resets the machine because k symbols have been recorded, while the second correspond to the case in which the display becomes empty after β.

Now we put together the m runs in an experiment. Note that each run involves two distinct types of choices: first the machine chooses a trace distribution D, then D in turn chooses a trace β. We do not make any assumptions on the first type of choices. However, once D_i is chosen for run i, D_i is solely responsible for selecting a trace β_i. That is, for any $i \neq j$, the choice of β_i by D_i is independent from the choice of β_j by D_j. Therefore, assuming trace distributions $\vec{D}$ are chosen, the probability of generating a depth-k sample $O = \beta_0, \ldots, \beta_{m-1}$ can be expressed as:

$$\mathbf{P}_{\vec{D},k}[O] := \prod_{i=0}^{m-1} \mathbf{P}_{D_i,k}[\beta_i].$$

For a set $\mathcal{O}$ of such samples, we have $\mathbf{P}_{\vec{D},k}[\mathcal{O}] := \sum_{O \in \mathcal{O}} \mathbf{P}_{\vec{D},k}[O]$.

7.3.2 Frequencies

Our statistical analysis is based on the frequencies with which finite traces from $(\mathsf{Act}_l)^{\leq k}$ appear in a sample O. Formally, the *frequency* of β in O is given by:

$$\mathsf{freq}(O)(\beta) := \frac{|\{i \mid 0 \leq i < m \text{ and } \beta = \beta_i\}|}{m}.$$

Although each run is governed by a possibly different distribution, we can still obtain useful information from frequencies of traces. This is done as follows. Fix k, l, m, $\vec{D}$ and $\beta \in (\mathsf{Act}_l)^{\leq k}$. For each $0 \leq i \leq m-1$, we say that a *success* occurs at the i-th run just in case the observer records exactly β at the i-th run. Thus, the probability of a success at the i-th run is given by $\mathbf{P}_{D_i,k}[\beta]$. This can be viewed as a Bernoulli distribution with parameter $\mathbf{P}_{D_i,k}[\beta]$. Let X_i denote such a random variable. Then the random variable $Z := \frac{1}{m} \sum_{i=1}^{m} X_i$ represents the frequency of successes in the m trials governed by $\vec{D}$. Moreover,

using linearity of expectation, the expected value of this frequency is:

$$\begin{aligned}
&\mathbf{E}_{\beta}^{\vec{D},k} && := \mathbf{E}\, Z \\
&= \mathbf{E}(\frac{1}{m}\sum_{i=0}^{m-1} X_i) \\
&= \frac{1}{m}\sum_{i=0}^{m-1} \mathbf{E}(X_i) \\
&= \frac{1}{m}\sum_{i=0}^{m-1} \mathbf{P}_{D_i,k}[\beta].
\end{aligned}$$

Notice, both $\mathsf{freq}(O)$ and $\mathbf{E}^{\vec{D},k}$ can be viewed as points in the metric space $[0,1]^{\mathsf{Act}^{\leq k}}$ with distance function $\mathsf{dist}(\vec{u},\vec{v}) := \sup_{\beta\in\mathsf{Act}^{\leq k}} |u_\beta - v_\beta|$. Thus the distance $\mathsf{dist}(\mathsf{freq}(O), \mathbf{E}^{\vec{D},k})$ provides a very natural way to quantify the deviation between $\mathsf{freq}(O)$ and $\mathbf{E}^{\vec{D},k}$. This plays a central role in classifying acceptable outcomes of $\vec{D}$.

7.3.3 Acceptable Outcomes: Motivation

Returning to our original goal, we would like to define a set of acceptable outcomes of $\mathcal{A}$. This is done by defining a set of acceptable outcomes for each sequence $\vec{D}$ of trace distributions. Thus, in the terminology of hypothesis testing, we develop a test with this null hypothesis: the sample O is generated by the sequence $\vec{D}$.

Fix an $\alpha \in (0,1)$ as the desired level of the test. Also fix the sample type $\langle k,\, l,\, m\rangle$. The set $\mathsf{Obs}(\vec{D},k,l,m,\alpha)$ of acceptable outcomes should then satisfy the following:

1. $\mathbf{P}_{\vec{D},k}[\mathsf{Obs}(\vec{D},k,l,m,\alpha)] \geq 1-\alpha$, and
2. $\mathbf{P}_{\vec{D}',k}[\mathsf{Obs}(\vec{D},k,l,m,\alpha)]$ is minimized for different choices of $\vec{D}'$ with $\vec{D}' \neq \vec{D}$.

Condition 1 says the probability of false rejection (i.e., rejecting O as a sample generated by $\vec{D}$ while it is such) is at most α. Condition 2 says the probability of false acceptance (i.e., accepting O as a sample generated by $\vec{D}$ while it is not) should be reasonably small. Note that the probability of false acceptance depends highly upon the choice of $\vec{D}'$. Loosely speaking, if $\vec{D}$ and $\vec{D}'$ are very close to each other, then the probability of false acceptance becomes very high.

The design of our test stems from the concept of interval estimation. After each experiment, we try to make an educated guess about the trace distributions governing our machine, based on the sample just observed.

In case the m trials are identically distributed, i.e., controlled by the same trace distribution D, one typically uses $\mathsf{freq}(O)(\beta)$ as an estimator for the value $\mathbf{P}_{D,k}[\beta]$. (By virtue of Proposition 3.3.5, this also gives an estimator for D.) Since the probability of making exactly the right guess is small, an interval around $\mathsf{freq}(O)(\beta)$ is used to guarantee that the guess is correct with probability $1-\alpha$, where α is the prescribed level. That is, if $\mathsf{freq}(O)(\beta)$ is observed, then our guess is $\mathbf{P}_{D,k}[\beta]$ falls in the interval $[\mathsf{freq}(O)(\beta)-r, \mathsf{freq}(O)(\beta)+r]$, where r depends on the level α.

Inverting this interval around $\mathbf{P}_{D,k}[\beta]$, we obtain a set of values for $\mathsf{freq}(O)(\beta)$, namely, the interval $[\mathbf{P}_{D,k}[\beta]-r, \mathbf{P}_{D,k}[\beta]+r]$. If a frequency from this interval is actually observed, then our guess about $\mathbf{P}_{D,k}[\beta]$ would be correct. Thus, a frequency vector $\mathsf{freq}(O)$ is deemed acceptable if for all β, $\mathsf{freq}(O)(\beta)$ is within the appropriate interval around $\mathbf{P}_{D,k}[\beta]$.

In the formal definitions that follow, the situation is slightly different: we do not always have the same trace distribution in all m trials. Thus we cannot give an estimate to the value $\mathbf{P}_{D,k}[\beta]$ for a single trace distribution D. Instead, we use $\mathsf{freq}(O)(\beta)$ as an estimator for $\mathbf{E}_\beta^{\vec{D},k} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{P}_{D_i,k}[\beta]$, an average from the m trace distributions.

7.3.4 Acceptable Outcomes: Definition

As explained above, we accept a sample O if $\mathsf{freq}(O)$ is within some distance r of the value $\mathbf{E}^{\vec{D},k}$. Our task is to find an appropriate $r \in [0,1]$ such that Condition 1 is satisfied. Moreover, for Condition 2, we need to minimize r in order to reduce the probability of false acceptance.

Recall that the (closed) ball centered at $\mathbf{E}^{\vec{D},k}$ with radius r is given by:

$$\mathbf{B}_r(\mathbf{E}^{\vec{D},k}) := \{v \in [0,1]^{(\mathsf{Act}_l)^{\leq k}} \mid \forall \beta \in (\mathsf{Act}_l)^{\leq k}, |v(\beta) - \mathbf{E}_\beta^{\vec{D},k}| \leq r\}.$$

Then $\mathsf{freq}^{-1}(\mathbf{B}_r(\mathbf{E}^{\vec{D},k}))$ is the set of samples whose frequencies deviate from the average $\mathbf{E}^{\vec{D},k}$ by at most r.

Definition 7.3.1. Fix $k, l, m \in \mathbb{N}$ and a sequence $\vec{D}$ of trace distributions from $\mathsf{TrDist}(\mathcal{A}, k, l)$. Let

$$\bar{r} := \inf\{r \mid \mathbf{P}_{\vec{D},k}[\mathsf{freq}^{-1}(\mathbf{B}_r(\mathbf{E}^{\vec{D},k}))] > 1-\alpha\}.$$

The set of type-$\langle k, l, m\rangle$ *acceptable outcomes* of $\vec{D}$ (with level α) is defined to be:

$$\mathsf{Obs}(\vec{D}, k, l, m, \alpha) := \mathsf{freq}^{-1}(\mathbf{B}_{\bar{r}}(\mathbf{E}^{\vec{D},k})) = \{O \mid \mathsf{dist}(\mathsf{freq}(O), \mathbf{E}^{\vec{D},k}) \leq \bar{r}\}.$$

The set of type-$\langle k, l, m\rangle$ *acceptable outcomes* of $\mathcal{A}$ (with level α) is then:

$$\mathsf{Obs}(\mathcal{A}, k, l, m, \alpha) := \bigcup_{\vec{D} \in (\mathsf{TrDist}(\mathcal{A},k,l))^m} \mathsf{Obs}(\vec{D}, k, l, m, \alpha).$$

Example 7.3.1. Let Act be $\{a, b, c\}$ and α be 0.05. Consider the automata $\mathcal{A}$ in Figure 7.7, with a nondeterministic choice between two branches. Let $\vec{D}$ be a sequence of 10 trace distributions generated by: four adversaries that choose the left branch with probability 1 and six that choose the right branch with probability 1.

Then the average of the 10 induced trace distributions assign the value 0.4 to a and 0.3 to each of b and c. Notice the frequency of a in every possible outcome is 0.4. Thus the following two outcomes have the greatest distance from the average: the one in which b never occurs and the one in which c never occurs. It is easy to verify that $\mathsf{Obs}(\vec{D}, 1, 3, 10, 0.05)$ contains all but these two outcomes.

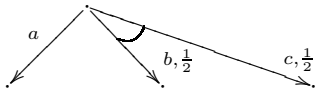

Figure 7.7: Nondeterministic branching

It is interesting to note that, while our notion of acceptable outcomes captures the clustering of samples around the expected value, it often fails to capture individual outcomes with relatively high probability. We illustrate this point with the following example.

Example 7.3.2. Consider an almost fair coin, say, with 0.51 for heads and 0.49 for tails. Suppose we toss this coin 10 times. The most likely outcome, all heads, has frequency vector $\langle 1, 0 \rangle$, which lies very far from the expected frequency of $\langle 0.51, 0.49 \rangle$. In fact, it is easy to check that for $\alpha = 0.005$, this most likely outcome is rejected.

Finally, we define our notion of observation preorder based on acceptable outcomes.

Definition 7.3.2. Let $\mathcal{A}, \mathcal{B}$ be probabilistic automata and let $\alpha \in (0, 1)$ be given. We write $\mathcal{A} \leq_\alpha \mathcal{B}$ if, for all $k, l, m \in \mathbb{N}$, $\mathsf{Obs}(\mathcal{A}, k, l, m, \alpha)$ is a subset of $\mathsf{Obs}(\mathcal{A}, k, l, m, \alpha)$. We say that $\mathcal{A}$ and $\mathcal{B}$ are *observationally indistinguishable* up to level α just in case $\mathcal{A} \leq_\alpha \mathcal{B}$ and $\mathcal{B} \leq_\alpha \mathcal{A}$.

7.4 Characterization of Trace Distribution Semantics

Let us briefly recapitulate our development. Our goal is to show that the testing preorder defined in Section 7.3.4 coincides with trace distribution inclusion, as defined in Section 3.2. In Section 5.3, we established that the set of trace distributions of an image finite automaton forms an algebraic CPO. Therefore the following are equivalent for image finite automata $\mathcal{A}$ and $\mathcal{B}$:

– $\mathcal{A} \leq_{\mathsf{td}} \mathcal{B}$;

– for all $k, l \in \mathbb{N}$, $\mathcal{A} \leq_{\mathsf{td}}^{k,l} \mathcal{B}$.

By virtue of this observation, it suffices to prove the following finitary characterization theorem.

Theorem 7.4.1. *Let $\mathcal{A}$ and $\mathcal{B}$ be image finite probabilistic automata. Let $\alpha \in (0,1)$ and $k, l \in \mathbb{N}$ be given. We have $\mathsf{TrDist}(\mathcal{A}, k, l) \subseteq \mathsf{TrDist}(\mathcal{B}, k, l)$ if and only if, for all m, $\mathsf{Obs}(\mathcal{A}, k, l, m, \alpha) \subseteq \mathsf{Obs}(\mathcal{B}, k, l, m, \alpha)$.*

Since $\mathsf{Obs}(\mathcal{A}, k, l, m, \alpha)$ is entirely defined in terms of $\mathsf{TrDist}(\mathcal{A}, k, l)$ and parameters k, l, m and α, the "only if" direction of Theorem 7.4.1 is trivial. For the converse, we assume there is $D \in \mathsf{TrDist}(\mathcal{A}, k, l) \setminus \mathsf{TrDist}(\mathcal{B}, k, l)$ and our goal is to find $m \in \mathbb{N}$ and a sample $O \in \mathsf{Obs}(\mathcal{A}, k, l, m, \alpha) \setminus \mathsf{Obs}(\mathcal{B}, k, l, m, \alpha)$.

Intuitively, we obtain such O by running the trace distribution machine repeatedly under D. For each $m \in \mathbb{N}$, let D^m denote the length-m sequence in which every element is D. Recall from Section 7.3.4 that an outcome is acceptable if its frequency vector deviates minimally from the expected frequency vector. Our claim is, as the number of trials increases, the amount of deviation allowed decreases to 0. In other words, given any small $\delta > 0$, we can find $m \in \mathbb{N}$ such that any acceptable outcome of a width-m experiment must have a frequency vector within distance δ of the expectation. This claim, together with the fact that we can always separate the point $\mathbf{P}_{D^m,k}$ from the set $\{\mathbf{P}_{\vec{K},k} \mid \vec{K} \in \mathsf{TrDist}(\mathcal{B}, k, l)\}$ (Corollaries 4.2.4 and 6.1.6), allows us to distinguish acceptable outcomes of D^m from those generated by trace distributions in $\mathsf{TrDist}(\mathcal{B}, k, l)$.

Before presenting the formal proofs, let us further motivate our approach by considering again the coin-flipping example. Suppose $\mathcal{A}$ is the fair coin and we conduct 100 experiments on $\mathcal{A}$. In this case, every outcome is just as likely as every other outcome. Yet a frequency vector close to $\langle 0.5, 0.5\rangle$ (for example $\langle 0.49, 0.51\rangle$) is much more likely to be observed than a frequency vector far away from $\langle 0.5, 0.5\rangle$ (for example $\langle 0.01, 0.99\rangle$). This is because there are many more outcomes with frequency $\langle 0.49, 0.51\rangle$ than there are outcomes with $\langle 0.01, 0.99\rangle$. As we increase the number of trials, this clustering effect intensifies and the probability of observing a frequency vector with large deviation becomes very small.

This simple idea also applies in the case of m independent coin flips, where each coin may have a different bias. This is formalized in the following lemma, which is an analog of the weak law of large numbers for independent Bernoulli variables.

Lemma 7.4.2. *Let $\alpha \in (0,1)$ and $\delta > 0$ be given. There exists $M \in \mathbb{N}$ such that for all $m \geq M$ and sequences $X_1, \ldots, X_m$ of independent Bernoulli variables,*

$$\mathbf{P}[|Z - \mathbf{E}\, Z| \geq \delta] \leq \alpha,$$

where $Z = \frac{1}{m}\sum_{i=1}^{m} X_i$ represents the success frequency in these m trials.

Proof. Take $M \geq \frac{1}{4\delta^2\alpha}$ and let $m, X_1, \ldots, X_m$ be given as stated. Assume that each Bernoulli variable X_i has parameter $p_i \in [0,1]$. First note that for all $p \in [0,1]$, $p(1-p) \leq \frac{1}{4}$. Then

$$\begin{aligned}\mathbf{Var}[Z] &= \mathbf{Var}[\frac{1}{m}\sum_{i=1}^{m} X_i] = \frac{1}{m^2}\sum_{i=1}^{m}\mathbf{Var}[X_i] \\ &= \frac{1}{m^2}\sum_{i=1}^{m} p_i(1-p_i) \leq \frac{1}{m^2}\sum_{i=1}^{m}\frac{1}{4} = \frac{1}{4m}.\end{aligned}$$

By Chebychev's inequality (Theorem 2.0.3), we have

$$\mathbf{P}[|Z - \mathbf{E}\,Z| \geq \delta] \leq \frac{1}{\delta^2}\,\mathbf{Var}[Z] \leq \frac{1}{\delta^2}\cdot\frac{1}{4m} \leq \frac{1}{\delta^2}\cdot\frac{1}{4M} \leq \frac{4\delta^2\alpha}{4\delta^2} = \alpha.$$

□

In our case, successes correspond to occurrences of a particular trace β: if the machine operates according to trace distributions $\vec{D}$, then each run i corresponds to a Bernoulli variable with parameter $\mathbf{P}_{D_i,k}[\beta]$ (see Section 7.3.2). Thus Lemma 7.4.2 gives the following corollary.

Corollary 7.4.3. *Given any $\delta > 0$, there exists $M \in \mathbb{N}$ such that for all $m \geq M$, $\beta \in \mathsf{Act}^{\leq k}$ and sequences $\vec{D}$ of trace distributions in $\mathsf{TrDist}(\mathcal{A})$,*

$$\mathbf{P}_{\vec{D},k}[\{O \in \mathcal{U} \mid |\,\mathsf{freq}(O)(\beta) - \mathbf{E}_{\beta}^{\vec{D},k}\,| \geq \delta\}] \leq \alpha.$$

Now we consider all sequences $\beta \in (\mathsf{Act}_l)^{\leq k}$ at the same time. This is where we must restrict to sequences over Act_l (rather than Act), since otherwise we are concerned with infinitely many β's.

Lemma 7.4.4. *Given any $\delta > 0$, there exists $M \in \mathbb{N}$ such that for all $m \geq M$ and sequences $\vec{D}$ of trace distributions in $\mathsf{TrDist}(\mathcal{A}, k, l)$,*

$$\mathbf{P}_{\vec{D},k}[\mathsf{freq}^{-1}(\mathbf{B}_{\delta}(\mathbf{E}^{\vec{D},k}))] \geq 1 - \alpha.$$

Proof. Let n be the cardinality of $(\mathsf{Act}_l)^{\leq k}$. By Corollary 7.4.3, we may choose M such that for all $m \geq M$, $\beta \in \mathsf{Act}^{\leq k}$ and sequences $\vec{D}$ of trace distributions in $\mathsf{TrDist}(\mathcal{A})$,

$$\mathbf{P}_{\vec{D},k}[\{O \in \mathcal{U} \mid |\,\mathsf{freq}(O)(\beta) - \mathbf{E}_{\beta}^{\vec{D},k}\,| \geq \delta\}] \leq \frac{\alpha}{n}.$$

Then for all $m \geq M$ and sequences $\vec{D}$, we have

$$\begin{aligned}
&\mathbf{P}_{\vec{D},k}[\mathsf{freq}^{-1}(\mathbf{B}_\delta(\mathbf{E}^{\vec{D},k}))] \\
&= \mathbf{P}_{\vec{D},k}[\{O \in \mathcal{U} \mid \forall \beta \mid \mathsf{freq}(O)(\beta) - \mathbf{E}^{\vec{D},k}_\beta \mid < \delta\}] && \text{definition of } \mathsf{dist} \\
&= 1 - \mathbf{P}_{\vec{D},k}[\{O \in \mathcal{U} \mid \exists \beta \mid \mathsf{freq}(O)(\beta) - \mathbf{E}^{\vec{D},k}_\beta \mid \geq \delta\}] \\
&\geq 1 - \sum_{\beta \in (\mathsf{Act}_l)^{\leq k}} \mathbf{P}_{\vec{D},k}[\{O \in \mathcal{U} \mid \mid \mathsf{freq}(O)(\beta) - \mathbf{E}^{\vec{D},k}_\beta \mid \geq \delta\}] \\
&\geq 1 - n \cdot \frac{\alpha}{n} = 1 - \alpha && \text{choice of } M
\end{aligned}$$

□

We are now ready for the proof of Theorem 7.4.1.

Proof of Theorem 7.4.1. The "only if" direction is trivial. For the converse, assume there is $D \in \mathsf{TrDist}(\mathcal{A}, k, l) \setminus \mathsf{TrDist}(\mathcal{A}, k, l)$. Let δ denote the distance between the point $\mathbf{P}_{D^m,k}$ and the set $\{\frac{1}{m}\sum_0^{m-1} \mathbf{P}_{\vec{K},k} \mid \vec{K} \in \mathsf{TrDist}(\mathcal{B}, k, l)\}$. By Corollaries 4.2.4 and 6.1.6, δ must be non-zero.

By Lemma 7.4.4, we can find $M_\mathcal{A}$ and $M_\mathcal{B}$ such that for all $m \geq \max(M_\mathcal{A}, M_\mathcal{B})$ and all sequences of trace distributions $\vec{K}$ in $\mathsf{TrDist}(\mathcal{B}, k, l)$,

$$\begin{aligned}
\mathbf{P}_{D^m,k}[\mathsf{freq}^{-1}(\mathbf{B}_{\frac{\delta}{3}}(\mathbf{E}^{D^m,k}))] &\geq 1 - \frac{\alpha}{2} > 1 - \alpha \\
\mathbf{P}_{\vec{K},k}[\mathsf{freq}^{-1}(\mathbf{B}_{\frac{\delta}{3}}(\mathbf{E}^{\vec{K},k}))]) &\geq 1 - \frac{\alpha}{2} > 1 - \alpha.
\end{aligned}$$

Therefore, we have

$$\mathsf{Obs}(D^m, k, \alpha) \subseteq \mathsf{freq}^{-1}(\mathbf{B}_{\frac{\delta}{3}}(\mathbf{E}^{D^m,k})) = \mathsf{freq}^{-1}(\mathbf{B}_{\frac{\delta}{3}}(\mathbf{P}_{D^m,k}))$$

and, for all sequences $\vec{K}$ in $\mathsf{TrDist}(\mathcal{B}, k, l)$,

$$\mathsf{Obs}(\vec{K}, k, \alpha) \subseteq \mathsf{freq}^{-1}(\mathbf{B}_{\frac{\delta}{3}}(\mathbf{E}^{\vec{K},k})) = \mathsf{freq}^{-1}(\mathbf{B}_{\frac{\delta}{3}}(\sum_0^{m-1} \frac{1}{m} \mathbf{P}_{\vec{K},k})).$$

Since $\mathsf{dist}(\mathbf{P}_{D^m,k}, \sum_0^{m-1} \frac{1}{m} \mathbf{P}_{\vec{K},k}) \geq \delta$, we have

$$\mathbf{B}_{\frac{\delta}{3}}(\mathbf{P}_{D^m,k}) \cap \mathbf{B}_{\frac{\delta}{3}}(\sum_0^{m-1} \frac{1}{m} \mathbf{P}_{\vec{K},k}) = \emptyset.$$

Therefore $\mathsf{Obs}(D^m, k, \alpha) \not\subseteq \mathsf{Obs}(\mathcal{B}, k, l, \alpha)$. □

Theorem 7.4.5. *Let $\mathcal{A}$ and $\mathcal{B}$ be image finite probabilistic automata and let $\alpha \in (0, 1)$ be given. We have $\mathcal{A} \leq_{\mathsf{td}} \mathcal{B}$ if and only if $\mathcal{A} \leq_\alpha \mathcal{B}$.*

Proof. We have the following chain of equivalences:

$$
\begin{aligned}
&\mathcal{A} \leq_{\mathsf{td}} \mathcal{B} \\
&\Leftrightarrow \mathcal{A} \leq_{\mathsf{td}}^{k,l} \mathcal{B} \text{ for all } k,l \in \mathbb{N} && \text{Theorem 5.3.15} \\
&\Leftrightarrow \mathsf{Obs}(\mathcal{A},k,l,m,\alpha) \subseteq \mathsf{Obs}(\mathcal{B},k,l,m,\alpha) \text{ for all } k,l,m \in \mathbb{N} && \text{Theorem 7.4.1} \\
&\Leftrightarrow \mathcal{A} \leq_{\alpha} \mathcal{B} && \text{definition of } \leq_{\alpha}
\end{aligned}
$$

□

7.5 Conclusions

The work on trace distribution machine presents a first step in developing statistical testing techniques for systems with nondeterministic behavior. We show that, under some appropriate finiteness assumptions, nondeterministic choices are "harmless". The rationale behind this statement is that we can view a nondeterministic choice among events as a weighted sum of those events, but with unknown weights. Therefore the behavior of a process is represented by a convex closed set of distributions, rather than a single distribution. This retains many of the nice properties of purely probabilistic processes and we are able to use hypothesis tests to characterize an existing semantic equivalence.

We see much potential in applying our ideas to "black-box" verification, where we have little or no control over the system of interest. Given such a system, one can construct a probabilistic automaton as the test hypothesis and use samples generated from the actual system to either accept or reject the hypothesis. This method provides rigorous guarantees regarding error probabilities.

We define very simple hypothesis tests in this chapter, because we do not have a special problem in mind and thus cannot make use of any domain knowledge. In practice, one can design more powerful tests (i.e., those that also control false positive errors) using specific properties of the distributions involved. Also, it may be sufficient to consider simple or one-sided hypotheses, for which standard methods exist for finding uniformly most powerful tests. (In contrast, our tests have composite and two-sided alternative hypotheses.)

Part II

Parallel Composition

8

Discussions

As explained in Chapter 1, nondeterminism is a convincing way of modeling uncertainties in a distributed environment, therefore we focus on models that combine nondeterministic and probabilistic choices. There we also talked about the notions of semantic equivalence and parallel composition, and why they are important for formal verification. In this chapter, we elaborate on these issues.

Let us start with a motivating example, taken from [Seg95b, LSV03].

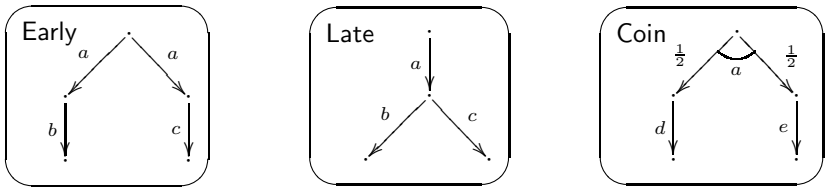

Figure 8.1: Probabilistic automata Early, Late and Coin

As its name suggests, automaton Early forces the choice between b and c at the very beginning of an execution, as we choose one of the two available a-transitions. On the other hand, automaton Late allows us to postpone this decision until after the a-transition. Automaton Coin will act as a context (or environment) for automata Early and Late. It has a probabilistic a-transition leading to a uniform distribution on two states, one of which enables a d-transition while the other enables an e-transition.

Consider the trace distribution semantics of [Seg95b], where each possible behavior is a probability space on the set of traces that is induced by a history-dependent adversary (cf. Chapter 3). It is easy to see that Early and Late are equivalent in this semantics, because every trace distribution of Early can be emulated in Late by postponing the choice between b and c and vice versa by making the same choice earlier.

However, under the parallel composition mechanism of [Seg95b, LSV03], the composites Early $\|$ Coin and Late $\|$ Coin are *not* trace distribution equivalent. Figure 8.2 below illustrates a trace distribution of Late $\|$ Coin that cannot be

mimicked in Early || Coin.

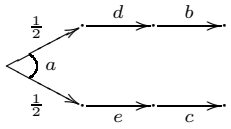

Figure 8.2: Non-substitutivity of trace distribution equivalence

This trace distribution assigns probability $\frac{1}{2}$ to each of these traces: adb and aec. It is induced by an adversary that chooses the b-transition in Late if and only if the random choice in Coin results in the left state. Such total correlations between actions d and b, and between actions e and c, cannot be achieved by an adversary for the composite Early || Coin, because doing so requires the adversary to predict the outcome of the coin toss in Coin.

This example shows that Segala's trace distribution semantics is *not* compositional: two equivalent automata behave differently when they are placed in the same context. The idea of a distinguishing context is further developed in [LSV03], showing that the difficulty with compositionality is in some sense inherent to trace-style semantics. The particular technical result is that the coarsest pre-congruence refining trace distribution preorder (induced by inclusion of sets of trace distributions) coincides with a probabilistic simulation preorder. Put more simply, compositionality forces us into the realm of branching-style semantics, where internal branching structures can be used to distinguish processes.

A good portion of this PhD work is directed towards a better understanding of the relationship between trace-style semantics and compositionality. We will argue that the result of [LSV03] is a consequence of the type of adversaries used to define trace distributions (cf. Sections 11.1 and 11.2). These adversaries have too much "observational power", in that they can use internal state and history of all components to make scheduling decisions. Although such adversaries (or policies) are standard in the setting of Markov decisions processes, we claim that they are not always meaningful when we try to model distributed processes.

To further motivate our interest in compositionality issues, we discuss in Section 8.1 the connection between modularity and compositionality. Section 8.2 then explains why we are interested primarily in trace-style semantics, such as the trace distribution semantics we saw in Chapters 3 and 7. In Section 8.3, we recall the interleaving interpretation of parallel composition and compare a few existing approaches in defining parallel composition for probabilistic processes. Finally, we outline in Section 8.4 our own approaches to the problem, which will be presented in detail in Chapters 10 and 11.

8.1 Modularity and Compositionality

In a distributed environment, system components communicate with each other via some well-defined mechanism. For instance, they access shared memory locations or send messages to each other on an asynchronous network. Apart from this communication, the evolution of each individual component is assumed to be independent from the behavior of other components. This independence assumption is a foundation for the so-called *modular* approaches to system development and analysis, where large and complex systems are decomposed into (or built up from) smaller and more tangible subsystems.

Modular methods are popular in various stages of the development process, including design, implementation and analysis. They often come in two flavors.

(i) *Vertical decomposition*: starting from some abstract specification of desired behavior, one can incrementally replace higher level specifications with lower level ones that contain more and more implementation details, provided each substitution involves only components that fall under certain behavioral equivalence (or inclusion) relation.

(ii) *Horizontal decomposition*: starting from a specification of desired behavior, one can decompose the required system into smaller, more tangible units that operate in parallel, provided the composition of these subsystems is behaviorally equivalent to (or included in) the original system.

In both cases, it is essential that our formal models come with a convincing notion of parallel composition, one that reflects our intuitions about independent evolution of parallel components. This requirement usually takes the form of a semantic compositionality theorem, relating the parallel composition operator to the relevant notion of system behavior. Here we distinguish "semantic" compositionality from the "syntactic" version, which simply states that a parallel composition operator is definable in a particular formal framework, without reference to the associated notion of observable behavior. Syntactic definability alone is not sufficient to validate modular reasoning as described in (i) and (ii) above.

Compositionality issues have been well-studied for nondeterministic models of distributed computation. For probabilistic processes, however, a careful study of parallel composition is somewhat harder to find. Many articles that do define parallel composition operators for probabilistic processes fail to provide an explanation of the intuitive meaning of these operators, much less a semantic compositionality theorem. A survey on such syntactic definability of parallel composition can be found in [SdV04]. In Section 8.3 below, we give a comparison of several frameworks that do address the issue of semantic compositionality.

8.2 Linear vs. Branching Semantics

In the literature, one can find a great variety of probabilistic process semantics, most of which are extensions of familiar semantic notions for labeled transition systems. Earlier proposals include probabilistic bisimulation [LS91] and testing preorder [YL92], followed by probabilistic simulation [SL95, LSV03], observational testing preorder [SV03] and many others.

Overall, semantics of a more branching character, such as bisimulation and simulation, have been more common than their linear counterparts, such as trace distribution preorder [Seg95b]. One likely reason is that a trace-style semantics requires one to resolve all nondeterministic choices by means of specifying an adversary. Once coupled with an adversary, a system becomes purely probabilistic and can be analyzed as a discrete-time Markov chain. Process behavior is then defined by quantifying over all possible adversaries. And, as we mentioned before, different classes of adversaries may be defined by varying their powers, leading to different notions of trace-style semantics.

In comparison, branching-style semantics are easier to define and more pleasant to work with. For instance, in order to establish bisimilarity between two processes, one simply defines a binary relation on states (or on state distributions) and proves that the proposed relation satisfies certain transfer properties. Most importantly, these transfer properties are typically *local*, concerning only the states in relation and their near successors.

Despite the apparent advantages of branching-style semantics, we remain interested in trace-style semantics for a number of reasons. First, many fundamental questions in verification are posed in terms of probabilities of observable events. For example, in a leader election algorithm, we may wish to calculate a lower bound for the probability of electing a leader during the first round. Questions such as this can be answered very naturally in a trace-style semantics, where we reason directly with probability distributions on traces. In contrast, a branching-style semantics only allows us to establish correspondences between specifications, without reference to probabilities of complex events (e.g., those that express causal dependencies between different actions). Indeed, branching-style semantics are often used as proof tools for trace-style semantics. For instance, a common technique for proving trace distribution inclusion is to establish the existence of a probabilistic simulation. In this sense, trace-style semantics are more fundamental compared to their branching counterparts.

Moreover, we believe that trace-style semantics capture more closely the idea of externally visible behavior, because they abstract away from state information. This is important in, for example, the setting of *black-box testing*, where we often have no convenient access to the actual architecture of a system and hence no state information is available.

Finally, as shown in [SAGG+93, DGRV00], we are often willing to say that a low-level automaton implements a high-level one, even if there exists no bisimulation

relation between them. In other words, trace-style equivalence is useful when bisimilarity is considered too fine.

8.3 Existing Approaches

Most existing proposals of parallel composition for probabilistic processes are based on the *interleaving* interpretation of parallel composition:

(i) every atomic step of a composite system is an atomic step of one of its components;

(ii) the scheduling among components is arbitrary[1].

Sometimes, components may synchronize on shared actions; that is, simultaneous executions are allowed if all transitions involved carry the same action label. However, distinct actions are never carried out at the same time. They are instead interleaved in arbitrary order as in (ii) above.

In other words, the parallel composition of two component executions is treated as a nondeterministic choice among all possible interleavings. This is generally regarded as a simplifying assumption, because it reduces the complexity of single-step evolution. In a probabilistic setting, some extra machinery is needed to resolve these nondeterministic choices introduced by parallel composition. Below we attempt to summarize a few prominent approaches.

– *Parameterized composition* [JLY01, DHK98]. Each (binary) composition operator $\|_p$ is parameterized with a real number $p \in [0,1]$, indicating the bias towards the left process. Sometimes a family of such operators are considered, with p ranging over some subset of $[0,1]$. Each $\|_p$ is essentially a very static adversary, resolving the choice between two processes in exactly the same way at every step.

  We find this approach unsatisfactory in two aspects. First, there is no obvious reason that one should privilege a single number $p \in [0,1]$ as a faithful representative of the underlying situation between two processes. Even if we consider all possible $\|_p$'s in the family, the quantification remains at a high level and does not enter into the reasoning of individual executions. That is, within a *single* execution, the left process will always receive bias p and the right process bias $1-p$, regardless of activities both in and around the processes.

  Second, these operators have some unpleasant technical properties. For example, the operator $\|_p$ is not associative except for degenerate cases in which p equals 0 or 1. And, unless we adopt the uniform distribution (i.e.,

[1] In order to remove trivial counterexamples of progress properties, one often needs to impose an appropriate fairness constraint on the scheduling. This is important in actual verification, although it is not treated explicitly in this thesis.

$p = \frac{1}{2}$), the operator $\|_p$ is not commutative. These properties are quite contrary to our intuitions about parallel processes.

- *Real-time delay* [WSS94]. Each state s of a process is associated with a delay parameter δ_s. Upon entering a state, every process draws a real-time delay from an exponential distribution with parameter δ_s. Among a group of parallel processes, the process with the shortest delay performs the next move. Since exponential distributions are memoryless, one can calculate the bias towards each component from the delay parameters of all components. In particular, one need not keep track of how long a component has already delayed in the currect state when a global transition takes place. Therefore, this approach essentially uses *history-independent* adversaries to resolve nondeterministic choices arising from parallel composition.

  Here we find it questionable that the delay patterns of processes can be universally characterized by exponential distributions. Furthermore, in actual applications, one must supply a parameter δ_s for each state s. It is again unclear how this can be done feasibly. Therefore, in our opinion, this theory has very limited utility in the setting of verification.

- *Compose-and-schedule* [DHK98, Seg95b]. Nondeterministic choices remain unresolved in the composition of parallel processes. Eventually, a possible behavior of the composite is obtained by specifying a *history-dependent* adversary, which has access to internal history of every component and is responsible for resolving *local* nondeterministic choices (i.e., those within each component) as well as *global* nondeterministic choices (i.e., those between parallel components).

Clearly, the last approach (compose-and-schedule) is the most robust, in that scheduling decisions may depend on dynamic behaviors of the entire system. However, as illustrated in Figure 8.2, history-dependent adversaries can create causal dependencies across different components, interfering with the modularity assumption mentioned in Section 8.1.

If we move to the less robust approach of history-independent scheduling, it is indeed possible to define a trace-style semantics [WSS94]. Nonetheless, we continue to explore alternative solutions, because (i) we are skeptical about the assumptions made in [WSS94], and (ii) we feel that there is ample middle ground between history-dependent scheduling of [Seg95b] and the history-independent version of [WSS94].

Finally, we mention an approach that deviates somewhat from the usual interleaving interpretation. In the models of [dAHJ01, vGSS95], components may make simultaneous moves, even if they are not involved in action synchronization. Assuming independence of coin tosses, the probability of a composite move can be calculated simply by multiplying the probabilities of all atomic moves involved. In this setting, it is also possible to obtain a compositional trace-style semantics [dAHJ01]. However, synchronous (or lock-step) execution is a nontrivial assumption in distributed computing [Lyn96]. Some important problems,

such as distributed consensus, are solvable in the synchronous setting but not in the asynchronous setting. The model of [dAHJ01] does not provide a very natural setting for analyzing algorithms designed for asynchronous networks.

8.4 Our Approaches

Our main goal is to develop a modeling framework that (i) supports compositional reasoning and (ii) is sufficiently robust for modeling randomized computation in a distributed environment. We come with two proposals in this thesis, both of which are shown to have a compositional trace-style semantics.

The first proposal, presented in Chapter 10, uses the idea of distributed scheduling. A token-structure is imposed among processes in order to eliminate global scheduling conflicts. In this way, scheduling decisions are entirely local: they are based on local information and they affect local actions.

Chapter 11 presents the second proposal, which returns to centralized scheduling with a version of the so-called *partial-information* adversaries. These adversaries may use dynamic information in their scheduling decisions, but certain aspects of the history and/or current state are hidden from them. In particular, we impose additional axioms on the transition structures, so that local information is hidden from the adversary.

We stress that Chapters 10 and 11 are independent, although both rely on basic definitions of Probabilistic I/O Automata given in Chapter 9.

9

Probabilistic I/O Automata

In this chapter, we extend the probabilistic automata framework (cf. Chapter 3) with input/output (I/O) distinction, following the tradition of Input/Output Automata (IOA) of Lynch and Tuttle [LT89]. This will serve as the basis of our developments in Chapters 10 and 11.

Variations of probabilistic I/O automata have appeared in many places, for example, [PSL00, WSS94, BPW04b]. Although all of these models go by the same name, the actual definitions diverge significantly. The present chapter aims to provide a clean and unifying formulation.

9.1 Basic Definitions

We assume a fixed, countably infinite alphabet Act of action symbols. Inspired by [vGSS95], we define reactive and generative transition structures as follows.

Definition 9.1.1. Let S be a set of states and let $X \subseteq \mathsf{Act}$ be given.

(i) A *reactive* transition structure on $\langle S, X \rangle$ is a function

$$\mathbf{R} : S \times X \to \mathcal{P}(\mathsf{Disc}(S)).$$

(ii) A *generative* transition structure on $\langle S, X \rangle$ is a function

$$\mathbf{G} : S \to \mathcal{P}(\mathsf{Disc}(X \times S)).$$

A state $s \in S$ *blocks* (input) action $a \in X$ if $\mathbf{R}(\langle s, a \rangle) = \emptyset$. And s is said to be *quiescent* if $\mathbf{G}(s) = \emptyset$.

A reactive transition structure $\mathbf{R}$ describes a system that reacts to input signals. Given a state s and an action a, $\mathbf{R}(\langle s, a \rangle)$ yields a set of discrete distributions on S. Thus we allow non-deterministic choices over possible distributions on end states, while each such distribution specifies an effect of randomization on system evolution. We use variables μ, ν, etc., for these state distributions. Figure 9.1 below illustrates two such reactive systems.

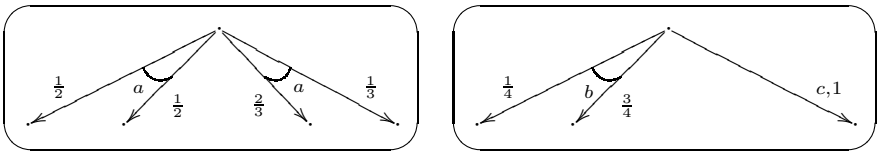

Figure 9.1: Examples of Reactive Transition Systems

On the other hand, a generative transition structure $\mathbf{G}$ describes a system that evolves in an active fashion. That is, every state s enables a (possibly empty) set of *transition bundles*, where each bundle is a discrete distribution on $\mathsf{Act} \times S$. Again, we have non-deterministic choices over bundles, while each bundle specifies a random choice over next transitions. We use variables f, g, etc., for these transition bundles. Figure 9.2 illustrates two such generative systems.

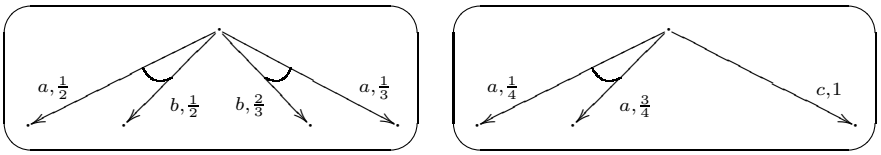

Figure 9.2: Examples of Generative Transition Systems

Next, we use I/O distinction to combine the reactive and generative system types, yielding our notion of probabilistic I/O automata. Notice that we impose I/O distinction not only on the action signature, but also on the transition structure.

Definition 9.1.2. A *probabilistic I/O automaton (PIOA) A* is a tuple

$$\langle S_A, s_A^0, I_A, O_A, H_A, \mathbf{R}_A, \mathbf{G}_A \rangle$$

where:

1. S_A is a set of states with initial state $s_A^0 \in S_A$;
2. $\{I_A, O_A, H_A\}$ are pairwise disjoint subsets of Act, referred to as: *input*, *output* and *hidden* actions, respectively;
3. $\mathbf{R}_A$ is a reactive transition structure on $\langle S_A, I_A \rangle$ and $\mathbf{G}_A$ is a generative transition structure on $\langle S_A, O_A \cup H_A \rangle$.

The automaton A is said to be *closed* if I_A is empty and *open* otherwise. As usual, input and output actions are *visible*, while output and hidden actions are *locally controlled*. The union $I_A \cup O_A \cup H_A$ is often denoted by Act_A. Notice

that we omit the input enabling axiom of IOA (i.e., all inputs are accepted at every state). This flexibility facilitates our introduction of switched PIOAs in Chapter 10.

For simplicity, we will assume that S_A is countable. Also we assume that $\mathbf{R}_A(\langle s, a\rangle)$ and $\mathbf{G}_A(s)$ are countable for all $s \in S_A$ and $a \in I_A$.

9.1.1 Branches

Recall from Chapter 3 that a path in a probabilistic automaton is a possibly infinite sequence of the form $s_0 a_1 \mu_1 s_1 \ldots$, satisfying the obvious reachability conditions. In the present development, paths are enriched with additional information from the reactive and generative transition structures. They are also given a new name: *execution branches*. This is in keeping with the terminology of *execution trees*, which we introduce in Definition 9.3.3 below. To improve readability, symbols appearing on a branch are separated by dots.

Definition 9.1.3. Let A be a PIOA and let $s \in S_A$ be given. We use joint recursion to define the set of *execution branches* from s, denoted $\mathsf{Bran}(s)$, together with the function $\mathsf{last} : \mathsf{Bran}(s) \to S_A$.

- The length-one sequence containing s (written $\underline{s}$) is in $\mathsf{Bran}(s)$ and is called the *empty* branch. We define $\mathsf{last}(\underline{s}) := s$.
- For all $r \in \mathsf{Bran}(s)$, $a \in I_A$, $\mu \in \mathbf{R}_A(\langle \mathsf{last}(r), a\rangle)$ and $s' \in \mathsf{Supp}(\mu)$, we have $r.a.\mu.s' \in \mathsf{Bran}(s)$. Moreover, $\mathsf{last}(r.a.\mu.s') := s'$.
- For all $r \in \mathsf{Bran}(s)$, $f \in \mathbf{G}_A(\mathsf{last}(r))$ and $\langle a, s'\rangle \in \mathsf{Supp}(f)$, we have $r.f.a.s' \in \mathsf{Bran}(s)$. Moreover, $\mathsf{last}(r.f.a.s') := s'$.

A branch r is said to be a *one-step prefix* of another branch r', denoted $r \sqsubseteq^1 r'$, if r' if of the form $r.a.\mu.s'$ or $r.f.a.s'$.

The *trace* of a branch r is defined in the usual way:

- $\mathsf{tr}(\underline{s}) := \epsilon$,
- $\mathsf{tr}(r.a.\mu.s') := \mathsf{tr}(r).a$ (in this case $a \in I_A$), and
- $\mathsf{tr}(r.f.a.s')$ is $\mathsf{tr}(r).a$ if $a \in O_A$ and $\mathsf{tr}(r)$ if $a \in H_A$.

If $r \in \mathsf{Bran}(s^0_A)$, it is said to be *rooted*. We also We write $\mathsf{Bran}(A)$ for $\mathsf{Bran}(s^0_A)$.

Notice that execution branches are always finite, because $\mathsf{Bran}(s)$ is given by a recursive definition. Also, since A has countable state space and transition structure, we know that $\mathsf{Bran}(s)$ is countable.

An infinite branch from s is simply an infinite subset of $\mathsf{Bran}(s)$ that is linearly ordered by the prefix ordering $\sqsubseteq$ on sequences. We write $\mathsf{Bran}^{\leq\omega}(s)$ for the set of finite and infinite branches from s. Similarly, $\mathsf{Bran}^{\leq\omega}(A) := \mathsf{Bran}^{\leq\omega}(s^0_A)$.

Minimal Branches

Below we introduce an important class of branches: minimal branches. Minimality is important because distinct minimal branches with the same trace always represent mutually exclusive events. When we consider branches that are not necessarily minimal, distinct branches with the same trace may be prefix-related (i.e., one event is strictly included in the other).

Definition 9.1.4. Let A be a PIOA and let $s \in S_A$ be given.

- A branch $r \in \mathsf{Bran}(s)$ is said to be *minimal* if every proper prefix of r in $\mathsf{Bran}(s)$ has a strictly shorter trace. We write $\mathsf{Bran}_{\min}(s)$ for the set of minimal branches in $\mathsf{Bran}(s)$ and $\mathsf{Bran}_{\min}(A)$ for $\mathsf{Bran}_{\min}(s^0_A)$.
- For each $\alpha \in \mathsf{Act}_A^{<\omega}$, let $\mathsf{tr}^{-1}(\alpha)$ denote the set $\{r \in \mathsf{Bran}(A) \mid \mathsf{tr}(r) = \alpha\}$ and let $\mathsf{tr}^{-1}_{\min}(\alpha)$ denote the set $\{r \in \mathsf{Bran}_{\min}(A) \mid \mathsf{tr}(r) = \alpha\}$.

Notice, the empty branch $\underline{s^0_A}$ is minimal. For non-empty r, it is minimal if and only if its last action label is visible.

Reachability

It is often convenient to speak of reachability with non-zero probability, abstracting way from the actual probability distributions. Given $s, s' \in S_A$ and $a \in \mathsf{Act}_A$, we say that s' is *reachable* (in one step) from s via action a, denoted $s \xrightarrow{a} s'$, just in case:

- $s.a.\mu.s' \in \mathsf{Bran}(s)$ for some μ, or
- $s.f.a.s' \in \mathsf{Bran}(s)$ for some f.

Similarly, a state s is *reachable* if there exists $r \in \mathsf{Bran}(A)$ such that $\mathsf{last}(r) = s$.

9.1.2 Sub-Automata

Given two PIOAs A and B with the same action signature, one can speak of A being a *sub-automaton* of B. Intuitively, it means A can be obtained from B by removing certain states and/or transitions. This is made precise in Definition 9.1.5 below.

Definition 9.1.5. Suppose A and B are PIOAs with the same action signature $\{I, O, H\}$. We say that A is a *sub-automaton* of B, denoted $A \subseteq B$, if

- $S_A \subseteq S_B$ and $s^0_A = s^0_B$;
- for all $s \in S_A$ and $a \in I$, $\mathbf{R}_A(\langle s, a\rangle) \subseteq \mathbf{R}_B(\langle s, a\rangle)$ and $\mathbf{G}_A(s) \subseteq \mathbf{G}_B(s)$.

9.2 Composition of PIOAs

We start with the usual notion of compatibility: two PIOAs A and B are said to be *compatible* if $O_A \cap O_B = \mathsf{Act}_A \cap H_B = \mathsf{Act}_B \cap H_A = \emptyset$. In other words, we have (i) A and B have pairwise disjoint sets of locally controlled actions, and (ii) A may not synchronize with hidden actions of B and vice versa. These conditions ensures that A and B are sufficiently independent. In the sequel, we require that automata under parallel composition must be pairwise compatible.

9.2.1 Examples

Let us give some examples to illustrate how we intend to compose reactive and generative transition structures. Consider automata A, B and C in Figure 9.3 and assume that action a is in the signatures of all three automata, while b is in the signatures of B and C only.

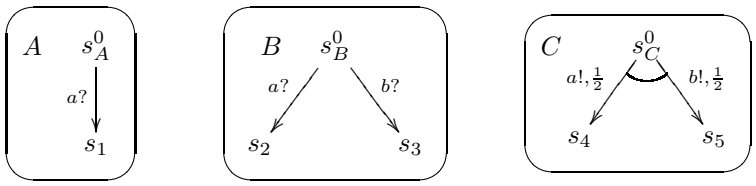

Figure 9.3: Automata A, B and C

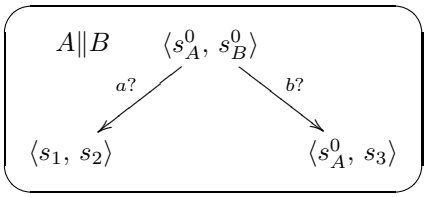

Figure 9.4: Parallel Composite$A \| B$

First we consider $A \| B$. Here both A and B are reactive, therefore their composite is constructed in a straightforward manner via synchronization of shared actions. In particular, if input a is provided, then both A and B react and move to corresponding new states. If, on the other hand, b is provided, then only B reacts and A simply stutters (i.e., no transition takes place). This is illustrated in Figure 9.4.

Next we add C to the parallel composition. Now the composite exhibits generative behavior, because both actions a and b are locally controlled by C. In

$A\|B\|C$, these action each take place with probability $\frac{1}{2}$, just as in C. If a is chosen, then all three components participate in the transition. Otherwise, b is chosen and only B and C participate. This is illustrated in Figure 9.5.

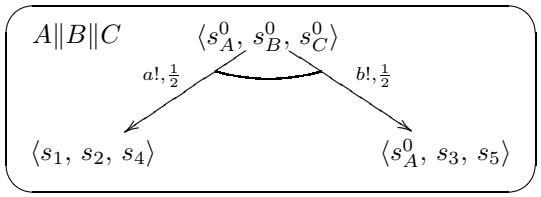

Figure 9.5: Parallel Composite $A\|B\|C$

Finally, we consider the case where two or more automata enable locally controlled bundles. Figure 9.6 illustrates an automaton D with a single output bundle labeled d. When C and D are composed, there is simply a nondeterministic choice between the two bundles.

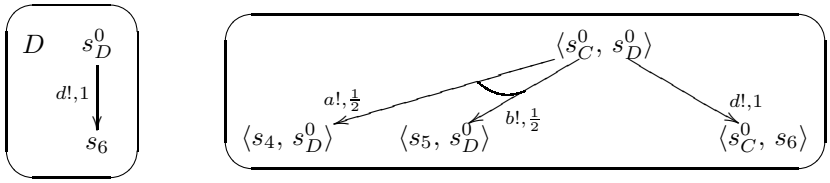

Figure 9.6: Automata D and $C\|D$

Despite their simplicity, these examples demonstrate our basic idea of parallel composition: in each step of the composite, at most one component behaves actively, while all others react to the action performed by the active component. Nondeterministic choices are used to model situations in which multiple components exhibit generative behavior.

9.2.2 Composition

We continue with formal definitions. First recall from Chapter 2 the definition of the product of a family of probability distributions.

Let $\{A_i \mid 1 \leq i \leq n\}$ be a set of pairwise compatible PIOAs. For readability, we replace all subscripts A_i with i. (The same convention will be adopted throughout the rest of this thesis.) The *parallel composite* of $\{A_i \mid 1 \leq i \leq n\}$, denoted $\uparrow\uparrow_{i=1}^{n} A_i$, is the PIOA B with the following state space and action signature:

1. $S_B := \prod_{i=1}^n S_i$ with $s_B^0 := \langle s_1^0, \ldots, s_n^0 \rangle$;
2. $I_B := \bigcup_{i=1}^n I_i \setminus \bigcup_{i=1}^n O_i$, $O_B := \bigcup_{i=1}^n O_i$, and $H_B := \bigcup_{i=1}^n H_i$;

The reactive transition structure $\mathbf{R}_B$ and the generative transition structure $\mathbf{G}_B$ are given in Definition 9.2.1 and Definition 9.2.2, respectively.

Definition 9.2.1. Let $\vec{s} \in S_B$ and $a \in I_B$ be given. We define $\mathbf{R}_B(\langle \vec{s}, a \rangle) \subseteq \mathsf{Disc}(S_B)$ to be the set of all discrete distributions of the form $\prod_{i=1}^n \mu_i$ for some family $\vec{\mu} \in \prod_{i=1}^n \mathsf{Disc}(S_i)$ satisfying:

- if $a \notin I_i$, then $\mu_i = \mathsf{Dirac}(s_i)$;
- otherwise, $\mu_i \in \mathbf{R}_i(\langle s_i, a \rangle)$.

In other words, each process A_i stutters if the given input a is not in the signature of A_i. Otherwise, a distribution μ_i is selected nondeterministically from $\mathbf{R}_i(\langle s_i, a \rangle)$ and a state t_i is selected randomly according to μ_i. A product construction on state distributions μ_i then yields a typical member of $\mathbf{R}_B(\langle \vec{s}, a \rangle)$. This reflects our assumption that processes evolve independently,

The definition of $\mathbf{G}_B$ for $B = \uparrow\uparrow_{i=1}^n A_i$ is slightly more complicated, where exactly one component B_j is generative and all others are reactive.

Definition 9.2.2. Let $\vec{s} \in S_B$ and $1 \leq j \leq n$ be given. Let N_j denote the index set $(O_B \cup H_B) \times \{i \mid 1 \leq i \leq n, i \neq j\}$. Suppose we have a transition bundle $g_j \in \mathbf{G}_j(s_j)$ and a family $\vec{\mu} \in \prod_{\langle a, i \rangle \in N_j} \mathsf{Disc}(S_i)$ of state distributions so that: for all $\langle a, i \rangle \in N_j$,

- if $a \notin I_i$, then $\mu_{a,i} = \mathsf{Dirac}(s_i)$;
- otherwise, $\mu_{a,i} \in \mathbf{R}_i(\langle s_i, a \rangle)$.

Then g_j and $\vec{\mu}$ are said to *generate* the following distribution f on $(O_B \cup H_B) \times S_B$: for all $\langle a, \vec{t} \rangle$,

$$f(\langle a, \vec{t} \rangle) := g_j(\langle a, t_j \rangle) \cdot \prod_{i \neq j} \mu_{a,i}(t_i).$$

With slight abuse of notation, we write $f = g_j \times \prod_{\langle a, i \rangle \in N_j} \mu_{a,i}$.

We define $\mathbf{G}_B^j(\vec{s}) \subseteq \mathsf{Disc}((O_B \cup H_B) \times S_B)$ to be the set of all bundles f so that f is generated by some $g_j \in \mathbf{G}_j(s_j)$ and some $\vec{\mu} \in \prod_{\langle a, i \rangle \in N_j} \mathsf{Disc}(S_i)$ satisfying the conditions above. Then $\mathbf{G}_B(\vec{s}) := \bigcup_{1 \leq j \leq n} \mathbf{G}_B^j(\vec{s})$.

Here an active component A_j is selected, as well as a transition bundle g_j locally controlled by A_j. Both choices are nondeterministic. Once g_j is specified, a pair $\langle a, t_j \rangle$ is chosen *randomly* according to g_j. The other processes A_i either stutter or react to the action performed by A_j, whichever dictated by their

action signatures. Note that the choice of the family $\vec{\mu}$ is nondeterministic and is independent from the particular pair $\langle a, t_j \rangle$ drawn from g_j.

Lemma 9.2.1 below shows that the new bundles f constructed in Definition 9.2.2 are in fact well-defined discrete distributions.

Lemma 9.2.1. *The bundle f in Definition 9.2.2 is well-defined.*

Proof. We need to verify that f is a discrete distribution on $(O_B \cup H_B) \times S_B$. First consider fixed $a \in O_j \cup H_j$. By the definition of f, we have

$$\sum_{\vec{t} \in S_B} f(\langle a, \vec{t} \rangle) = \sum_{t_1 \in S_1} \cdots \sum_{t_n \in S_n} g_j(\langle a, t_j \rangle) \cdot \prod_{i \neq j} \mu_{a,i}(t_i).$$

We can rearrange the sums and factor out $g_j(\langle a, t_j \rangle)$ to obtain:

$$\sum_{t_j \in S_j} g_j(\langle a, t_j \rangle) \cdot \Big(\sum_{t_1 \in S_1} \cdots \sum_{t_{j-1} \in S_{j-1}} \sum_{t_{j+1} \in S_{j+1}} \cdots \sum_{t_n \in S_n} \prod_{i \neq j} \mu_{a,i}(t_i) \Big).$$

Since each $\mu_{a,i}$ is a discrete distribution on S_i, an easy inductive argument shows that

$$\sum_{t_1 \in S_1} \cdots \sum_{t_{j-1} \in S_{j-1}} \sum_{t_{j+1} \in S_{j+1}} \cdots \sum_{t_n \in S_n} \prod_{i \neq j} \mu_{a,i}(t_i) = 1.$$

Then we have $\sum_{\vec{t} \in S_B} f(\langle a, \vec{t} \rangle) = \sum_{t_j \in S_j} g_j(\langle a, t_j \rangle)$.

Now notice that $f(\langle a, \vec{t} \rangle) = 0$ whenever $a \notin O_j \cup H_j$. Therefore,

$$\begin{aligned} \sum_{\langle a, \vec{t} \rangle \in (O_B \cup H_B) \times S_B} f(\langle a, \vec{t} \rangle) &= \sum_{a \in O_j \cup H_j} \sum_{\vec{t} \in S_B} f(\langle a, \vec{t} \rangle) \\ &= \sum_{a \in O_j \cup H_j} \sum_{t_j \in S_j} g_j(\langle a, t_j \rangle) && \text{calculation above} \\ &= 1 && g_j \text{ discrete distribution} \end{aligned}$$

Therefore f is a discrete distribution on $(O_B \cup H_B) \times S_B$. □

This completes the definition of parallel composition for PIOAs. We write $\Vert^n$ for the n-ary composition operator and, when $n = 2$, we omit the superscript and use infix notation. Due to symmetries in our definitions, it is easy to see that $\Vert$ is commutative. We claim that $\Vert$ is also associative, because both $(A \Vert B) \Vert C$ and $A \Vert (B \Vert C)$ are isomorphic to $\Vert^3 \{A, B, C\}$. We omit the details.

Finally, we make a remark on synchronization deadlocks in parallel composition. These occur when two components can decide individually whether they are willing to synchronize. For example, consider a case in which component A sends a message to component B with probability p, but B accepts the message with some probability $q < 1$. Then, with probability $p(1 - q)$, there is a communication failure.

In the literature, one typically uses a normalization mechanism to collect and redistribute deadlock probabilities. Here we are able to avoid this cumbersome step, by imposing I/O distinction directly on the transition structures. The definition of reactive structures (cf. Definition 9.1.1) enforces that every input is either completely blocked (i.e., $\mathbf{R}(\langle s, a\rangle) = \emptyset$) or received with probability 1.

In practice, one may wish to add an input enabling axiom to uniformly disallow input blocking. This is done, for example, in Chapter 11 of the current thesis. For generality of the underlying PIOA model, we have decided against the inclusion of such an axiom. Indeed, in the switched PIOA model of Chapter 10, input blocking is an intended feature of an active component (i.e., one possessing the unique activity token).

9.2.3 Projection

In Chapter 2, we described projection operators for discrete distributions on a product space. These extend immediately to projection operators for composite transition bundles of Definitions 9.2.1.

Definition 9.2.3. Let $\{A_i \mid 1 \leq i \leq n\}$ be a set of pairwise compatible PIOAs and let B denote $\Uparrow_{i=1}^{n} A_i$. Let $\vec{s} \in S_B$, $a \in I_B$ and $\mu \in \mathbf{R}_B(\vec{s}, a)$ be given. The *jth-projection* of μ is the discrete distribution $\mathsf{proj}_j(\mu)$ on S_i, as defined in Chapter 2.

Using the same idea, we define projection on bundles given in Definition 9.2.2. To avoid confusion, we write proj_L for the left projection in a binary Cartesian product, instead of proj_1.

Definition 9.2.4. Let $\{A_i \mid 1 \leq i \leq n\}$ be a set of pairwise compatible PIOAs and let B denote $\Uparrow_{i=1}^{n} A_i$. Let $\vec{s} \in S_B$ and $f \in \mathbf{G}_B(\vec{s})$ be given. Let j be the unique index such that $\mathsf{proj}_L(\mathsf{Supp}(f)) \subseteq O_j \cup H_j$ (equivalently, $f \in \mathbf{G}_B^j(\vec{s})$). The *$j$th-projection* of f, denoted $\mathsf{proj}_j(f)$, is the discrete distribution on $(O_j \cup H_j) \times S_j$ given by:

$$\mathsf{proj}_j(f)(\langle a, t\rangle) := \sum_{\vec{t} \in S_B \,:\, t_j = t} f(\langle a, \vec{t}\rangle).$$

For every $a \in \mathsf{proj}_L(\mathsf{Supp}(f))$ and $i \neq j$, the *$\langle a, i\rangle$th-projection* of f, denoted $\mathsf{proj}_{a,i}(f)$, is the discrete distribution on S_i given by:

$$\mathsf{proj}_{a,i}(f)(t) := \frac{\sum_{\vec{t} \in S_B \,:\, t_i = t,\, t_j = u} f(\langle a, \vec{t}\rangle)}{\mathsf{proj}_j(f)(\langle a, u\rangle)},$$

where u is any state in S_j such that $\mathsf{proj}_j(f)(\langle a, u\rangle) \neq 0$.

Lemmas 9.2.2 and 9.2.3 below show that these projection operators are in fact well-defined.

Lemma 9.2.2. *The distribution* $\mathsf{proj}_j(f)$ *in Definition 9.2.4 is well-defined and is in* $\mathbf{G}_j(s_j)$.

Proof. By the definition of $\mathbf{G}_B(\vec{s})$, we may choose $g_j \in \mathbf{G}_j(s_j)$ such that f is generated by g_j. It suffices to show $\mathsf{proj}_j(f) = g_j$. Let $\langle a, t\rangle \in (O_j \cup H_j) \times S_j$ be given. By definition,

$$\mathsf{proj}_j(f)(\langle a, t\rangle) = \sum_{\vec{t} \in S_B \,:\, t_j = t} f(\langle a, \vec{t}\rangle) = \sum_{\vec{t} \in S_B \,:\, t_j = t} g_j(\langle a, t_j\rangle) \cdot \prod_{i \neq j} \mu_{a,i}(t_i).$$

We can rearrange the sums and factor out $g_j(\langle a, t_j\rangle)$ to obtain:

$$\mathsf{proj}_j(f)(\langle a, t\rangle) = g_j(\langle a, t\rangle) \cdot \Big(\sum_{t_1 \in S_1} \cdots \sum_{t_{j-1} \in S_{j-1}} \sum_{t_{j+1} \in S_{j+1}} \cdots \sum_{t_n \in S_n} \prod_{i \neq j} \mu_{a,i}(t_i)\Big).$$

Since every $\mu_{a,i}$ is a discrete distribution on S_i, the second factor equals 1. Hence $\mathsf{proj}_j(f)(\langle a, t\rangle) = g_j(\langle a, t\rangle)$. □

Lemma 9.2.3. *The distribution* $\mathsf{proj}_{a,i}(f)$ *in Definition 9.2.4 is well-defined. Also, if* $a \in I_i$, *then* $\mathsf{proj}_{a,i}(f) \in \mathbf{R}_i(\langle s_i, a\rangle)$; *otherwise,* $\mathsf{proj}_{a_i}(f) = \mathsf{Dirac}(s_i)$.

Proof. By the definition of $\mathbf{G}_B(\vec{s})$, we may choose $\mu_{a,i} \in \mathsf{Disc}(S_i)$ and $g_j \in \mathbf{G}_j(s_j)$ such that f is generated by $\mu_{a,i}$ and g_j. It suffices to show $\mathsf{proj}_{a,i}(f) = \mu_{a,i}$. Let $t \in S_i$ be given. By definition, $\mathsf{proj}_{a,i}(f)(t)$ equals

$$\frac{\sum_{\vec{t} \in S_B \,:\, t_i = t,\, t_j = u} f(\langle a, \vec{t}\rangle)}{\mathsf{proj}_j(f)(\langle a, u\rangle)} = \frac{\sum_{\vec{t} \in S_B \,:\, t_i = t,\, t_j = u} (g_j(\langle a, t_j\rangle) \cdot \prod_{k \neq j} \mu_{a,k}(t_k))}{\mathsf{proj}_j(f)(\langle a, u\rangle)}.$$

Factoring out $g_j(\langle a, u\rangle)$ and $\mu_{a,i}(t)$, the numerator becomes

$$g_j(\langle a, u\rangle) \cdot \mu_{a,i}(t) \cdot \sum_{\vec{t} \in S_B \,:\, t_i = t,\, t_j = u} \prod_{k \neq i,j} \mu_{a,k}(t_k)).$$

Again the third factor is easily seen to be 1 and hence the numerator equals $g_j(\langle a, u\rangle) \cdot \mu_{a,i}(t)$. Moreover, we saw in the proof of Lemma 9.2.2 that $\mathsf{proj}_j(f) = g_j$, therefore the denominator equals $g_j(\langle a, u\rangle)$. Now we have

$$\mathsf{proj}_{a,i}(f)(t) = \frac{g_j(\langle a, u\rangle) \cdot \mu_{a,i}(t)}{g_j(\langle a, u\rangle)} = \mu_{a,i}(t).$$

Notice we haven't used any additional assumption on u, therefore the equality holds regardless of the choice of u. □

Given these projection operators on transition bundles, it is straightforward to define projection on execution branches.

Definition 9.2.5. Let $\{A_i \mid 1 \leq i \leq n\}$ be a set of pairwise compatible PIOAs and let B denote $\uparrow\!|_{i=1}^{n} A_i$. Let $\vec{s} \in S_B$ and $1 \leq i \leq n$ be given. We define, recursively, the *i-th projection* operator on $\mathsf{Bran}(\vec{s})$ as follows:

- $\mathsf{proj}_i(\langle \underline{s_1^0, \ldots, s_n^0} \rangle) := \underline{s_i^0}$;
- $\mathsf{proj}_i(r.a.\mu.\vec{t})$ equals
  - $\mathsf{proj}_i(r).a.\mathsf{proj}_i(\mu).t_i$, if $a \in I_i$;
  - $\mathsf{proj}_i(r)$, otherwise;
- $\mathsf{proj}_i(r.f.a.\vec{t})$ equals
  - $\mathsf{proj}_i(r).\mathsf{proj}_i(f).a.t_i$, if i is the unique index with $\mathsf{proj}_L(\mathsf{Supp}(f)) \subseteq O_i \cup H_i$;
  - $\mathsf{proj}_i(r).a.\mathsf{proj}_{a,i}(f).t_i$, if $a \in I_i$;
  - $\mathsf{proj}_i(r)$, otherwise.

These projected branches are well-defined by virtue of Lemma 9.2.4 below.

Lemma 9.2.4. *Let $1 \leq i \leq n$ be given. For all $q \in \mathsf{Bran}(\vec{s})$, we have*

1. $\mathsf{proj}_i(\mathsf{last}(q)) = \mathsf{last}(\mathsf{proj}_i(q))$;
2. *if q is $r.a.\mu.\vec{t}$ and $a \in I_i$, then $\mathsf{proj}_i(\mu) \in \mathbf{R}_i(\langle \mathsf{last}(\mathsf{proj}_i(r)), a \rangle)$ and $t_i \in \mathsf{Supp}(\mathsf{proj}_i(\mu))$;*
3. *if q is $r.f.a.\vec{t}$ and $a \in O_i \cup H_i$, then $\mathsf{proj}_i(f) \in \mathbf{G}_i(\mathsf{last}(\mathsf{proj}_i(r)))$ and $\langle a, t_i \rangle \in \mathsf{Supp}(\mathsf{proj}_i(f))$;*
4. *if q is $r.f.a.\vec{t}$ and $a \in I_i$, then $\mathsf{proj}_{a,i}(f) \in \mathbf{R}_i(\langle \mathsf{last}(\mathsf{proj}_i(r)), a \rangle)$ and $t_i \in \mathsf{Supp}(\mathsf{proj}_{a,i}(f))$;*

Proof. We proceed by induction on the length of r. The base case is trivial.

Consider a branch of the form $r.a.\mu.\vec{t}$ and let $\vec{u}$ denote $\mathsf{last}(r)$. By the induction hypothesis, we have $\mathsf{proj}_i(\mathsf{last}(r)) = u_i = \mathsf{last}(\mathsf{proj}_i(r))$. Recall that $\mu = \prod_{i=1}^{n} \mathsf{proj}_i(\mu_i)$. We have two cases.

- $a \in I_i$. Then by Definition 9.2.1 we have $\mathsf{proj}_i(\mu) \in \mathbf{R}_i(\langle u_i, a \rangle)$. Since $\vec{t} \in \mathsf{Supp}(\mu)$, it must be that $t_i \in \mathsf{Supp}(\mathsf{proj}_i(\mu))$. Moreover, $\mathsf{proj}_i(\mathsf{last}(q)) = t_i = \mathsf{last}(\mathsf{proj}_i(q))$.
- $a \notin I_i$. Then by Definition 9.2.1 we have $\mathsf{proj}_i(\mu) = \mathsf{Dirac}(u_i)$. Since $\vec{t} \in \mathsf{Supp}(\mu)$, it must be that $t_i = u_i$. Therefore $\mathsf{proj}_i(\mathsf{last}(q)) = t_i = u_i = \mathsf{last}(\mathsf{proj}_i(r)) = \mathsf{last}(\mathsf{proj}_i(q))$.

Now we consider a branch of the form $r.f.a.\vec{t}$. Again, let $\vec{u}$ denote $\mathsf{last}(r)$ and we have $\mathsf{proj}_i(\mathsf{last}(r)) = u_i = \mathsf{last}(\mathsf{proj}_i(r))$ by the induction hypothesis. By Definition 9.2.2 we may choose unique j such that $f = g_j \times \prod_{\langle a, i \rangle \in N_j} \mu_{a,i}$ for some $g_j \in \mathbf{G}_j(u_j)$ and family $\{\mu_{a,i}\}_{\langle a, i \rangle \in N_j} \in \prod_{\langle a, i \rangle \in N_j} \mathsf{Disc}(S_i)$. We have three cases.

– $i = j$. Then we have $\mathsf{proj}_i(f) = g_i \in \mathbf{G}_i(u_i)$. Since $\langle a, \vec{t} \rangle \in \mathsf{Supp}(f)$, it must be that $\langle a, t_i \rangle \in \mathsf{Supp}(g_i) = \mathsf{Supp}(\mathsf{proj}_i(f))$. Moreover,

$$\mathsf{proj}_i(\mathsf{last}(q)) = t_i = \mathsf{last}(\mathsf{proj}_i(q)).$$

– $i \neq j$ and $a \in I_i$. Then by Definition 9.2.2 we have $\mathsf{proj}_{a,i}(f) \in \mathbf{R}_i(\langle u_i, a \rangle)$. Since $\langle a, \vec{t} \rangle \in \mathsf{Supp}(f)$, it must be that $t_i \in \mathsf{Supp}(\mathsf{proj}_{a,i}(f))$. Moreover, $\mathsf{proj}_i(\mathsf{last}(q)) = t_i = \mathsf{last}(\mathsf{proj}_i(q))$.

– $i \neq j$ and $a \notin I_i$. Then by Definition 9.2.2 we have $\mathsf{proj}_{a,i}(f) = \mathsf{Dirac}(u_i)$. Since $\langle a, \vec{t} \rangle \in \mathsf{Supp}(\mu)$, it must be that $t_i = u_i$. Then $\mathsf{proj}_i(\mathsf{last}(q)) = t_i = u_i = \mathsf{last}(\mathsf{proj}_i(r)) = \mathsf{last}(\mathsf{proj}_i(q))$.

□

9.3 Probabilistic Systems

As we saw in Chapter 8, history-dependent adversaries with full observational power can produce problematic schedules in a parallel composition (Figure 8.2). In Chapter 11, we will argue that such schedules are results of inconsistent modeling and should be excluded from our semantics. To exclude these schedules in a systematic manner, we pair each PIOA with a set of acceptable schedules, forming a *probabilistic system* (Definition 9.3.2 below).

First, we make explicit the notion of schedules in an I/O setting. This is an extension of the notion of adversaries described in Chapter 3.

Definition 9.3.1. Let A be a PIOA. An *input scheduler* σ for A is a partial function

$$\sigma : \mathsf{Bran}(A) \times I_A \rightharpoonup \mathsf{Disc}(S_A)$$

such that: for all $\langle r, a \rangle \in \mathsf{Bran}(A) \times I$, if $\sigma(\langle r, a \rangle)$ is defined, then it is in $\mathbf{R}_A(\langle \mathsf{last}(r), a \rangle)$. An *output scheduler* ρ for A is a partial function

$$\rho : \mathsf{Bran}(A) \rightharpoonup \mathsf{Disc}((O_A \cup H_A) \times S_A)$$

such that: for all $r \in \mathsf{Bran}(A)$, if $\rho(r)$ is defined, then it is in $\mathbf{G}_A(\mathsf{last}(r))$. An *I/O scheduler* for A is then a pair $\langle \sigma, \rho \rangle$ where σ is an input scheduler for A and ρ is an output scheduler for A.

I/O schedulers remove certain types of non-deterministic choices in A. The input scheduler σ specifies the reactive schedule: given a finite history r and an input signal a that is not blocked in $\mathsf{last}(r)$, σ selects a distribution from $\mathbf{R}_A(\langle \mathsf{last}(r), a \rangle)$. Similarly, the output scheduler ρ specifies the generative schedule: given a finite history r, ρ selects a bundle from $\mathbf{G}_A(\mathsf{last}(r))$ if $\mathsf{last}(r)$ is not quiescent.

Notice that the output scheduler may stop all generative behavior by setting $\rho(r)$ to $\perp$, even if $\mathbf{G}_A(\mathsf{last}(r))$ is non-empty. Similarly, the input scheduler may refuse an input a even if $\mathbf{R}_A(\langle \mathsf{last}(r),\, a\rangle)$ is non-empty. Again, such freedom is included here so that our underlying framework is as flexible as possible. Depending on one's application, additional axioms may be imposed on the I/O schedulers. For example, in Chapter 10 we require I/O schedulers to enable all inputs while the automaton is in an inactive state (Definition 10.2.1); and in Chapter 11 we require I/O schedulers to be *determinant* (Definition 11.3.1), capturing the idea that actual implementations use deterministic input handling policies. One could go even farther by requiring that I/O schedulers are history-independent. Each such I/O scheduler corresponds to a deterministic sub-automaton (cf. Definition 9.1.5) of the original automaton. This approach is taken in [CCK$^+$06b].

Using the notion of I/O schedulers, it is straightforward to define probabilistic systems.

Definition 9.3.2. A *probabilistic system* $\mathcal{A}$ is a pair $\langle A,\, \mathcal{S}\rangle$, where A is a PIOA and $\mathcal{S}$ is a set of I/O schedulers for A. Such a system is *full* if $\mathcal{S}$ is the set of all I/O schedulers for A.

In the rest of this section, we define the *execution tree* induced by a triple $\langle A,\, \sigma,\, \rho\rangle$. This is analogous to a probabilistic execution in the PA framework (cf. Chapter 3).

Definition 9.3.3. Let A be a PIOA and let $\langle \sigma,\, \rho\rangle$ be an I/O scheduler for A. The *execution tree* generated by $\langle A,\, \sigma,\, \rho\rangle$ is the function $\mathbf{Q}_{\sigma,\rho} : \mathsf{Bran}(A) \to [0, 1]$ defined recursively by:

- $\mathbf{Q}_{\sigma,\rho}(\underline{s^0_A}) = 1$;
- given r' of the form $r.a.\mu.s'$,
  - $\mathbf{Q}_{\sigma,\rho}(r') := \mathbf{Q}_{\sigma,\rho}(r) \cdot \mu(s')$, if $\mu = \sigma(\langle r,\, a\rangle)$;
  - $\mathbf{Q}_{\sigma,\rho}(r') := 0$, otherwise;
- given r' of the form $r.f.a.s'$,
  - $\mathbf{Q}_{\sigma,\rho}(r') := \mathbf{Q}_{\sigma,\rho}(r) \cdot f(\langle a,\, s'\rangle)$, if $f = \rho(r)$;
  - $\mathbf{Q}_{\sigma,\rho}(r') := 0$, otherwise.

We say that a branch r is *reachable* under $\langle \sigma,\, \rho\rangle$ if $\mathbf{Q}_{\sigma,\rho}(r) \neq 0$.

Note that the input scheduler σ is the empty function whenever A is closed. In that case, we write $\mathbf{Q}_\rho$ for $\mathbf{Q}_{\sigma,\rho}$. We claim that $\mathbf{Q}_\rho$ induces a probability space over the sample space $\Omega_A := \mathsf{Bran}^{\leq\omega}(A)$. The construction is completely standard, so we provide an outline below and refer the reader to, for example, [Seg95b] for details.

(i) Each $r \in \mathsf{Bran}(A)$ generates a *cone* of executions as follows: $\mathbf{C}_r := \{r' \in \mathsf{Bran}^{\leq\omega}(A) \mid r \sqsubseteq r'\}$.

(ii) Let $\mathcal{F}_A$ denote the smallest σ-field on Ω_A generated by the collection $\{\mathbf{C}_r \mid r \in \mathsf{Bran}(A)\}$.

(iii) Construct a (unique) probability measure $\mathbf{m}_\rho$ on $\mathcal{F}_A$ such that $\mathbf{m}_\rho[\mathbf{C}_r] = \mathbf{Q}_\rho(r)$ for all r in $\mathsf{Bran}(A)$.

In this way, $\mathbf{Q}_\rho$ gives rise to the probability space $(\Omega_A, \mathcal{F}_A, \mathbf{m}_\rho)$.

For an open PIOA, an execution tree does not always induce a probability measure, because it does not take into account the probabilities with which various inputs are provided by the environment. For example, if r' is of the form $r.a.\mu.s'$, the value $\mathbf{Q}_{\sigma,\rho}(r')$ is computed from $\mathbf{Q}_{\sigma,\rho}(r)$ and $\mu(s')$, neither of which contains information about the probability of a being provided as an input.

Under the right conditions, one can prove that execution trees for an open PIOA induce sub-probability distributions that are *conditional* upon occurrences of inputs. For example, in the case of switched PIOAs of Chapter 10, every execution tree $\mathbf{Q}_{\sigma,\rho}$ restricts to a sub-probability distribution on $\mathsf{tr}^{-1}(\alpha)$ for any finite trace α (Proposition 10.2.1). A similar claim is proven in Chapter 11 for execution trees induced by determinate I/O schedulers (Proposition 11.3.1).

Overall, the notion of execution trees plays an important role in our technical development. It gives great flexibility in manipulating open components, which are typically part of a parallel composition forming a closed PIOA. In the end, we are assured that any probabilistic statement about the final, closed composite is meaningful; that is, it is based on a well-defined probability measure.

10

Distributed Scheduling

This chapter presents the framework of switched probabilistic input/output automata (or switched PIOAs), augmenting the original PIOA framework with an explicit control exchange mechanism. Using this mechanism, we model a network of processes passing a single token among them, so that the location of this token determines which process is scheduled to make the next move. This token structure therefore implements a distributed scheduling scheme: scheduling decisions are always made by the (unique) active component.

Distributed scheduling allows us to draw a clear line between local and global nondeterministic choices. We then require that local nondeterministic choices are resolved using strictly local information. This eliminates problematic schedules that arise under the more common centralized scheduling scheme (cf. Figures 8.1 and 8.2). As a result, we are able to prove that our trace-style semantics is compositional.

10.0.1 Token Structure

We propose a composition mechanism where local scheduling decisions are based on strictly local information, while global scheduling conflicts are eliminated using a control-passage mechanism. Note that the term *control* is used here in the spirit of "control flow" in sequential programming: a component is said to possess the control of a system if it is scheduled to actively perform the next action. This should not be confused with the notion of controllers for plants, as in control theory.

Intuitively, we model a network of processes passing a single token among them, with the property that a process enables a locally controlled transition (i.e., non-input) only if it possesses the token. Thus, the location of this unique token determines which process is scheduled to make the next move. We call this model *switched probabilistic input/output automata* (or *switched PIOAs* for short). It augments the *probabilistic input/output automata (PIOA)* model of Chapter 9 with additional structures and axioms for control exchange.

In particular, we add a predicate active on the set of states, indicating whether an automaton is active or inactive. We require that locally controlled actions are

enabled only if the automaton is active. In other words, an inactive automaton must be quiescent and can only accept inputs from the environment.

This activity status can be changed only by performing special control input and control output actions. Control inputs correspond to an incoming token, thus switching the automaton from inactive mode to active mode. And vice versa for control outputs. We make sure that all such control synchronizations are "handshakes": at most two components may participate in a transition labeled by a control action. Together with an appropriate initialization condition, this ensures that at most one component is active at any point of an execution.

In this framework, scheduling decisions are always made locally: each process is equipped with a local scheduler, which has access to local history and is responsible for resolving local non-deterministic choices. Among other things, the local scheduler chooses when to give up the activity token and to whom the token is sent. This is precisely the sense in which our scheduling scheme is *distributed*: global scheduling is performed collectively by all local schedulers. This scheme eliminates the need for adversaries such as the one in Figure 8.2 and allows us to give a compositional trace-style semantics (Definition 10.2.4 and Theorem 10.4.1).

10.0.2 Related Work

Distributed scheduling (as opposed to centralized scheduling) has been a mainstream approach in the area of security analysis [BPW04b, Can01], where information flow is a sensitive issue. Compared with the *interactive Turing machines* of [Can01] and *asynchronous reactive systems* of [BPW04b], our framework provides much better modeling flexibility, as we allow local nondeterministic choices to accommodate both lack of information and implementation freedom. However, we must admit this is an unfair comparison, because the two frameworks mentioned above are highly specialized for computational reasonings in cryptography.

10.0.3 From Distributed to Centralized

For those who may be skeptical of distributed scheduling, we argue that centralized scheduling can be implemented in our framework by modeling adversaries explicitly via an *arbiter* automaton. In other words, processes do not exchange control among each others directly, but they do so via the arbiter. This arbiter observes the whole system by means of action synchronization and it makes scheduling decisions according to its observations. Since the input signature of such an arbiter is completely flexible, this gives us a convenient way to specify what information is available for inter-component scheduling. This approach will be further discussed in Section 10.5, where we define *controllable PIOAs*.

10.1 Switched PIOAs

We now augment the PIOA model of Chapter 9 with additional structures and axioms, yielding the notion of switched PIOAs. These changes implement the token structure described in Section 10.0.1, which eliminates global scheduling conflicts by ensuring that:

(i) at any point of an execution, at most one component is active;

(ii) the currently active component always selects the next active component.

We begin with a distinction between *active* and *inactive* states of an automaton. Then we designate special *control actions* and impose five *switch axioms*, formalizing our intuitions about control passage among components. This yields Definition 10.1.1 below. For technical simplicity, we assume that Act is partitioned into two sets: BAct (*basic actions*) and CAct (*control actions*). Both sets are assumed to be countably infinite.

Definition 10.1.1. A *switched PIOA* is given by a PIOA A, together with a function $\mathsf{active}_A : S_A \to \{0,1\}$ and a set $\mathsf{Sync}_A \subseteq O_A \cap \mathsf{CAct}$ of *synchronized control actions* such that the following (universally quantified) axioms are satisfied.

(S1) $\mathsf{active}_A(s) = 0 \;\Rightarrow .\; \mathbf{G}_A(s) = \emptyset \wedge \forall a \in I_A.\; \mathbf{R}_A(\langle a, s\rangle) \neq \emptyset$

(S2) $\mathsf{active}_A(s) = 1 \;\Rightarrow .\; \forall a \in I_A.\; \mathbf{R}_A(\langle a, s\rangle) = \emptyset$

(S3) $(s \xrightarrow{a} s' \;\wedge\; a \in I_A \cap \mathsf{CAct}) \;\Rightarrow\; \mathsf{active}_A(s') = 1$

(S4) $(s \xrightarrow{a} s' \;\wedge\; a \in (O_A \cap \mathsf{CAct}) \setminus \mathsf{Sync}_A) \;\Rightarrow\; \mathsf{active}_A(s') = 0$

(S5) $(s \xrightarrow{a} s' \;\wedge\; a \in \mathsf{BAct} \cup H_A \cup \mathsf{Sync}_A) \;\Rightarrow\; \mathsf{active}_A(s) = \mathsf{active}_A(s')$

To increase readability, we classify the action symbols of A as follows:

- $\mathsf{BI}_A := I_A \cap \mathsf{BAct}$ (*basic inputs*);
- $\mathsf{BO}_A := O_A \cap \mathsf{BAct}$ (*basic outputs*);
- $\mathsf{CI}_A := I_A \cap \mathsf{CAct}$ (*control inputs*);
- $\mathsf{CO}_A := (O_A \cap \mathsf{CAct}) \setminus \mathsf{Sync}_A$ (*control outputs*).

Essentially, we have a partition $\{\mathsf{BI}_A, \mathsf{BO}_A, H_A, \mathsf{CI}_A, \mathsf{CO}_A, \mathsf{Sync}_A\}$ of Act_A. We say that A is *initially active* if $\mathsf{active}_A(s_A^0) = 1$. Otherwise, it is *initially inactive*.

The first two axioms constrain the behavior of A based on its activity status. Essentially, Axiom (S1) says that an inactive automaton is a reactive machine, therefore all inactive states of A must be quiescent and satisfy the usual input

enabling assumption. On the other hand, an active automaton is a generative machine, therefore Axiom (S2) requires all active states of A to be input blocking.

The last three axioms specify how actions of various types change the activity status of an automaton. Axioms (S3) and (S4) say that control inputs lead to active states and control outputs to inactive states. Axiom (S5) says that no other actions may change the activity status.

Together, these five axioms describe an "activity cycle" for the automaton A:

(i) while in inactive mode, A does not enable locally controlled transitions, although it may still receive inputs from its environment;

(ii) when A receives a control input it moves into active mode, where it may perform hidden or output transitions, possibly followed by a control output;

(iii) via this control output A returns to inactive mode.

This is captured in Lemma 10.1.1 below.

Lemma 10.1.1. *Let A be a switched PIOA and let s, s' in S_A and $a \in \mathsf{Act}_A$ be given. Suppose that $s \xrightarrow{a} s'$.*

1. *If $a \in \mathsf{BI}_A$, then $\mathsf{active}_A(s) = \mathsf{active}_A(s') = 0$.*
2. *If $a \in \mathsf{CI}_A$, then $\mathsf{active}_A(s) = 0$ and $\mathsf{active}_A(s') = 1$.*
3. *If $a \in \mathsf{BO}_A \cup H_A \cup \mathsf{Sync}_A$, then $\mathsf{active}_A(s) = \mathsf{active}_A(s') = 1$.*
4. *If $a \in \mathsf{CO}_A$, then $\mathsf{active}_A(s) = 1$ and $\mathsf{active}_A(s') = 0$.*

Proof. For Item (1), note that $a \in I_A$. By the definition of $s \xrightarrow{a} s'$, we may choose distribution $\mu \in \mathbf{R}_A(\langle s, a\rangle)$ such that $s' \in \mathsf{Supp}(\mu)$. Therefore, by Axiom (S2), we know that that $\mathsf{active}_A(s) = 0$. Applying Axiom (S5) we have $\mathsf{active}_A(s) = \mathsf{active}_A(s') = 0$. Item (3) follows similarly from Axioms (S1) and (S5).

For Item (2), we first use Axiom (S2) to argue that $\mathsf{active}_A(s) = 0$. Moreover, Axiom (S3) implies $\mathsf{active}_A(s') = 1$. Item (4) follows similarly from Axioms (S1), (S4). □

To give some concrete examples of switched PIOAs, we return to automata Early, Late and Coin of Figure 8.1. Their adaptations to the switched PIOA framework are illustrated in Figure 10.1 below. We have chosen to assign actions b and c to the basic output signature of Early′ and Late′, whereas a, d and e are basic outputs of Coin′. Following conventions in process algebra, we use ? to indicate input actions and ! to indicate output actions.

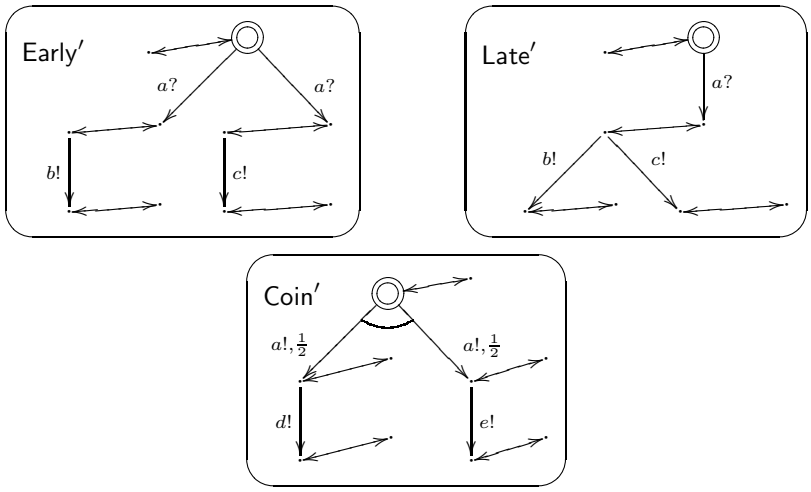

Figure 10.1: Adaptations of Early, Late and Coin

Due to the additional predicate active, the state spaces have been doubled. Active states are drawn in the foreground and inactive ones in the background. Thus, Early′ and Late′ are initially inactive and Coin′ is initially active.

Each two-headed arrow indicates a control output from active to inactive and a control input from inactive to active. We assume that Early′ and Late′ have a sole control input go and a sole control output done; and vice versa for Coin′. For a clearer picture, we have omitted the names of control actions, as well as non-essential input loops.

10.2 External Behavior

Observe that switched PIOAs are defined as a special class of PIOAs. Therefore all of the technical notions developed in Chapter 9 can be inherited. The present section uses I/O schedulers to define a semantics of external behavior, while Section 10.3 deals with parallel composition.

First we impose an input enabling axiom on input schedulers for a switched PIOA. This reflects our intention that an inactive automaton is reactive, thus willing to accept all inputs in its signature. Switched probabilistic systems are then defined accordingly.

Definition 10.2.1. Let A be a switched PIOA. An *I/O* scheduler for A is a pair $\langle \sigma, \rho \rangle$ satisfying:

- $\langle \sigma, \rho \rangle$ is an I/O scheduler for A in the sense of Definition 9.3.1;
- for all $\langle r, a \rangle \in \mathsf{Bran}(A) \times I_A$, if $\mathsf{active}_A(\mathsf{last}(r)) = 0$, then $\sigma(\langle r, a \rangle) \neq \bot$.

A *switched probabilistic system* $\mathcal{A}$ is a pair $\langle A, \mathcal{S}\rangle$ where A is a switched PIOA and $\mathcal{S}$ is a set of I/O schedulers for A as defined above. Such a system is *full* if $\mathcal{S}$ is the set of all I/O scheduler for A.

10.2.1 Conditional Probability Distributions

Recall from Section 9.3 that each I/O scheduler $\langle \sigma, \rho\rangle$ for A induces an execution tree $\mathbf{Q}_{\sigma,\rho} : \mathsf{Bran}(A) \to [0,1]$. We also commented that an execution tree in an open PIOA does not always induce a probability measure, because it does not take into account input probabilities.

We now show that, in the case of switched PIOAs, one can in fact make meaningful probabilistic statements based on execution trees, as long as these statements are conditioned upon occurrences of inputs. This claim is formalized in Proposition 10.2.1 below.

Proposition 10.2.1. *Let A be a switched PIOA and let $\langle \sigma, \rho\rangle$ be an I/O scheduler for A. Let $\alpha \in (I_A \cup O_A)^{<\omega}$ be given and assume that $\mathsf{tr}^{-1}(\alpha)$ in A is nonempty. Then the restriction of $\mathbf{Q}_{\sigma,\rho}$ to $\mathsf{tr}^{-1}_{\min}(\alpha)$ is a discrete sub-probability distribution.*

The proof of Proposition 10.2.1 relies on some auxiliary lemmas. First we show that every non-minimal branch ends in an active state. This is essentially a corollary of Lemma 10.1.1.

Lemma 10.2.2. *Let A be any switched PIOA and let s be a state in A. For every non-minimal branch r in $\mathsf{Bran}(s)$, $\mathsf{active}_A(\mathsf{last}(r)) = 1$.*

Proof. Since r is non-minimal, it must be non-empty and of the form $q.f.a.t$ where $a \in H_A$. Then we have $\mathsf{last}(q) \xrightarrow{a} t$. By Lemma 10.1.1, we know that $\mathsf{active}_A(\mathsf{last}(r)) = \mathsf{active}_A(t) = 1$. □

Extending Lemma 10.2.2, we show that inputs transitions are never preceded by hidden transitions.

Lemma 10.2.3. *Let A be any switched PIOA and let s be a state in A. Let $r, r' \in \mathsf{Bran}(s)$ be given and suppose r' is of the form $r.a.\mu.s'$. Then r is minimal.*

Proof. By the structure of r' we know that $a \in I_A$. Lemma 10.1.1 guarantees that $\mathsf{active}_A(\mathsf{last}(r)) = 0$. Thus, by Lemma 10.2.2, r must be minimal. □

Next we consider the case in which an output action a takes place after a trace α. Notice, this lemma applies to PIOAs in general.

Lemma 10.2.4. *Let A be a PIOA and let $\langle \sigma, \rho\rangle$ be an I/O scheduler for A. Let $\alpha \in (I_A \cup O_A)^{<\omega}$ and $a \in O_A$ be given. Suppose that $\mathsf{tr}^{-1}(\alpha a)$ in A is nonempty. The following holds for every $r \in \mathsf{tr}^{-1}_{\min}(\alpha)$.*

(i) Let $\mathbf{C}$ *denote the set of branches* $r' \in \mathsf{tr}^{-1}(\alpha)$ *such that* $r \sqsubseteq r' \sqsubseteq r''$ *for some* $r'' \in \mathsf{tr}^{-1}_{\min}(\alpha a)$*. For each* $k \in \mathbb{N}$*, let* $\mathbf{C}_k$ *denote the set of* $r' \in \mathbf{C}$ *such that* r' *extends* r *with* k *transitions. Then* $\sum_{r' \in \mathbf{C}_k} \mathbf{Q}_{\sigma,\rho}(r') \leq \mathbf{Q}_{\sigma,\rho}(r)$*.*

(ii) $\sum_{r'' \in \mathsf{tr}^{-1}_{\min}(\alpha a),\ r \sqsubseteq r''} \mathbf{Q}_{\sigma,\rho}(r'') \leq \mathbf{Q}_{\sigma,\rho}(r)$*.*

Proof. Observe that all of the infinite sums above are countable, since the state space and transition structures of A are countable.

We prove Item (i) by induction on k. The base case is trivial since r is the unique element in $\mathbf{C}_0$. Consider $r'' \in \mathbf{C}_{k+1}$. Since the last transition in r'' is a hidden transition, r'' must be of the form $r'.f.b.s''$ where $r' \in \mathbf{C}_k$ and $b \in H_A$. By the definition of $\mathbf{Q}_{\sigma,\rho}$, we know that $\mathbf{Q}_{\sigma,\rho}(r'') \neq 0$ implies $f = \rho(r')$ and $\langle b,\, s'' \rangle \in \mathsf{Supp}(f)$. Therefore we have the following.

$$
\begin{aligned}
&\sum_{r'' \in \mathbf{C}_{k+1}} \mathbf{Q}_{\sigma,\rho}(r'') \\
&= \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \bot\}} \ \sum_{\{\langle b,\, t \rangle \in \mathsf{Supp}(\rho(r'))\ \mid\ r'.\rho(r').b.t \in \mathbf{C}_{k+1}\}} \mathbf{Q}_{\sigma,\rho}(r') \cdot \rho(r')(\langle b,\, t \rangle) \\
&= \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \bot\}} \Big(\mathbf{Q}_{\sigma,\rho}(r') \cdot \sum_{\{\langle b,\, t \rangle \in \mathsf{Supp}(\rho(r'))\ \mid\ r'.\rho(r').b.t \in \mathbf{C}_{k+1}\}} \rho(r')(\langle b,\, t \rangle) \Big) \\
&\leq \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \bot\}} \mathbf{Q}_{\sigma,\rho}(r') \cdot 1 \\
&\leq \sum_{r' \in \mathbf{C}_k} \mathbf{Q}_{\sigma,\rho}(r')
\end{aligned}
$$

By the induction hypothesis, this is at most $\mathbf{Q}_{\sigma,\rho}(r)$.

We move on to Item (ii). By the definition of minimality, every $r'' \in \mathsf{tr}^{-1}_{\min}(\alpha a)$ is of the from $r'.f.a.s''$ with $r' \in \mathbf{C}_k$ for some k. Again, by the definition of $\mathbf{Q}_{\sigma,\rho}$, we have $\mathbf{Q}_{\sigma,\rho}(r'') \neq 0$ implies $f = \rho(r)$ and $\langle a,\, s'' \rangle \in \mathsf{Supp}(f)$. This implies:

$$
\begin{aligned}
&\sum_{r'' \in \mathsf{tr}^{-1}_{\min}(\alpha a),\ r \sqsubseteq r''} \mathbf{Q}_{\sigma,\rho}(r'') \\
&= \sum_{k=0}^{\infty} \ \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \bot\}} \ \sum_{\{s'' \ \mid\ \langle a,\, s'' \rangle \in \mathsf{Supp}(\rho(r'))\}} \mathbf{Q}_{\sigma,\rho}(r') \cdot \rho(r')(\langle a,\, s'' \rangle).
\end{aligned}
$$

Therefore, it suffices to show that all partial sums are less than or equal to $\mathbf{Q}_{\sigma,\rho}(r)$. To save space, let $L_{r'}$ denote

$$
\sum_{\{s'' \ \mid\ \langle a,\, s'' \rangle \in \mathsf{Supp}(\rho(r'))\}} \rho(r')(\langle a,\, s'' \rangle),
$$

and let $M_{r'}$ denote

$$
\sum_{\{\langle b,\, t \rangle \in \mathsf{Supp}(\rho(r'))\ \mid\ r'.\rho(r').b.t \in \mathbf{C}_{k+1}\}} \rho(r')(\langle b,\, t \rangle)).
$$

For each $k \in \mathbb{N}$, we have

$$\begin{aligned}
&\sum_{\{r' \in \mathbf{C}_k \mid \rho(r') \neq \bot\}} \; \sum_{\{s'' \mid \langle a, s'' \rangle \in \mathsf{Supp}(\rho(r'))\}} \mathbf{Q}_{\sigma,\rho}(r') \cdot \rho(r')(\langle a,\, s'' \rangle) \\
&= \sum_{\{r' \in \mathbf{C}_k \mid \rho(r') \neq \bot\}} \mathbf{Q}_{\sigma,\rho}(r') \cdot L_{r'} \\
&\leq \sum_{\{r' \in \mathbf{C}_k \mid \rho(r') \neq \bot\}} \mathbf{Q}_{\sigma,\rho}(r') \cdot (1 - M_{r'}) \\
&= \sum_{\{r' \in \mathbf{C}_k \mid \rho(r') \neq \bot\}} \mathbf{Q}_{\sigma,\rho}(r') - \sum_{\{r' \in \mathbf{C}_k \mid \rho(r') \neq \bot\}} \mathbf{Q}_{\sigma,\rho}(r') \cdot M_{r'} \\
&\leq \sum_{r' \in \mathbf{C}_k} \mathbf{Q}_{\sigma,\rho}(r') \; - \sum_{r' \in \mathbf{C}_{k+1}} \mathbf{Q}_{\sigma,\rho}(r'),
\end{aligned}$$

where the last inequality follows from the proof of Item (i). Now, for all $K \in \mathbb{N}$,

$$\begin{aligned}
&\sum_{k=0}^{K} \; \sum_{\{r' \in \mathbf{C}_k \mid \rho(r') \neq \bot\}} \; \sum_{\{s'' \mid \langle a, s'' \rangle \in \mathsf{Supp}(\rho(r'))\}} \mathbf{Q}_{\sigma,\rho}(r') \cdot \rho(r')(\langle a,\, s'' \rangle) \\
&\leq \sum_{k=0}^{K} \; (\sum_{r' \in \mathbf{C}_k} \mathbf{Q}_{\sigma,\rho}(r') \; - \sum_{r' \in \mathbf{C}_{k+1}} \mathbf{Q}_{\sigma,\rho}(r')) \\
&= \sum_{r' \in \mathbf{C}_0} \mathbf{Q}_{\sigma,\rho}(r') \; - \sum_{r' \in \mathbf{C}_{K+1}} \mathbf{Q}_{\sigma,\rho}(r')) \\
&\leq \sum_{r' \in \mathbf{C}_0} \mathbf{Q}_{\sigma,\rho}(r') = \mathbf{Q}_{\sigma,\rho}(r).
\end{aligned}$$

□

Proof of Proposition 10.2.1. We proceed by induction on the length of α. If α is empty, then $\mathsf{tr}^{-1}_{\min}(\alpha)$ contains a unique element, namely, $\underline{s^0_A}$. Our claim holds because by definition $\mathbf{Q}_{\sigma,\rho}(\underline{s^0_A}) = 1$.

Consider α' of the form αa. We have two cases.

- $a \in I_A$. Let $r' \in \mathsf{tr}^{-1}_{\min}(\alpha')$ be given. By the definition of minimality, r' must be of the form $r.a.\mu.s'$. By Lemma 10.2.3, r is minimal and hence in $\mathsf{tr}^{-1}_{\min}(\alpha)$. Moreover, by the definition of $\mathbf{Q}_{\sigma,\rho}$, we know that $\mathbf{Q}_{\sigma,\rho}(r') \neq 0$

implies $\mu = \sigma(\langle r, a\rangle)$. Therefore,

$$\begin{aligned}
&\sum_{r' \in \mathsf{tr}^{-1}_{\min}(\alpha')} \mathbf{Q}_{\sigma,\rho}(r') \\
&= \sum_{\{r \in \mathsf{tr}^{-1}_{\min}(\alpha) \mid \sigma(\langle r,a\rangle) \neq \bot\}} \; \sum_{s' \in \mathsf{Supp}(\sigma(\langle r,a\rangle))} \mathbf{Q}_{\sigma,\rho}(r) \cdot \sigma(\langle r, a\rangle)(s') \\
&= \sum_{\{r \in \mathsf{tr}^{-1}_{\min}(\alpha) \mid \sigma(\langle r,a\rangle) \neq \bot\}} (\mathbf{Q}_{\sigma,\rho}(r) \cdot \sum_{s' \in \mathsf{Supp}(\sigma(\langle r,a\rangle))} \sigma(\langle r, a\rangle)(s')) \\
&\leq \sum_{\{r \in \mathsf{tr}^{-1}_{\min}(\alpha) \mid \sigma(\langle r,a\rangle) \neq \bot\}} \mathbf{Q}_{\sigma,\rho}(r) \\
&\leq \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} \mathbf{Q}_{\sigma,\rho}(r) \; \leq 1,
\end{aligned}$$

where the last inequality follows from the induction hypothesis.

– $a \in O_A$. By Lemma 10.2.4, we have

$$\begin{aligned}
\sum_{r' \in \mathsf{tr}^{-1}_{\min}(\alpha')} \mathbf{Q}_{\sigma,\rho}(r') &= \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} \; \sum_{r'' \in \mathsf{tr}^{-1}_{\min}(\alpha a),\ r \sqsubseteq r''} \mathbf{Q}_{\sigma,\rho}(r'') \\
&\leq \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} \mathbf{Q}_{\sigma,\rho}(r) \leq 1.
\end{aligned}$$

Again, the last inequality follows from the induction hypothesis.

□

10.2.2 Likelihood Assignments

we define a trace-style notion of external behavior for switched probabilistic systems. In particular, we derive a *likelihood assignment* from each triple $\langle A, \sigma, \rho\rangle$, where A is a switched PIOA and $\langle \sigma, \rho\rangle$ is an I/O scheduler for A in the sense of Definition 10.2.1. This is analogous to the notion of *trace distributions* for probabilistic automata (cf. Chapter 3).

Likelihood assignments are behavioral abstractions of execution trees. Roughly speaking, the probability of observing a certain trace $\alpha \in (I_A \cup O_A)^{<\omega}$ is the probability of the automaton executing *any* branch with trace α. This can be computed by summing the probabilities of all such branches in the execution tree. As we mentioned at the end of Section 9.3, execution trees of open PIOAs need not always induce probability measures. That is the reason we opt for the term "likelihood", rather than "probability". Nonetheless, the method of abstraction is completely analogous.

Likelihood assignments are defined via a lifting of the trace operator

$$\mathsf{tr} : \mathsf{Bran}(A) \to (I_A \cup O_A)^{<\omega}.$$

Definition 10.2.2. Let A be a switched PIOA and let $\langle \sigma, \rho \rangle$ be an I/O scheduler for A in the sense of Definition 10.2.1. The *likelihood assignment* induced by $\langle A, \sigma, \rho \rangle$, denoted $\mathbf{L}_{\sigma,\rho}$, is the function $\mathsf{tr}(\mathbf{Q}_{\sigma,\rho}) : (I_A \cup O_A)^{<\omega} \to [0,1]$ given as follows.

$$\mathsf{tr}(\mathbf{Q}_{\sigma,\rho})(\alpha) := \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} \mathbf{Q}_{\sigma,\rho}(r).$$

This is well-defined by virtue of Proposition 10.2.1.

As with execution trees, we omit the input scheduler σ whenever A is closed. In that case, each $\mathbf{L}_\rho$ induces a probability measure on the sample space $\Omega := (I_A \cup O_A)^{\leq\omega}$. The σ-field $\mathcal{F}$ on Ω is generated by the collection $\{\mathbf{C}_\alpha \mid \alpha \in (I_A \cup O_A)^{<\omega}\}$, where $\mathbf{C}_\alpha := \{\alpha' \in \Omega \mid \alpha \sqsubseteq \alpha'\}$. The measure $\mathbf{m}^\rho$ on $\mathcal{F}$ is uniquely determined by the equations $\mathbf{m}^\rho[\mathbf{C}_\alpha] = \mathbf{L}_\rho(\alpha)$ for all $\alpha \in (I_A \cup O_A)^{<\omega}$.

Thus, our notion of likelihood assignments generalizes the notion of trace distributions proposed by Segala (cf. Chapter 3 of the present thesis and [Seg95b]). This is analogous to the relationship between execution trees and probabilistic executions.

Since probabilistic executions and trace distributions are not well-defined in the presence of inputs, we have traditionally relied on *closing contexts* in order to define the behavior of open automata [CLSV04a, CLSV04b]. Under that approach, a possible behavior of an open automaton A is a trace distribution of $A \| C$, where C is any closing context for A (i.e., C is compatible with A and every input action of A is an output of C). This cumbersome step often complicates our proofs of behavioral inclusion, obscuring ideas that are more fundamental. For example, a hiding operator is used to remove extra output actions in a closing context C (i.e., those that are not inputs to A), and we needed to prove that such hiding is well-behaved with respect to parallel composition.

In contrast, there is no need to quantify over closing contexts under the current setup, because execution trees and likelihood assignments are well-defined for open automata. The quantification is implicit in our definitions, since an execution tree can be seen as a collection of conditional sub-probability distributions (cf. Proposition 10.2.1). This leads to a very simple and natural notion of external behavior.

Definition 10.2.3. Let $\mathcal{A} = \langle A, \mathcal{S} \rangle$ be a switched probabilistic system. An *external behavior* of $\mathcal{A}$ is a likelihood assignment $\mathbf{L}_{\sigma,\rho}$ induced by some $\langle \sigma, \rho \rangle \in \mathcal{S}$. We write $\mathsf{ExtBeh}(\mathcal{A})$ for the set of all external behaviors of $\mathcal{A}$.

As usual, implementation is given by behavioral inclusion.

Definition 10.2.4. Switched probabilistic systems $\mathcal{A} = \langle A, \mathcal{S} \rangle$ and $\mathcal{B} = \langle B, \mathcal{T} \rangle$ are said to be *comparable* if:

- $\mathsf{active}_A(s^0_A) = \mathsf{active}_B(s^0_B)$ and
- $I_A = I_B$, $O_A = O_B$, and $\mathsf{Sync}_A = \mathsf{Sync}_B$.

Given such comparable $\mathcal{A}$ and $\mathcal{B}$, we say that $\mathcal{A}$ *implements* $\mathcal{B}$ if $\mathsf{ExtBeh}(\mathcal{A}) \subseteq \mathsf{ExtBeh}(\mathcal{B})$.

This concludes our treatment of external behavior for switched PIOAs. In the next section, we consider parallel composition of switched PIOAs.

10.3 Parallel Composition

Recall from Section 9.2 the definition of parallel composition for PIOAs. It is based on action synchronization and does not attempt to resolve nondeterministic choices among parallel components. In the present section, we extend that definition to switched PIOAs, taking care that the parallel composite still satisfies all switch axioms.

Then, departing from the "compose-and-schedule" approach (cf. Section 8.3), we describe how to compose I/O schedulers for compatible switched PIOAs to form a single I/O scheduler for their composite. This in turn yields parallel composition for switched probabilistic systems. Thus, our approach can be described as "schedule-and-compose", where parallel composition is imposed *after* local schedules have been completely specified.

10.3.1 Composing Switched PIOAs

As usual, we need an appropriate notion of compatibility: switched PIOAs A and B are said to be *compatible* if

- they are compatible as PIOAs;
- $\mathsf{Act}_A \cap \mathsf{Sync}_B = \mathsf{Act}_B \cap \mathsf{Sync}_A = \mathsf{Cl}_A \cap \mathsf{Cl}_B = \emptyset$;
- at most one of them is initially active.

Since switched PIOAs are special cases of PIOAs, one may apply the operator $\uparrow\uparrow$ of Section 9.2 to compatible switched PIOAs. Unfortunately, the result does not always satisfy all switch axioms. We give a simple example.

Example 10.3.1. Consider automata D and E in Figure 10.2 below and assume that all actions shown are control actions.

If from the initial state the composite $D \uparrow\uparrow E$ receives an input signal a, then D moves into an active state, s_1, and E remains at its initial state. This is shown in Figure 10.3. In state $\langle s_1, s_E^0 \rangle$, the composite is considered active, because D is. However, an input transition with label b is still enabled, violating Axiom (S2). Moreover, suppose in fact an input signal b is received from state $\langle s_1, s_E^0 \rangle$. Then in the resulting state $\langle s_1, s_2 \rangle$ both D and E are active. This state violates Axiom (S4), because a single control output (say c) is not sufficient to deactivate both components (Figure 10.3).

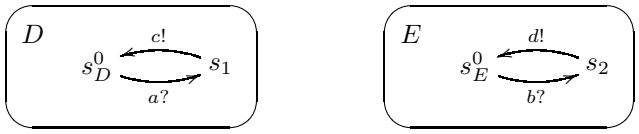

Figure 10.2: Automata D and E

$$\langle s_D^0,\, s_E^0\rangle \xrightarrow{a?} \langle s_1,\, s_E^0\rangle \xrightarrow{b?} \langle s_1,\, s_2\rangle \xrightarrow{c!} \langle s_D^0,\, s_2\rangle$$

Figure 10.3: A Potential Execution of $D \uparrow\uparrow E$

This is a counterintuitive scenario: if the environment of $D \uparrow\uparrow E$ is itself a switched PIOA, then it should have become inactive after providing the first control input a, thus unable to provide the second control input b. In fact, it is shown in [CLSV04b] that any state with more than one active components is unreachable, provided the closing environment is also a switched PIOA. (The proof involves lengthy inductive arguments and is omitted here.)

This example suggests that, when switched PIOAs are composed using the PIOA parallel operator $\uparrow\uparrow$, the resulting state space and reactive transition structure both contain too many elements. Therefore, we are prompted to consider an appropriate sub-automaton with fewer states and fewer input transitions, as in Definition 10.3.1 below.

Definition 10.3.1. Let $\{A_i \mid i \in I\}$ be a set of pairwise compatible switched PIOAs. The *parallel composite*, $\|_{i=1}^n A_i$, is based on the sub-automaton B of $\uparrow\uparrow_{i=1}^n A_i$ obtained by

(i) removing all states in which more than one A_i's are active;

(ii) removing all input transitions from states in which at least one S_i is active.

Moreover, $\mathsf{Sync}_B := \bigcup_{1\le i\le n} \mathsf{Sync}_i \cup \bigcup_{1\le i,j\le n} (\mathsf{CI}_i \cap \mathsf{CO}_j)$, and $\mathsf{active}_B(\vec{s}) := 0$ if and only if $\mathsf{active}_i(s_i) = 0$ for all i.

Although the signature of B is completely specified in Definition 10.3.1, it is instructive to provide a list of explicit identities.

Lemma 10.3.1. *The following equalities hold:*

- $\mathsf{BI}_B = \bigcup_{1\le i\le n} \mathsf{BI}_i \setminus \bigcup_{1\le i\le n} \mathsf{BO}_i$;
- $\mathsf{CI}_B = \bigcup_{1\le i\le n} \mathsf{CI}_i \setminus \bigcup_{1\le i\le n} \mathsf{CO}_i$;

- $\mathsf{BO}_B = \bigcup_{1\le i\le n} \mathsf{BO}_i$;
- $\mathsf{CO}_B = \bigcup_{1\le i\le n} \mathsf{CO}_i \setminus \bigcup_{1\le i\le n} \mathsf{CI}_i$.

Proof. By definition, $I_B = \bigcup_{1\le i\le n} I_i \setminus \bigcup_{1\le i\le n} O_i$. Since BAct and CAct are disjoint, we have the desired properties about BI_B and CI_B.

Similarly, $O_B = \bigcup_{1\le i\le n} O_i$, therefore $\mathsf{BO}_B = \bigcup_{1\le i\le n} \mathsf{BO}_i$ and $O_B \cap \mathsf{CAct} = \bigcup_{1\le i\le n} \mathsf{CO}_i$. Applying the definitions of CO_B and Sync_B, we have $\mathsf{CO}_B = \bigcup_{1\le i\le n} \mathsf{CO}_i \setminus \bigcup_{1\le i\le n} \mathsf{CI}_i$. □

To show that such B is a well-defined PIOA, we need to verify (i) $s_B^0 \in S_B$ and (ii) S_B is closed under the reduced transition structures. Clearly, the first claim holds by the definition of compatibility. The second is confirmed by Lemmas 10.3.2 and 10.3.3 below.

For convenience, we partition S_B into two sets:

- $S_{B,0}$ is the set of all $\vec{s}$ such that $\mathsf{active}_i(s_i) = 0$ for all i;
- $S_{B,1}$ is the set of all $\vec{s}$ such that $\mathsf{active}_i(s_i) = 1$ for exactly one i.

Lemma 10.3.2. *Let $\vec{s} \in S_B$ and $a \in I_B$ be given. For all $\mu \in \mathbf{R}_B(\langle \vec{s}, a\rangle)$:*

- *$a \in \mathsf{BI}_B$ implies $\mathsf{Supp}(\mu) \subseteq S_{B,0}$;*
- *$a \in \mathsf{CI}_B$ implies $\mathsf{Supp}(\mu) \subseteq S_{B,1}$.*

Proof. By definition, $\mathbf{R}_B(\langle \vec{s}, a\rangle)$ is empty whenever $\vec{s} \in S_{B,1}$. Therefore we may assume that $\vec{s} \in S_{B,0}$. Let $\mu \in \mathbf{R}_B(\langle \vec{s}, a\rangle)$ and $\vec{s}' \in \mathsf{Supp}(\mu)$ be given.

First assume $a \in \mathsf{BI}_B$. For every i, if $a \notin \mathsf{Act}_i$, it must be the case that $s_i = s_i'$ and hence $\mathsf{active}_i(s_i) = \mathsf{active}_i(s_i') = 0$. Otherwise, we have $a \in \mathsf{BI}_i$ and we may apply Lemma 10.1.1 to conclude that $\mathsf{active}_i(s_i') = 0$. Therefore $\vec{s}' \in S_{B,0}$.

Now assume $a \in \mathsf{CI}_B$. By compatibility, $a \in \mathsf{Act}_j$ for exactly one j. Choose such j. By Lemma 10.1.1, we know $\mathsf{active}_j(s_j') = 1$. For all other i, $a \notin \mathsf{Act}_i$ and hence $\mathsf{active}_i(s_i) = \mathsf{active}_i(s_i') = 0$. This proves $\vec{s}' \in S_{B,1}$. □

Lemma 10.3.3. *Let $\vec{s} \in S_B$ and $f \in \mathbf{G}_B(\vec{s})$ be given. For every $\langle a, \vec{s}'\rangle \in \mathsf{Supp}(f)$;*

- *If $a \in \mathsf{BO}_B \cup \mathsf{Sync}_B \cup H_B$, then $\vec{s}' \in S_{B,1}$;*
- *If $a \in \mathsf{CO}_B$, then $\vec{s}' \in S_{B,0}$.*

Proof. By Axiom (1), we know that $\mathbf{G}_i(s_i)$ is empty for every i with $\mathsf{active}_i(s_i) = 0$. This implies $s \in S_{B,1}$, because otherwise $\mathbf{G}_B(\vec{s})$ would be empty. Let j be the unique index with $\mathsf{active}_j(s_j) = 1$ and choose $g_j \in \mathbf{G}_j(s_j)$ such that f is generated by g_j. By Definition 9.2.2, a must be in $O_j \cup H_j$. We have the following cases.

1. $a \in H_j \cup \mathsf{Sync}_j$. Compatibility of switched PIOAs requires that $a \notin \mathsf{Act}_i$ for all $i \neq j$. This implies, for all $i \neq j$, $s_i = s'_i$ and hence $\mathsf{active}_i(s'_i) = \mathsf{active}_i(s_i) = 0$. On the other hand, we may apply Lemma 10.1.1 to A_j and conclude that $\mathsf{active}_i(s'_j) = \mathsf{active}_i(s_j) = 1$. Therefore, $\vec{s}' \in S_{B,1}$.

2. $a \in \mathsf{BO}_j$. For every i such that $a \notin \mathsf{Act}_i$, we know that $s_i = s'_i$ and hence $\mathsf{active}_i(s_i) = \mathsf{active}_i(s'_i) = 0$. For every i such that $i \neq j$ and $a \in \mathsf{Act}_i$, it must be the case that $a \in \mathsf{BI}_i$, so we apply Lemma 10.1.1 to conclude that $\mathsf{active}_i(s'_i) = 0$. As in the previous case, we know $\mathsf{active}_i(s'_j) = 1$. Therefore, $\vec{s}' \in S_{B,1}$.

3. $a \in \mathsf{CO}_j \cap \mathsf{CI}_k$ for some $k \neq j$. By Lemma 10.1.1, we have $\mathsf{active}_j(s'_j) = 0$ and $\mathsf{active}_k(s'_k) = 1$. By the compatibility of switched PIOAs, there is at most one such k. For all other indices i, $\mathsf{active}_i(s_i) = \mathsf{active}_i(s'_i) = 0$. Again we conclude $\vec{s}' \in S_{B,1}$.

4. $a \in \mathsf{CO}_B$. By the definition of CO_B, we know that $a \notin \mathsf{Act}_i$ for all $i \neq j$. Hence $\mathsf{active}_i(s_i) = \mathsf{active}_i(s'_i) = 0$ for all $i \neq j$. By Lemma 10.1.1, we have $\mathsf{active}_j(s'_j) = 0$. Thus, $\vec{s}' \in S_{B,0}$.

□

It remains to show that B satisfies all switch axioms.

Lemma 10.3.4. *The PIOA B, together with active_B and Sync_B, satisfies Axioms (S1) through (S5) in Definition 10.1.1.*

Proof. Note that $\mathsf{active}_B(\vec{s}) = 0$ if and only if $\vec{s} = S_{B,0}$. For Axiom (S1), let $\vec{s} \in S_{B,0}$ and $a \in I_B$ be given. Applying Axiom (S1) on each component, we know that $\mathbf{G}_i(s_i)$ is empty for every i with $\mathsf{active}_i(s_i) = 0$ and hence $\mathbf{G}_B(\vec{s}) = \emptyset$. On the other hand, for all i with $a \in I_i$, Axiom (S2) requires $\mathbf{R}_i(\langle s_i, a\rangle)$ is non-empty. Hence $\mathbf{R}_B(\langle \vec{s}, a\rangle)$ is non-empty. This proves that B satisfies Axiom (S1).

Axiom (S2) follows from the definition of $\mathbf{R}_B$. Axioms (S3) through (S5) follow from Lemmas 10.3.2 and 10.3.3. □

We adopt the same notational conventions as with $\uparrow\uparrow$. Namely, $\|^n$ denotes the n-ary operator and $\|$ denotes the (infix) binary operator. Again commutativity is trivial. For associativity, it is easy to see that $(A \parallel B) \parallel C$ has the same state space as $\|^3 \{A, B, C\}$. Similarly for $A \parallel (B \parallel C)$. The transition structures are isomorphic because they are based on parallel composition of PIOAs, which is associative.

10.3.2 Composing I/O Schedulers

The goal of this section is to extend the parallel operator $\|$ to switched probabilistic systems, therefore we consider composition of I/O schedulers. For that end, we use the various notions of projection defined in Chapter 9 (Section 9.2).

Definition 10.3.2. Let $\{A_i \mid 1 \leq i \leq n\}$ be a set of pairwise compatible switched PIOAs and let B denote $\|_{i=1}^{n} A_i$. Suppose we have, for each i, an I/O scheduler $\langle \sigma_i, \rho_i \rangle$ for A_i. These I/O schedulers are said to generate the following I/O scheduler $\langle \sigma, \rho \rangle$ for B. Let $r \in \mathsf{Bran}(B)$ be given and let $\vec{s}$ denote $\mathsf{last}(r)$.

- If $\mathsf{active}_B(\vec{s}) = 1$, then $\sigma(\langle r, a \rangle) := \perp$ for all $a \in I_B$.
- If $\mathsf{active}_B(\vec{s}) = 0$, then for all $a \in I_B$, $\sigma(\langle r, a \rangle) := \prod_{i=1}^{n} \mu_i$, where μ_i equals $\mathsf{Dirac}(s_i)$ whenever $a \notin I_i$ and $\sigma_i(\langle \mathsf{proj}_i(r), a \rangle)$ otherwise.
- If $\mathsf{active}_B(\vec{s}) = 0$, then $\rho(r) := \perp$.
- If $\mathsf{active}_B(\vec{s}) = 1$, then $\rho(r) \neq \perp$ if and only $\rho_j(\mathsf{proj}_j(r)) \neq \perp$, where j is the unique index with $\mathsf{active}_j(s_j) = 1$. In that case, $\rho(r)$ is the bundle $f = \rho_j(\mathsf{proj}_j(r)) \times \prod_{\langle a, i \rangle \in N_j} \mu_{a,i}$, where $\mu_{a,i}$ equals $\mathsf{Dirac}(s_i)$ whenever $a \notin I_i$ and $\sigma_i(\langle \mathsf{proj}_i(r), a \rangle)$ otherwise.

Lemma 10.3.5. *The I/O scheduler $\langle \sigma, \rho \rangle$ in Definition 10.3.2 is well-defined.*

Proof. Let $r \in \mathsf{Bran}(B)$ and $a \in I_B$ be given. Let $\vec{s}$ denote $\mathsf{last}(r)$. First we consider the case where $\mathbf{R}_B(\langle \mathsf{last}(r), a \rangle)$ is non-empty. Since B satisfies Axiom (S2), it must be the case that $\mathsf{active}_B(\vec{s}) = 0$ and hence $\mathsf{active}_i(s_i) = 0$ for all i.

By Axiom (S1), $\mathbf{R}_i(\langle s_i, a \rangle)$ is non-empty for all $a \in I_i$. By the definition of input schedulers, this implies $\sigma_i(\langle \mathsf{proj}_i(r), a \rangle)$ is defined and is in $\mathbf{R}_i(\langle s_i, a \rangle)$. By the definition of $\mathbf{R}_B$, we have that $\prod_{i=1}^{n} \mu_i$ is in $\mathbf{R}_B(\langle \vec{s}, a \rangle)$. This proves that $\sigma(\langle r, a \rangle)$ is in $\mathbf{R}_B(\langle \mathsf{last}(r), a \rangle)$ whenever $\mathbf{R}_B(\langle \mathsf{last}(r), a \rangle)$ is non-empty.

Now assume that $\mathbf{R}_B(\langle \mathsf{last}(r), a \rangle)$ is empty. By Axiom (S1), we may conclude that $\mathsf{active}_B(\vec{s}) = 1$, in which case $\sigma(\langle r, a \rangle)$ is by definition undefined for all $a \in I_B$. This completes the proof that σ is a well-defined input scheduler for B.

For the output scheduler ρ, we need to show that $\rho(r) \in \mathbf{G}_B(\mathsf{last}(r))$ whenever $\rho(r)$ is defined. Therefore, we may focus on the case in which $\mathsf{active}_B(\vec{s}) = 1$. By the definition of S_B, there is in fact unique j with $\mathsf{active}_j(s_j) = 1$. Assume without loss that $\rho_j(\mathsf{proj}_j(r))$ is defined. By the definition of output schedulers, $\rho_j(\mathsf{proj}_j(r)) \in \mathbf{G}_j(s_j)$.

Moreover, we know that $\mathsf{active}_i(s_i) = 0$ for all $i \neq j$. Fix $a \in O_B \cup H_B$ and $i \neq j$. By Axiom (S1), $\mathbf{R}_i(\langle s_i, a \rangle)$ is non-empty whenever $a \in I_i$. This implies that $\sigma_i(\langle \mathsf{proj}_i(r), a \rangle)$ is defined and is in $\mathbf{R}_i(\langle s_i, a \rangle)$. Therefore, the family $\{\mu_{a,i}\}_{\langle a, i \rangle \in N_j}$ satisfies the conditions in Definition 9.2.2 and thus the bundle f generated by $\rho_j(\mathsf{proj}_j(r))$ and $\{\mu_{a,i}\}_{\langle a, i \rangle \in N_j}$ is in $\mathbf{G}_B(\mathsf{last}(r))$. This completes the proof that ρ is a well-defined input scheduler for B. □

Notice that Definition 10.3.2 and the proof of Lemma 10.3.5 rely on the definition of $\|$ and switch axioms, therefore they do not apply to PIOAs in general.

Roughly speaking, the parallel composition mechanism for PIOAs does not attempt to resolve global non-determinism, therefore it is not possible to combine two local schedules to form a single global schedule. The token structure of switched PIOAs serves precisely the purpose of eliminating such global non-determinism.

Definition 10.3.2 induces to a very natural notion of composition for switched probabilistic systems.

Definition 10.3.3. Let $\{\mathcal{A}_i \mid 1 \leq i \leq n\}$ be a set of probabilistic systems where $\mathcal{A}_i = \langle A_i, \mathcal{S}_i \rangle$ and $\{A_i \mid 1 \leq i \leq n\}$ are pairwise compatible switched PIOAs. The *parallel composite*, denoted $\|_{i=1}^{n} \mathcal{A}_i$, is the probabilistic system $\mathcal{B} = \langle B, \mathcal{T} \rangle$ defined as follows:

- the underlying switched PIOA is $B = \|_{i=1}^{n} A_i$;
- the set $\mathcal{T}$ of I/O schedulers contains precisely those $\langle \sigma, \rho \rangle$ generated by some family $\{\langle \sigma_i, \rho_i \rangle\}_{1 \leq i \leq n} \in \prod_{i=1}^{n} \mathcal{S}_i$.

Again, we adopt notational conventions as in the case of $\|$ for switched PIOAs. Commutativity and associativity follow similarly.

Before ending this section, let us briefly revisit automata Early', Late' and Coin' of Figure 10.1. Consider the full probabilistic systems induced by these automata (i.e., each automaton is paired with all possible local I/O schedulers). We claim that, when Late' and Coin' are composed using Definition 10.3.3, it is no longer possible to obtain the schedule depicted in Figure 8.2. This is because the local I/O scheduler of Late' must choose between b and c without "knowing" the random outcome in Coin'. Extending this intuition, it is not hard to show that $\mathsf{Early}' \parallel \mathsf{Coin}'$ and $\mathsf{Late}' \parallel \mathsf{Coin}'$ are equivalent in our external behavior semantics.

10.4 Compositionality

We proceed to state and prove our main theorem: the external behavior semantics for switched probabilistic systems (Definition 10.2.3) is compositional with respect to the composition operator introduced in Definition 10.3.3.

Theorem 10.4.1. *Let $\mathcal{A} = \langle A, \mathcal{S} \rangle$, $\mathcal{C} = \langle C, \mathcal{U} \rangle$ and $\mathcal{D} = \langle D, \mathcal{V} \rangle$ be switched probabilistic systems. Assume that $\mathcal{A}$ and $\mathcal{D}$ are comparable and $\mathsf{ExtBeh}(\mathcal{A}) \subseteq \mathsf{ExtBeh}(\mathcal{D})$. Moreover, assume that $\mathcal{C}$ is compatible with both $\mathcal{A}$ and $\mathcal{D}$. Then $\mathsf{ExtBeh}(\mathcal{A} \parallel \mathcal{C}) \subseteq \mathsf{ExtBeh}(\mathcal{D} \parallel \mathcal{C})$.*

To prove this theorem, we need quite a few auxiliary results. Recall from Definition 10.2.2 that likelihood assignments are defined in terms of minimal execution branches. We will start with a pasting result on minimal branches in a parallel

composite (Section 10.4.1, Lemma 10.4.4). Then, in Section 10.4.2, we consider pasting results for execution trees and likelihood assignments. That lays sufficient ground for the proof of Theorem 10.4.1 in Section 10.4.3.

Throughout the rest of this section, let A_1 and A_2 be compatible switched PIOAs and define $B := A_1 \parallel A_2$. Moreover, let $\langle \sigma_1, \rho_1 \rangle$ and $\langle \sigma_2, \rho_2 \rangle$ be I/O schedulers for A_1 and A_2, respectively, and let $\langle \sigma, \rho \rangle$ denote the I/O scheduler for B generated by $\langle \sigma_1, \rho_1 \rangle$ and $\langle \sigma_2, \rho_2 \rangle$ (cf. Definition 10.3.2).

10.4.1 Minimal Execution Branches

Lemma 10.4.2 below says, when we project a minimal branch in B onto one of its components, the result is always minimal. Lemma 10.4.3 states that, given $r_1 \in \mathsf{Bran}_{\min}(A_1)$ and $r_2 \in \mathsf{Bran}_{\min}(A_2)$ with matching traces, we can "zip" them together in a unique way to form a minimal branch in B. Finally, Lemma 10.4.4 states that, given a fixed trace α, there is a bijective correspondence between

- $\mathsf{tr}^{-1}_{\min}(\alpha)$ in B and
- the Cartesian product of $\mathsf{tr}^{-1}_{\min}(\mathsf{proj}_1(\alpha))$ in A_1 and $\mathsf{tr}^{-1}_{\min}(\mathsf{proj}_2(\alpha))$ in A_2.

Lemma 10.4.2. *For every minimal branch r in $\mathsf{Bran}(B)$, both $\mathsf{proj}_1(r)$ and $\mathsf{proj}_2(r)$ are minimal.*

Proof. Without loss of generality, we consider only $\mathsf{proj}_1(r)$. Recall that empty branches are always minimal, so we may focus on non-empty branches.

Consider a minimal branch of the form $r.a.\mu.\vec{t}$ and let $\vec{s}$ denote $\mathsf{last}(r)$. Notice that, a must be in I_B, hence in $I_1 \cup I_2$. There are two cases:

- $a \in I_1$. Then $\mathsf{proj}_1(r.a.\mu.\vec{t}) = \mathsf{proj}_1(r).a.\mathsf{proj}_1(\mu).t_1$, which is minimal because a is visible.
- $a \notin I_1$. Then $\mathsf{proj}_1(r.a.\mu.\vec{t}) = \mathsf{proj}_1(r)$. Note that $\mu \in \mathbf{R}_B(\langle \vec{s}, a \rangle)$. Therefore, by Axiom (2), we know that $\mathsf{active}_B(\vec{s}) = 0$. This implies $\mathsf{active}_1(\mathsf{last}(\mathsf{proj}_1(r))) = \mathsf{active}_1(s_1) = 0$. Therefore by Lemma 10.2.2 we know $\mathsf{proj}_1(r)$ is minimal.

Now we consider a minimal branch of the form $r.f.a.\vec{t}$ and again let $\vec{s}$ denote $\mathsf{last}(r)$. In this case, a must be in O_B, hence in $O_1 \cup O_2$. Here we have three cases.

- $a \in O_1$. Then $\mathsf{proj}_1(r.f.a.\vec{t}) = \mathsf{proj}_1(r).\mathsf{proj}_1(f).a.t_1$, which is minimal because a is visible.
- $a \in I_1$. Then $\mathsf{proj}_1(r.f.a.\vec{t}) = \mathsf{proj}_1(r).a.\mathsf{proj}_{a,1}(f).t_1$, which is minimal because a is visible.

– $a \notin I_1$. Then $\mathsf{proj}_1(r.f.a.\vec{t}) = \mathsf{proj}_1(r)$. Moreover, note that f must be generated by some $g_2 \in \mathbf{G}_2(s_2)$. Therefore, by Axiom (2), we know that $\mathsf{active}_2(s_2) = 1$. By the definition of S_B, we have $\mathsf{active}_1(\mathsf{last}(\mathsf{proj}_1(r))) = \mathsf{active}_1(s_1) = 0$. Again, by Lemma 10.2.2, we know $\mathsf{proj}_1(r)$ is minimal.

□

Lemma 10.4.3. *Let $\alpha \in (I_B \cup O_B)^{<\omega}$ be given. Let p be a minimal branch of A_1 such that $\mathsf{tr}(p) = \mathsf{proj}_1(\alpha)$. Similarly for q in A_2. There is a unique minimal branch r of B such that $\mathsf{proj}_1(r) = p$, $\mathsf{proj}_2(r) = q$, and $\mathsf{tr}(r) = \alpha$.*

Proof. We proceed by induction on the length of α. If α is empty, then, by minimality, p and q are both empty. Take r to be the empty branch in B.

Consider αa. Let p' be a minimal branch of A_1 with trace $\mathsf{proj}_1(\alpha a)$ and let p denote the unique minimal prefix of p' with trace $\mathsf{proj}_1(\alpha)$. Similarly for $q \sqsubseteq q'$ in A_2. By induction hypothesis, choose a unique minimal branch r such that $\mathsf{proj}_1(r) = p$, $\mathsf{proj}_2(r) = q$, and $\mathsf{tr}(r) = \alpha$.

First assume that a is in $O_1 \cup H_1$. We have two cases.

– $a \notin I_2$. Then $\mathsf{proj}_2(\alpha) = \mathsf{proj}_2(\alpha a)$. Therefore $q = q'$ and we take r' to be the unique extension of r in which A follows p' and B idles after q.

– $a \in I_2$. Then q' ends with an a-transition. Let q_0 be the one-step prefix of q'. By Lemma 10.1.1, we know that $\mathsf{active}_2(\mathsf{last}(q_0)) = 0$. By Lemma 10.2.2, q_0 is minimal and hence coincides with q. Take r' to be the unique extension of r, in which A_1 follows p' and A_2 idles after r until the last step (i.e., the a-step).

The case in which a is locally controlled by A_2 is symmetric. It remains to consider the case where a is an input of B. Again, if a is not in the signature of A_1, then $p = p'$; otherwise, $a \in I_1$ and we apply Lemma 10.1.1 and Lemma 10.2.2 to conclude that p is the one-step prefix of p'. Similarly for q and q'. Take r' to be the unique (one-step) extension of r in which 1. A_i takes an a-step after r, if $a \in I_i$; 2. A_i idles after r otherwise; □

Lemma 10.4.4. *Let X denote $\mathsf{tr}^{-1}_{\min}(\alpha)$ in B. Moreover, let Y and Z denote $\mathsf{tr}^{-1}_{\min}(\mathsf{proj}_1(\alpha))$ in A_1 and $\mathsf{tr}^{-1}_{\min}(\mathsf{proj}_2(\alpha))$ in A_2, respectively. There exists an isomorphism $\mathsf{zip} : Y \times Z \to X$ whose inverse is $\langle \mathsf{proj}_1, \mathsf{proj}_2 \rangle$.*

Proof. By Lemma 10.4.2 and Lemma 10.4.3. □

10.4.2 Execution Trees and Likelihood Assignments

For the rest of this section, let $\mathbf{Q}, \mathbf{Q}_1$ and $\mathbf{Q}_2$ be abbreviations for $\mathbf{Q}_{\sigma,\rho}, \mathbf{Q}_{\sigma_1,\rho_1}$ and $\mathbf{Q}_{\sigma_2,\rho_2}$, respectively. Similarly for $\mathbf{L}, \mathbf{L}_1$ and $\mathbf{L}_2$. Lemma 10.4.5 below

says an execution tree of the parallel composite can be obtained as a pointwise product of the execution trees of the components. Lemma 10.4.6 then combines Lemma 10.4.4 and Lemma 10.4.5 to show the analogous result for likelihood assignments.

Lemma 10.4.5. *For every r in $\mathsf{Bran}(B)$, we have*

$$\mathbf{Q}(r) = \mathbf{Q}_1(\mathsf{proj}_1(p)) \cdot \mathbf{Q}_2(\mathsf{proj}_2(p)).$$

Proof. If r is empty, $\mathbf{Q}(r) = 1 = \mathbf{Q}_1(\mathsf{proj}_1(r)) \cdot \mathbf{Q}_2(\mathsf{proj}_2(r))$.

Consider $r' = r.a.\mu.\vec{t}$ and let $\vec{s}$ denote $\mathsf{last}(r)$. By Definition 9.2.1, μ is of the form $\mu_1 \times \mu_2$, where $\mu_i = \mathsf{Dirac}(s_i)$ whenever $a \notin I_i$. Define c_i to be 0 if $a \in I_i$ but $\mu_i \neq \sigma_i(\langle \mathsf{proj}_i(r),\, a\rangle)$. Otherwise, c_i is 1. Then we have

$$\begin{aligned}
\mathbf{Q}(r') &= \mathbf{Q}(r) \cdot \mu(\vec{t}) \cdot c_1 \cdot c_2 && \text{definitions } \sigma, \mathbf{Q} \\
&= \mathbf{Q}_1(\mathsf{proj}_1(r)) \cdot \mathbf{Q}_2(\mathsf{proj}_2(r)) \cdot c_1 \cdot \mu_1(t_1) \cdot c_2 \cdot \mu_2(t_2) && \text{I.H.} \\
&= \mathbf{Q}_1(\mathsf{proj}_1(r')) \cdot \mathbf{Q}_2(\mathsf{proj}_2(r')) && \text{definitions } \mathbf{Q}_1, \mathbf{Q}_2
\end{aligned}$$

Next we consider $r' = r.f.a.\vec{t}$ and also let $\vec{s}$ denote $\mathsf{last}(r)$. Without loss of generality, assume that f is generated by some g_1 and $\{\mu_{b,2}\}_{\langle b,\, 2\rangle \in N_1}$. Notice that, if $b \notin I_2$, then $\mu_{b,2}$ must be $\mathsf{Dirac}(s_2)$.

Now define c_1 to be 0 if $g_1 \neq \rho_1(\mathsf{proj}_1(r))$ and 1 otherwise. Similarly, define c_2 to be 0 if $a \in I_2$ but $\mu_{a,2} \neq \sigma_2(\langle \mathsf{proj}_2(r),\, a\rangle)$. Otherwise, c_2 is 1. Similar to the previous case, we have

$$\begin{aligned}
&\mathbf{Q}(r') \\
&= \mathbf{Q}(r) \cdot f(\langle a,\, \vec{t}\rangle) \cdot c_1 \cdot c_2 && \text{definitions } \rho, \mathbf{Q} \\
&= \mathbf{Q}_1(\mathsf{proj}_1(r)) \cdot \mathbf{Q}_2(\mathsf{proj}_2(r)) \\
&\qquad \cdot c_1 \cdot g_1(\langle a,\, t_1\rangle) \cdot c_2 \cdot \mu_{a,2}(t_2) && \text{definition } f \text{ and I.H.} \\
&= \mathbf{Q}_1(\mathsf{proj}_1(r')) \cdot \mathbf{Q}_2(\mathsf{proj}_2(r')) && \text{definitions } \mathbf{Q}_1, \mathbf{Q}_2
\end{aligned}$$

□

Lemma 10.4.6. *Let $\alpha \in (I_B \cup O_B)^{<\omega}$ be given. We have*

$$\mathbf{L}(\alpha) = \mathbf{L}_1(\mathsf{proj}_1(\alpha)) \cdot \mathbf{L}_2(\mathsf{proj}_2(\alpha)).$$

Proof. Let X denote $\mathsf{tr}^{-1}_{\min}(\alpha)$ in B. Let Y and Z denote $\mathsf{tr}^{-1}_{\min}(\mathsf{proj}_1(\alpha))$ in A_1

and $\mathsf{tr}^{-1}_{\min}(\mathsf{proj}_2(\alpha))$ in A_2, respectively. We have

$$\begin{aligned}
\mathbf{L}(\alpha) &= \sum_{r \in X} \mathbf{Q}(r) && \text{definition } \mathbf{L} \\
&= \sum_{r \in X} \mathbf{Q}_1(\mathsf{proj}_1(r)) \cdot \mathbf{Q}_2(\mathsf{proj}_2(r)) && \text{Lemma 10.4.5} \\
&= \sum_{p \in Y, q \in Z} \mathbf{Q}_1(p) \cdot \mathbf{Q}_2(q) && \text{Lemma 10.4.4} \\
&= (\sum_{p \in Y} \mathbf{Q}_1(p)) \cdot (\sum_{q \in Z} \mathbf{Q}_2(q)) && \text{factorization} \\
&= \mathbf{L}_1(\mathsf{proj}_1(\alpha)) \cdot \mathbf{L}_2(\mathsf{proj}_2(\alpha)). && \text{definition } \mathbf{L}_1 \text{ and } \mathbf{L}_2
\end{aligned}$$

□

10.4.3 Main Proof

Proof of Theorem 10.4.1. First note that, if $\mathcal{A}$ and $\mathcal{D}$ are comparable and $\mathcal{C}$ is compatible with both $\mathcal{A}$ and $\mathcal{D}$, then $\mathcal{A} \parallel \mathcal{C}$ is comparable to $\mathcal{D} \parallel \mathcal{C}$.

Let $\mathbf{L} \in \mathsf{ExtBeh}(\mathcal{A} \parallel \mathcal{C})$ be given. We need to show that $\mathbf{L}$ is also in $\mathsf{ExtBeh}(\mathcal{D} \parallel \mathcal{C})$. Let $\langle \sigma, \rho \rangle$ be an I/O scheduler for $A \parallel C$ such that $\mathbf{L} = \mathsf{tr}(\mathbf{Q}_{\sigma,\rho})$. By the definition of $\parallel$ for probabilistic systems, we may choose $\langle \sigma_A, \rho_A \rangle \in \mathcal{S}$ and $\langle \sigma_C, \rho_C \rangle \in \mathcal{U}$ so that they generate $\langle \sigma, \rho \rangle$. Let $\mathbf{L}_A$ and $\mathbf{L}_C$ denote $\mathsf{tr}(\mathbf{Q}_{\sigma_A,\rho_A})$ and $\mathsf{tr}(\mathbf{Q}_{\sigma_C,\rho_C})$, respectively.

On the other hand, we know that $\mathbf{L}_A \in \mathsf{ExtBeh}(\mathcal{A}) \subseteq \mathsf{ExtBeh}(\mathcal{D})$. Therefore, we may choose $\langle \sigma_D, \rho_D \rangle \in \mathcal{V}$ such that $\mathbf{L}_D := \mathsf{tr}(\mathbf{Q}_{\sigma_D,\rho_D}) = \mathbf{L}_A$. Let $\langle \sigma', \rho' \rangle$ denote the I/O scheduler generated by $\langle \sigma_D, \rho_D \rangle$ and $\langle \sigma_C, \rho_C \rangle$ and write $\mathbf{L}'$ for $\mathsf{tr}(\mathbf{Q}_{\sigma',\rho'})$.

Now, let I denote $I_{A\|C} = I_{D\|C}$ and O denote $O_{A\|C} = O_{D\|C}$. Applying Lemma 10.4.6, we have, for all $\alpha \in (I \cup O)^{<\omega}$,

$$\mathbf{L}(\alpha) = \mathbf{L}_A(\mathsf{proj}_A(\alpha)) \cdot \mathbf{L}_C(\mathsf{proj}_C(\alpha)).$$

Since $\mathcal{A}$ and $\mathcal{D}$ have the same external signature, we know that $\mathsf{proj}_A(\alpha) = \mathsf{proj}_D(\alpha)$. Moreover, by the choice of $\langle \sigma_D, \rho_D \rangle$, we have $\mathbf{L}_A = \mathbf{L}_D$. Hence $\mathbf{L}_A(\mathsf{proj}_A(\alpha)) = \mathbf{L}_D(\mathsf{proj}_D(\alpha))$.

Applying Lemma 10.4.6 again, we have

$$\mathbf{L}(\alpha) = \mathbf{L}_A(\mathsf{proj}_A(\alpha)) \cdot \mathbf{L}_C(\mathsf{proj}_C(\alpha)) = \mathbf{L}_D(\mathsf{proj}_D(\alpha)) \cdot \mathbf{L}_C(\mathsf{proj}_C(\alpha)) = \mathbf{L}'(\alpha).$$

This proves that $\mathbf{L} = \mathbf{L}' \in \mathsf{ExtBeh}(\mathcal{D} \parallel \mathcal{C})$. □

10.5 Centralized Scheduling with Arbiters

Our switched PIOA framework implements a distributed scheduling scheme: components rely on a token structure to avoid conflicts and scheduling decisions are always made by the (unique) active component. Some may argue that such a scheduling scheme does not realistically represent situations such as asynchronous message passing via an unpredictable network. In response, we outline a setting in which a designated component takes on the role of an *arbiter*, which is responsible for all global scheduling decisions in the system. In other words, we use our switched PIOA framework to recreate a centralized interpretation of component scheduling. The obvious advantage is that our external behavior semantics is compositional and hence we can freely replace components with others that are behaviorally equivalent.

First, we fix a nonempty, finite index set $\mathcal{I}$ and assume that the universal set CAct of control actions is $\bigcup_{i\in I}\{\mathsf{go}_i, \mathsf{done}_i\}$. We restrict our attention to controllable automata, defined as follows.

Definition 10.5.1. Let A be a switched PIOA and let $i \in \mathcal{I}$ be given. We say that A is *controllable* for i provided:

1. A is initially inactive;
2. $\mathsf{CI}_A = \{\mathsf{go}_i\}$ and $\mathsf{CO}_A = \{\mathsf{done}_i\}$.

In other words, A has a limited control interface, $\{\mathsf{go}_i, \mathsf{done}_i\}$, and must wait for an activation signal at the beginning of each execution. Aside from these restrictions, A is free to communicate with other components (not necessarily the arbiter) via synchronization of basic actions.

Various requirements can be placed on the I/O schedulers for A. For example, we may require that A performs at most one locally controlled action during each activation. Or A may take a finite number of internal steps, possibly followed by a visible action. The latter can be seen as a *fairness* condition, so that no one component is allowed to retain the activity token indefinitely.

To compose a set of (pairwise compatible) controllable automata, we use an arbiter automaton, which models uncertainties in the parallel environment.

Definition 10.5.2. Let $X \subseteq \mathsf{BAct}$ be given. An *arbiter* for $\langle \mathcal{I}, X\rangle$ is a switched PIOA Arb satisfying the following:

1. $I_{\mathsf{Arb}} = \{\mathsf{done}_i \mid i \in \mathcal{I}\} \cup X$ and $O_{\mathsf{Arb}} = \{\mathsf{go}_i \mid i \in \mathcal{I}\}$;
2. $\mathsf{active}_{\mathsf{Arb}}(s^0_{\mathsf{Arb}}) = 1$.

Such an arbiter manages the flow of the activity token among components, so that token exchange does not take place directly between components. This is depicted in Figure 10.4 below.

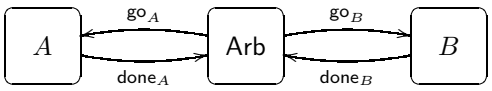

Figure 10.4: Arbitrated Composition

Different notions of parallel composition can be obtained by varying the choice of local I/O schedulers as well as arbiters. A simple example is the parameterized composition operator (cf. Section 8.3), which can be implemented with

- local I/O schedulers that always return control after one locally controlled move and
- an arbiter that schedules go_A with probability p and go_B with probability $1 - p$.

More complex examples can be obtained by varying the parameter X in Definition 10.5.2. This determines the observational power of the arbiter, i.e., the amount of information which can be used by the arbiter to make scheduling decisions. Such flexibility can be very useful when we wish to limit scheduling freedom in order to improve performance of algorithms. For example, the *write-oblivious* adversary model of [Cha96] requires that random outcomes cannot be used by adversaries until they are read by at least one process. This can be modeled by excluding all parameterized write actions from the set X.

10.6 Conclusions

We have presented the switched PIOA framework, which is designed for the purpose of modeling and analyzing stochastic systems. This framework accommodates both nondeterministic and probabilistic choices within components, and the associated notion of parallel composition is based on asynchronous communication under a distributed scheduling scheme. We define a trace-style semantics for this framework and prove it is compositional.

Throughout our development, a main focus is the notion of scheduling, that is, the mechanism with which non-deterministic choices are eliminated. Since the choices between parallel components are often considered nondeterministic, scheduling directly affects semantic behaviors of composite systems. However, in our experience with the literature, scheduling mechanisms are often just mentioned in passing, without due justification. Therefore, we provide a summary of some common scheduling schemes and try to compare them against our distributed scheduling scheme.

Compared to earlier versions [CLSV04a] and [CLSV04b], the current development presents several technical improvements. First of all, we introduce a new

formulation of PIOAs, applying I/O distinction to reactive and generative system types. Moreover, we have modified some of the defining axioms for switched PIOAs, simplifying the definition of external behavior. Finally, we provide a more flexible mechanism for reasoning with systems with open inputs. In particular, the notions of execution trees and likelihood assignments are directly defined for open components, without reference to closing contexts. This allows us to eliminate some of the cumbersome proofs involving renaming and hiding.

As for future research, we see much potential in the proposal of arbiters and controllable automata. We believe it can serve as a theoretical foundation in many application areas, including distributed consensus and process coordination. In particular, we would like to explore possibilities in modeling noisy scheduling [Asp00], as well as quantum-based and priority-based scheduling [AM99].

We are also interested in adapting the testing scenario of [SV03] to switched PIOAs. Since our semantics focuses on externally visible behavior, we expect to be able to derive a characterization based on frequencies of external observations.

11

Local-Oblivious Scheduling

In Chapter 8, we saw an example showing that the trace distribution semantics for probabilistic automata is not compositional (cf. Figures 8.1 and 8.2). We claimed that the main source of difficulty is the composition mechanism of probabilistic automata, where nondeterministic choices are resolved by a history-dependent adversary with perfect information. Such an adversary can make decisions in one component using internal information of the other, thus creating a form of "covert communication" between components (i.e., communication not described explicitly by action synchronization).

In this chapter, we explore the use of a weaker form of adversary, namely, local-oblivious adversaries. These adversaries have access to dynamic information that is visible from an external point of view. Local information, such as local coin tosses that have not been used in any meaningful way, is hidden from these adversaries. The main challenge is to capture these ideas formally, by imposing additional axioms on transition structures and modifying the composition mechanism. A semantic compositionality theorem follows quite easily once we have carefully implemented the idea of local-oblivious scheduling.

11.1 Introduction

Mathematically, an adversary is a function taking a finite execution of a system to an available next transition (or a convex combination of available transitions, depending on the particular formalism). Such a function resolves all non-deterministic choices in a system, so that random choices are the only remaining uncertainties. The probability of each execution is then completely determined by the sequence of coin tosses generating that execution.

Conceptually, adversaries are important for analyzing *worst-case* performance of distributed algorithms. One can image that the goal of an adversary is to prevent the protocol parties from completing their joint computational task (e.g., agreement). This is used to model adverse conditions in a real-life computational environment; for example, network congestion and process failures. By quantifying over all possible adversaries, we take into account the worst combination of events that can happen during an execution.

11.1.1 Adversary Models

As we mentioned in Chapter 8, different classes of adversaries can be obtained by varying the amount of history information that can be used by an adversary. Notice that, by restricting the knowledge of an adversary, we are essentially quantifying over a smaller class of adversaries. Therefore, it is often possible to obtain better complexity results by assuming that the adversaries "know" less. This has been studied extensively in the area of randomized consensus algorithms, where various forms of partial-information adversaries are referred to as *weak adversary models* [Asp03, AB04]. (Naturally, the perfect-information version is called the *strong adversary model*.)

Below we list a few common weak adversary models. They are primarily used to sidestep lower bound results proven for the strong model. More discussions and an example of a weak adversary algorithm can be found in Part III of this thesis.

(1) An *oblivious* adversary is a predetermined list of process names, thus it is completely independent of dynamic random choices [ABZ97].

(2) A *value-oblivious* adversary cannot observe values stored in memory registers and cannot distinguish operations that differ only in parameter [AKL99, AB04].

(3) A *write-oblivious* adversary cannot observe any value written to a memory register until that value has been read by some process [Cha96, Aum97].

(4) A *local-oblivious* adversary cannot observe any randomly chosen value until it is written to a shared memory [CIL87, CIL94].

11.1.2 Formal Modeling

Despite its importance in randomized distributed algorithms, the notion of scheduling has received little attention in the formal methods community. Most modeling frameworks we have seen (cf. Section 8.3) fall into one the following two categories.

- To make sure that parallel composition is well-defined, some particular mechanism is used to resolve nondeterministic choices [WSS94, DHK98, JLY01]. These mechanisms correspond to certain partial-information assumptions, although they were not identified as such in the original publications. To our best knowledge, none of these hidden assumptions resemble any weak adversary model actually used in the distributed computing literature.
- Nondeterministic choices remain in the formal models, then by default perfect-information adversaries (i.e., strong adversaries) are used in the underlying semantics [Seg95b, BK98, DHK98].

These observations suggest that scheduling has not been taken as a fundamental aspect of probabilistic modeling frameworks. A notable exception is [MMS03], where schedulers are modeled explicitly by Markov chains with an underlying semantics of probabilistic polynomial time Turing machines[1]. In order to carry out computational analysis of cryptographic protocols, these schedulers are required to satisfy a host of independence assumptions, some of which (e.g., environment independence) are clearly too strong for non-cryptographic analysis.

Another exception is [dA99], in which partial-information schedules (or policies) are obtained via a partition of the state space. Such a partition is viewed as an "indistinguishability" relation, and a partial information policy must make the same decision if two state sequences are indistinguishable. This is essentially the same idea behind *partially observable MDPs (POMDPs)*, originally studied in the context of reinforcement learning [KLA98].

We attribute the apparent simplicity of this POMDP approach to a rather strong assumption that is found in both [dA99] and [KLA98]. Namely, indistinguishable states must have the same set of enabled actions. This assumption makes sense in the original setting of planning, where an agent tries to choose among a number of actions in such a way that its expected reward is always maximized. However, it leads to transition structures that are too rigid for our modeling purposes: in many randomized algorithms, random outcomes *do* affect enabled operations. For example, in the cooperative sharing algorithm of Aumann and Kapah-Levy [AKL99], processes use random choices to determine which buffers will be read next.

While it is theoretically possible to relax the enabling assumption of [dA99], it has not been carried out in the literature (as far as we know). We conjecture that the resulting semantics is no longer simple and natural. This is because, if the partial-information adversary schedules an action that is not enabled, then certain rules must be invoked to produce a "default" behavior. Most likely, it is a "null" action that do not cause any state changes[2]. This may still give the adversary some back-door access to local information, because it can infer from these "null" actions that certain operations are not enabled.

Thus, in our opinion, an explicit and systematic treatment of partial-information scheduling is yet lacking in the formal methods literature. This makes it difficult to formally verify the weak adversary algorithms mentioned earlier. Currently, the best strategy is to model the algorithms in a perfect-information formalism and then rely on ad-hoc tricks to enforce partial-information assumptions. This is done, for instance, when we verify our own randomized consensus algorithm using the model checker PRISM [Che05c, PRI] (cf. Chapter 12).

In the present chapter, we experiment with a new approach of obtaining partial-information schedules. Namely, we restrict our framework in such a way that

[1] Other security-related formalisms, such as [Can01, BPW04b], follow a different modeling paradigm: distributed scheduling. We refer to Chapter 10 for more discussions.

[2] This is similar to the approach taken in [CCK+06b], where scheduled actions that are not enabled are simply absorbed/ignored.

certain types of branching structures simply cannot be expressed. We argue that the chosen branching structures correspond to local information that should be hidden from the adversary. As a result, when we quantify over all possible adversaries in the new framework, we obtain precisely those that are local-oblivious.

The goal of this exercise is two-fold. First, local-oblivious scheduling is generally regarded as a reasonable alternative to perfect-information scheduling, because it does not weaken the adversary excessively. Therefore, a formal framework for local-oblivious scheduling may prove useful in applications. Adding to the appeal, we present a compositionality theorem for our trace-style semantics. This, as discussed in Chapter 8, is not possible under perfect-information scheduling.

Second, we are hopeful that the same strategy can be used to model other weak adversary assumptions. That is, given a particular weak adversary model, we try to identify branching structures that correspond to hidden information. Once these branching structures are removed from the specifications, we can quantify as usual over all possible schedules, without violating the weak adversary assumption. This may provide a reprieve from the intractability results for partial-information model checking [dA99]. More precisely, it may help to reduce certain partial-information problems to perfect-information ones, which can then be modeled checked by a tool such as PRISM. (Of course, in light of [dA99], this cannot be done in full generality, but partial results can often be beneficial.)

11.1.3 Local-Oblivious Adversaries

Local-oblivious scheduling is the first weak adversary assumption used in the randomized consensus literature [CIL87, CIL94]. The rationale behind it is very simple: if a randomly chosen value simply sits in a local register, without being operated on or communicated to another process, then we have no conceivable reason to believe that it may affect global dynamics.

Our formal development takes place within the PIOA framework introduced in Chapter 9. To capture the idea of local-oblivious scheduling, we follow two essential steps:

(i) first, we impose several partial-information axioms on the transition structures of PIOAs, so that local information is hidden from the global adversary;

(ii) then, we define parallel composition in the "schedule-and-compose" style (cf. Chapter 10), so that local nondeterministic choices are resolved using local information only.

Notice that, if all individual components are purely probabilistic, then Step (ii) is not necessary. This is typically the case when we model a single protocol,

where every participant is fully specified up to coin tosses. However, when the target systems become more complex, we often try to abstract away from implementation details by grouping together smaller components to form a bigger component. In that case, Step (ii) becomes significant, because it enforces scoping rules for scheduling decisions. This allows us to separate global nondeterministic choices from local ones, without sacrificing the flexibility to treat a composite of multiple components as yet a single component.

11.2 Partial-Information Axioms

Recall from Chapter 9 that a PIOA A is a tuple $\langle S_A, s_A^0, I_A, O_A, H_A, \mathbf{R}_A, \mathbf{G}_A \rangle$, where $\mathbf{R}_A : S_A \times I_A \to \mathcal{P}(\mathsf{Disc}(S_A))$ is the reactive transition structure and $\mathbf{G}_A : S_A \to \mathcal{P}(\mathsf{Disc}((O_A \cup H_A) \times S_A))$ is the generative transition structure (cf. Definition 9.1.2). Throughout this chapter, we will assume that $\mathbf{R}_A$ satisfies the usual input-enabling axiom of IOA [LT89]. In that case, we say that A is *input-enabled*.

- **Input enabling**: $\mathbf{R}_A(s, a) \neq \emptyset$ for all $s \in S_A$ and $a \in I_A$.

In addition, we impose the following partial-information axioms.

(P1) For all $s \in S_A$, $a \in I_A$ and $\mu \in \mathbf{R}_A(s, a)$, there is $t \in S_A$ such that $\mu = \mathsf{Dirac}(t)$.

(P2) For all $s \in S_A$ and $f \in \mathbf{G}_A(s)$, if there exist $a \in H_A$ and $t \in S_A$ such that $f(a, t) > 0$, then $f = \mathsf{Dirac}(\langle a, t \rangle)$.

(P3) For all $s \in S_A$ and $f \in \mathbf{G}_A(s)$, if there exist $a \in O_A$ and $t_1, t_2 \in S_A$ such that $f(a, t_1) > 0$ and $f(a, t_2) > 0$, then $t_1 = t_2$.

Thus, we have essentially three types of transitions: *input*, *hidden* and *output*. For the first two types of transitions, probabilistic branching is completely forbidden (Axioms (P1) and (P2)). For output transitions, probabilistic branching is allowed only if all actions labels are distinct (Axiom (P3)).

Figure 11.1 below illustrates some examples of branching structures that are ruled out by Axioms (P1), (P2) and (P3).

Figure 11.1: Invisible probabilistic branching.

Notice, all four examples in Figure 11.1 share a common trait: although the specification contains no nondeterministic choices, there exist "truly distinct" branches (or paths) with the same trace. Here "truly distinct" means that the two branches are are incomparable under the prefix relation on branches.

In future discussions, we shall refer to this property as *invisible probabilistic branching*. The idea is, even though these branches correspond to mutually exclusive events, the difference cannot be observed from an external point of view. In other words, branching structures of this type are internal aspects of a system, which are abstracted away in a trace-style semantics.

It is not hard to see that the presence of invisible probabilistic branching gives the adversary "backdoor" access to local information (hence our axioms are designed specifically to eliminate invisible probabilistic branching). We illustrate this point with the following example.

Example 11.2.1. Consider processes A, B and Coin_0 illustrated in Figure 11.2. Here A and B are racing to write their own preference values into a shared memory register. The third process Coin_0 tosses a fair coin and announces the result by sending a one-bit message across a network to an unspecified recipient.

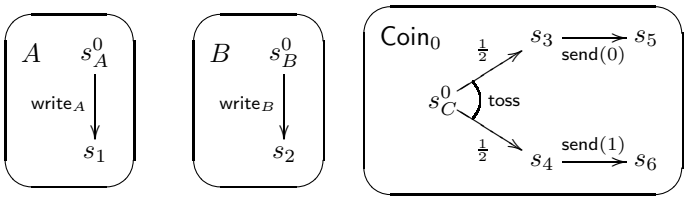

Figure 11.2: Automata A, B and Coin_0.

Notice, we have chosen a specification Coin_0 that exhibits invisible probabilistic branching (violating Axiom (P3)). Whenever the toss transition is scheduled, we obtain two "truly distinct" branches with the same trace: one ending in state s_3 and the other ending in state s_4.

Since adversaries are functions from finite branches to available next transitions, the mere presence of two distinct branches allows the adversary to make decisions based on the outcome of toss. For a concrete example, we define an adversary $\mathcal{A}$ as follows.

(i) At the initial state $\langle s_A^0, s_B^0, s_C^0\rangle$, $\mathcal{A}$ schedules the toss transition in Coin_0.

(ii) At the branch $\langle s_A^0, s_B^0, s_C^0\rangle.f.\mathsf{toss}.\langle s_A^0, s_B^0, s_3\rangle$, $\mathcal{A}$ schedules write_A followed by write_B. (Here f is the transition bundle generated by the toss transition in Coin_0, with both A and B stuttering in the same state.)

(iii) At the branch $\langle s_A^0, s_B^0, s_C^0\rangle.f.\mathsf{toss}.\langle s_A^0, s_B^0, s_4\rangle$, $\mathcal{A}$ schedules write_B followed by write_A.

(iv) Finally, $\mathcal{A}$ schedules the $\mathsf{send}(-)$ transition in Coin_0.

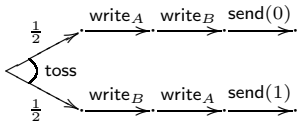

Figure 11.3: A probabilistic execution of $A \| B \| \mathsf{Coin}_0$.

Figure 11.3 depicts the probabilistic execution induced by $\mathcal{A}$. Clearly, $\mathcal{A}$ is not local-oblivious, because it uses the outcome of toss *before* that outcome is made public via the $\mathsf{send}(-)$ action.

This example shows that the amount of local information available to an adversary is very sensitive to the branching structures of our formal specifications. This observation motivates our underlying approach of restricted branching. Moreover, it is easy to construct a similar example if either Axiom (P1) or Axiom (P2) is removed. Therefore, all three axioms are necessary in capturing the notion of local-oblivious scheduling.

11.3 External Behavior

We define a notion of external behavior using I/O schedulers introduced in Chapter 9. Recall that an input scheduler resolves nondeterministic choices among input transitions carrying the same label, while an output scheduler resolves nondeterministic choices among locally controlled transition bundles. Since neither of these resolves nondeterministic choices between inputs and local activities, an I/O scheduler for an open PIOA does not always induce a probability measure.

In the case of switched PIOAs of Chapter 10, the token structure eliminates nondeterministic choices between inputs and locally-controlled activities. Therefore, an I/O scheduler for an open switched PIOA does induce a collection of sub-probability measures (cf. Proposition 10.2.1). Here, we prove that Proposition 10.2.1 holds even without the token structure of Chapter 10, as long as we strengthen our notion of I/O schedulers to eliminate nondeterministic choices between inputs and hidden transitions.

Definition 11.3.1. Let A be an input-enabled PIOA satisfying Axioms (P1)–(P3). An I/O scheduler $\langle \sigma, \rho \rangle$ for A is said to be *determinate* if the following conditions hold for all $r \in \mathsf{Bran}(A)$.

(i) Either $\sigma(r, a) = \bot$ for all $a \in I_A$, or $\sigma(r, a) \neq \bot$ for all $a \in I_A$;

(ii) If $\rho(r)$ is an output transition bundle, then $\sigma(r, a) \neq \bot$ for all $a \in I_A$.

(iii) If $\rho(r)$ is a hidden transition, then $\sigma(r, a) = \perp$ for all $a \in I_A$.

Thus, the behavior of a determinate I/O scheduler at a branch r falls into one of the following four categories:

- All activities are halted. That is, $\rho(r) = \perp$ and $\sigma(r, a) = \perp$ for all $a \in I_A$. This models a completely terminated process.
- All locally-controlled activities are halted, but all inputs are still enabled. That is, $\rho(r) = \perp$, but $\sigma(r, a) \neq \perp$ for all $a \in I_A$. This models an idle process that is waiting for an input signal.
- An output bundle is scheduled by ρ. In that case, all inputs are enabled by σ. This models an active process ready to communicate with other processes.
- A hidden transition is scheduled by ρ. In that case, all inputs are blocked by σ. This models an active process that blocks inputs temporarily in order to finish certain internal computation.

Intuitively, the last category reflects the fact that real-life implementations use deterministic mechanisms to handle input events. For example, FIFO queues may be used to store interrupting inputs, so that they can be handled when internal computations are finished. Since we quantify over all determinate I/O schedulers, we still take into account all possible situations (e.g., all input-handling policies). Moreover, we will show in Lemma 11.3.4 that, if A is not completely halted and not stuck in an infinite execution of hidden transitions, then eventually all inputs are accepted.

Example 11.3.1. Consider the PIOA illustrated in Figure 11.4 on the left. The execution tree shown in the middle is induced by an I/O scheduler that is *not* determinate: at every branch, the output scheduler schedules the internal loop but the input scheduler accepts the input a. A determinate I/O scheduler would induce an execution tree such as the one shown on the right.

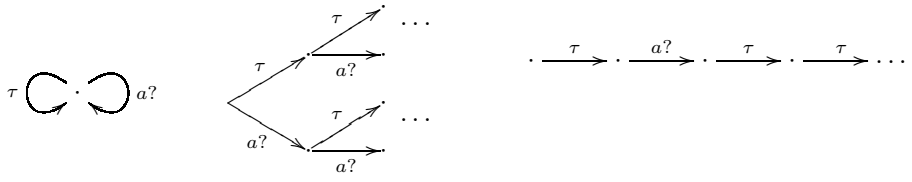

Figure 11.4: Interrupting inputs.

We now state an analog of Proposition 10.2.1 for determinate I/O schedulers. Observe that Proposition 11.3.1 fails without the addition assumption that $\langle \sigma, \rho \rangle$ is determinate. The execution tree illustrated in the middle of Figure 11.4 is a counterexample.

Proposition 11.3.1. *Let A be an input-enabled PIOA satisfying Axioms (P1)–(P3) and let $\langle \sigma, \rho \rangle$ be a determinate I/O scheduler for A. Let $\alpha \in (I_A \cup O_A)^{<\omega}$ be given and assume that $\mathsf{tr}^{-1}(\alpha)$ in A is nonempty. Then the restriction of $\mathbf{Q}_{\sigma,\rho}$ to $\mathsf{tr}^{-1}_{\min}(\alpha)$ is a discrete sub-probability distribution.*

The proof of Proposition 11.3.1 relies on Lemma 11.3.2 below.

Lemma 11.3.2. *Let A be an input-enabled PIOA satisfying Axioms (P1)–(P3) and let $\langle \sigma, \rho \rangle$ be a determinate I/O scheduler for A. The following holds for all $r \in \mathsf{Bran}(A)$.*

(i) For every $n \in \mathbb{N}$, there is at most one n-step extension r_1 of r such that $\mathsf{tr}(r_1) = \mathsf{tr}(r)$ and $\mathbf{Q}_{\sigma,\rho}(r_1) > 0$. If such r_1 exists, then $\mathbf{Q}_{\sigma,\rho}(r_1) = \mathbf{Q}_{\sigma,\rho}(r)$.

(ii) For every $a \in I_A$, there is at most one $r_1 \in \mathsf{Bran}_{\min}(A)$ such that $r \sqsubseteq r_1$, $\mathsf{tr}(r_1) = \mathsf{tr}(r)a$, and $\mathbf{Q}_{\sigma,\rho}(r_1) > 0$. If such r_1 exists, then $\mathbf{Q}_{\sigma,\rho}(r_1) = \mathbf{Q}_{\sigma,\rho}(r)$.

Proof. We prove Item (i) by induction on n. The base case is trivial. For the inductive step, suppose there are two such $(n+1)$-step extensions r_1 and r_2. By the induction hypothesis, there is at most one such n-step extension r'. Therefore, it must be the case that r' is a prefix of both r_1 and r_2. By the definition of $\mathbf{Q}_{\sigma,\rho}$ and Axiom (P2), we have $r_1 = r'.\rho(r').\tau.t = r_2$, where $\langle \tau, t \rangle$ is the unique member of $\mathsf{Supp}(\rho(r'))$. By the induction hypothesis, $\mathbf{Q}_{\sigma,\rho}(r_1) = \mathbf{Q}_{\sigma,\rho}(r') = \mathbf{Q}_{\sigma,\rho}(r)$.

For Item (ii), suppose again there are two such branches r_1 and r_2. Let $r' \in \mathsf{Bran}(A)$ be the longest common prefix of r_1 and r_2 such that $\mathsf{tr}(r) = \mathsf{tr}(r')$. Clearly, $r \sqsubseteq r'$. Consider the following three cases.

- r' is a one-step prefix of both r_1 and r_2. Then, by the definition of $\mathbf{Q}_{\sigma,\rho}$ and Axiom (P1), we have $r_1 = r'.a.\sigma(r').t = r_2$, where t is the unique member of $\mathsf{Supp}(\sigma(r'))$. Then $\mathbf{Q}_{\sigma,\rho}(r_1) = \mathbf{Q}_{\sigma,\rho}(r') = \mathbf{Q}_{\sigma,\rho}(r)$, where the second equality follows from Item (i),
- r' is a one-step prefix of neither r_1 nor r_2. Then, by the maximality of r', there exist $r'_1 \sqsubseteq r_1$ and $r'_2 \sqsubseteq r_2$ such that (i) both r'_1 and r'_2 extend r' with a hidden transition, (ii) $r'_1 \neq r'_2$, and (iii) $\mathbf{Q}_{\sigma,\rho}(r'_1) > 0$ and $\mathbf{Q}_{\sigma,\rho}(r'_2) > 0$. This yields a contradiction to Item (i).
- r' is a one-step prefix of either r_1 or r_2, but not both. Without loss, assume it is a one-step prefix of r_1. By the definition of $\mathbf{Q}_{\sigma,\rho}$ and Axiom (P1), r_1 must be of the form $r'.a.\sigma(r').t$, where t is the unique member

of $\mathsf{Supp}(\sigma(r'))$. On the other hand, since r' is not a one-step prefix of r_2, there exists $r_2' \sqsubseteq r_2$ such that r_2' is a one-step extension of r', $\mathbf{Q}_{\sigma,\rho}(r_2') > 0$, and $\mathsf{tr}(r_2') = \mathsf{tr}(r')$. This implies that $\rho(r')$ is a hidden transition, contradicting our assumption that $\langle \sigma, \rho \rangle$ is determinate.

□

Proof of Proposition 11.3.1. We proceed by induction on the length of α. If α is empty, then $\mathsf{tr}^{-1}_{\min}(\alpha)$ contains a unique element, namely, $\underline{s^0_A}$. Our claim holds because by definition $\mathbf{Q}_{\sigma,\rho}(\underline{s^0_A}) = 1$.

Consider α' of the form αa. We have two cases.

- $a \in I_A$. Let $r \in \mathsf{tr}^{-1}_{\min}(\alpha)$ be given. By Lemma 11.3.2, there is at most one $r' \in \mathsf{tr}^{-1}_{\min}(\alpha')$ with $r \sqsubseteq r'$. Moreover, if such r' exists, Lemma 11.3.2 guarantees that $\mathbf{Q}_{\sigma,\rho}(r') = \mathbf{Q}_{\sigma,\rho}(r)$. Therefore,

$$\sum_{r' \in \mathsf{tr}^{-1}_{\min}(\alpha')} \mathbf{Q}_{\sigma,\rho}(r') \leq \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} \mathbf{Q}_{\sigma,\rho}(r) \leq 1,$$

  where the second inequality follows from the induction hypothesis.

- $a \in O_A$. Note that Lemma 10.2.4 applies to PIOAs in general. Therefore, we have

$$\begin{aligned}\sum_{r' \in \mathsf{tr}^{-1}_{\min}(\alpha')} \mathbf{Q}_{\sigma,\rho}(r') &= \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} \sum_{r' \in \mathsf{tr}^{-1}_{\min}(\alpha a),\ r \sqsubseteq r'} \mathbf{Q}_{\sigma,\rho}(r') \\ &\leq \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} \mathbf{Q}_{\sigma,\rho}(r) \leq 1.\end{aligned}$$

  Again, the last inequality follows from the induction hypothesis.

□

By virtue of Proposition 11.3.1, we may define likelihood assignments as in Chapter 10. Again, these are behavioral abstractions of execution trees and are analogous to trace distributions for probabilistic automata. We refer to Section 10.2.2 for a more detailed discussion.

Definition 11.3.2. Let A be an input-enabled PIOA satisfying Axioms (P1)–(P3) and let $\langle \sigma, \rho \rangle$ be a determinate I/O scheduler for A. The *likelihood assignment* induced by $\langle A, \sigma, \rho \rangle$, denoted $\mathbf{L}_{\sigma,\rho}$, is the function $\mathsf{tr}(\mathbf{Q}_{\sigma,\rho}) : (I_A \cup O_A)^{<\omega} \to [0,1]$ given as follows.

$$\mathsf{tr}(\mathbf{Q}_{\sigma,\rho})(\alpha) := \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} \mathbf{Q}_{\sigma,\rho}(r).$$

External behavior and implementation can now be defined accordingly.

Definition 11.3.3. Let $\mathcal{A} = \langle A, \mathcal{S} \rangle$ be a probabilistic system such that: (i) A is an input-enabled PIOA satisfying Axioms (P1)–(P3) and (ii) every $\langle \sigma, \rho \rangle \in \mathcal{S}$ is determinate. An *external behavior* of $\mathcal{A}$ is a likelihood assignment $\mathbf{L}_{\sigma,\rho}$ induced by some $\langle \sigma, \rho \rangle \in \mathcal{S}$. We write $\mathsf{ExtBeh}(\mathcal{A})$ for the set of all external behaviors of $\mathcal{A}$.

Definition 11.3.4. Let $\mathcal{A} = \langle A, \mathcal{S} \rangle$ and $\mathcal{B} = \langle B, \mathcal{T} \rangle$ be probabilistic systems satisfying the two conditions in Definition 11.3.3. They are said to be *comparable* if $I_A = I_B$ and $O_A = O_B$. If, in addition, we have $\mathsf{ExtBeh}(\mathcal{A}) \subseteq \mathsf{ExtBeh}(\mathcal{B})$, then $\mathcal{A}$ is said to *implement* $\mathcal{B}$. This is denoted $\mathcal{A} \leq_{\mathbf{L}} \mathcal{B}$.

Below we make some useful observations about execution trees and likelihood assignments.

Proposition 11.3.3. *Let A be an input-enabled PIOA satisfying Axioms (P1)–(P3) and let $\langle \sigma, \rho \rangle$ be a determinate I/O scheduler for A. The following holds for every $\alpha \in (I_A \cup O_A)^{<\omega}$.*

(i) There is at most one $r \in \mathsf{tr}^{-1}_{\min}(\alpha)$ such that $\mathbf{Q}_{\sigma,\rho}(r) > 0$. (Clearly, if such r exists, then $\mathbf{L}_{\sigma,\rho}(\alpha) = \mathbf{Q}_{\sigma,\rho}(r)$; otherwise, $\mathbf{L}_{\sigma,\rho}(\alpha) = 0$.)

(ii) The set $X = \{r' \in \mathsf{tr}^{-1}(\alpha) \mid \mathbf{Q}_{\sigma,\rho}(r') > 0\}$ is linearly ordered by the prefix relation $\sqsubseteq$.

(iii) If X is nonempty, then $\mathbf{Q}_{\sigma,\rho}(r') = \mathbf{Q}_{\sigma,\rho}(r)$ for every $r' \in X$, where r is the unique minimal element in X.

Proof. We prove these claims simultaneously by induction on the length of α. For the base case, Item (i) follows from the definitions of $\mathbf{Q}_{\sigma,\rho}$ and $\mathbf{L}_{\sigma,\rho}$, and Items (ii) and (iii) follows from Item (i) of Lemma 11.3.2.

For the inductive step, consider α' of the form αa. By the induction hypothesis, there is at most one $r \in \mathsf{tr}^{-1}_{\min}(\alpha)$ with $\mathbf{Q}_{\sigma,\rho}(r) > 0$. If such r does not exists, then the claims for α' hold trivially. Suppose otherwise; that is, there is unique such r. Then, for every $r' \in \mathsf{tr}^{-1}(\alpha')$, $\mathbf{Q}_{\sigma,\rho}(r') > 0$ implies $r \sqsubseteq r'$.

First we consider Item (i). If $a \in I_A$, it is sufficient to apply Item (ii) of Lemma 11.3.2. Therefore we may focus on the case in which $a \in O_A$. Suppose there are two distinct branches $r_1, r_2 \in \mathsf{tr}^{-1}_{\min}(\alpha')$ with $\mathbf{Q}_{\sigma,\rho}(r_1) > 0$ and $\mathbf{Q}_{\sigma,\rho}(r_2) > 0$. By uniqueness of r, we know that r is a prefix of both r_1 and r_2. Let $r' \in \mathsf{Bran}(A)$ be the longest common prefix of r_1 and r_2. Clearly, $r \sqsubseteq r'$. We have the following three cases.

- r' is a one-step prefix of both r_1 and r_2. Then, by the definition of $\mathbf{Q}_{\sigma,\rho}$ and Axiom (P3), we have $r_1 = r'.\rho(r').a.t = r_2$, where t is the unique state with $\langle a, t \rangle \in \mathsf{Supp}(\rho(r'))$. This contradicts the assumption that $r_1 \neq r_2$.

- r' is a one-step prefix of neither r_1 nor r_2. Then, by the maximality of r', there exist $r'_1 \sqsubseteq r_1$ and $r'_2 \sqsubseteq r_2$ such that (i) both r'_1 and r'_2 extend r' with a hidden transition, (ii) $r'_1 \neq r'_2$, and (iii) $\mathbf{Q}_{\sigma,\rho}(r'_1) > 0$ and $\mathbf{Q}_{\sigma,\rho}(r'_2) > 0$. This yields a contradiction to Item (ii) of the induction hypothesis.

- r' is a one-step prefix of either r_1 or r_2, but not both. Without loss, assume it is a one-step prefix of r_1. By the definition of $\mathbf{Q}_{\sigma,\rho}$ and Axiom (P3), r_1 must be of the form $r'.\rho(r').a.t$, where t is the unique state with $\langle a, t\rangle \in \mathsf{Supp}(\rho(r'))$. In particular, we may conclude that $\rho(r')$ is not a hidden transition. On the other hand, since r' is not a one-step prefix of r_2, there exists $r'_2 \sqsubseteq r_2$ such that r'_2 is a one-step extension of r', $\mathbf{Q}_{\sigma,\rho}(r'_2) > 0$, and $\mathsf{tr}(r'_2) = \mathsf{tr}(r')$. This implies that $\rho(r')$ is a hidden transition, which is a contradiction.

This finishes the proof of Item (i) for α'. Items (ii) and (iii) again follow from Item (i) of Lemma 11.3.2. □

Proposition 11.3.3 is a strong (and perhaps surprising) result. It says, given a determinate I/O scheduler $\langle\sigma, \rho\rangle$ and a finite trace α, there is "essentially" at most one branch r with trace α that is reachable under $\langle\sigma, \rho\rangle$. We say "essentially" because there may in fact be more than one, but they differ in a very limited way: if r_1 and r_2 have the same trace and are both reachable under $\langle\sigma, \rho\rangle$, then one must be a prefix of the other. This fact will prove essential in our proof of compositionality.

The following lemmas are also used to prove compositionality. Lemma 11.3.4 says, if A has a nonempty input signature but at least one input is blocked, then all inputs are blocked. Moreover, if A has any visible behavior, then all inputs are eventually accepted. These are direct consequences of the definition of determinate I/O schedulers.

Lemma 11.3.4. *Let A be an input-enabled PIOA satisfying Axioms (P1)–(P3) and let $\langle\sigma, \rho\rangle$ be a determinate I/O scheduler for A. Let $\alpha \in (I_A \cup O_A)^{<\omega}$ be given.*

(i) Suppose there is $a \in I_A$ with $\mathbf{L}_{\sigma,\rho}(\alpha a) = 0$. Then the same holds for all $a \in I_A$.

(ii) Suppose there is $a \in I_A \cup O_A$ with $\mathbf{L}_{\sigma,\rho}(\alpha a) > 0$. Then the same holds for all $a \in I_A$.

Proof. To prove Item (i), we assume for the sake of contradiction that there is $a' \in I_A$ with $\mathbf{L}_{\sigma,\rho}(\alpha a') > 0$. Then we may choose $r' \in \mathsf{tr}^{-1}_{\min}(\alpha a')$ with $\mathbf{Q}_{\sigma,\rho}(r') > 0$. Let r denote the unique one-step prefix of r'. Then we know $\sigma(r, a') \neq \bot$. By the definition of determinate I/O schedulers, we have $\sigma(r, a'') \neq \bot$ for all $a'' \in I_A$. In particular, $\sigma(r, a) \neq \bot$ and thus $\mathbf{Q}_{\sigma,\rho}(r.a.\sigma(r, a).t) > 0$, where t is the unique state in $\mathsf{Supp}(\sigma(r, a))$. This contradicts our assumption on a.

For Item (ii), choose $a \in I_A \cup O_A$ with $\mathbf{L}_{\sigma,\rho}(\alpha a) > 0$. If $a \in I_A$, then the desired claim follows from Item (i) of the present lemma. Otherwise, we may choose $r' \in \mathsf{tr}^{-1}_{\min}(\alpha a)$ with $\mathbf{Q}_{\sigma,\rho}(r') > 0$. Let r be the unique one-step prefix of r'. Then $\mathbf{Q}_{\sigma,\rho}(r') > 0$ and $\rho(r)$ is an output transition bundle. By the definition of determinate I/O schedulers, we may conclude that $\sigma(r, a') \neq \bot$ for all $a' \in I_A$. Then, for all $a' \in I_A$, $\mathbf{Q}_{\sigma,\rho}(r.a'.\sigma(r,a').t_{a'}) > 0$, where $t_{a'}$ is the unique state in $\mathsf{Supp}(\sigma(r,a'))$. Therefore $\mathbf{L}_{\sigma,\rho}(\alpha a') > 0$ for all $a' \in I_A$. □

Lemma 11.3.5 is the obvious fact that, if some output action a occurs after a trace α, then we can find a branch r_0 with trace α such that an output bundle containing a is executed after r_0.

Lemma 11.3.5. *Let A be an input-enabled PIOA satisfying Axioms (P1)–(P3) and let $\langle \sigma, \rho \rangle$ be a determinate I/O scheduler for A. Let $\alpha \in (I_A \cup O_A)^{<\omega}$ and $a \in O_A$ be given. Suppose $\mathbf{L}_{\sigma,\rho}(\alpha a)$ is nonzero. Then the set $\{r \in \mathsf{tr}^{-1}(\alpha) \mid \mathbf{Q}_{\sigma,\rho}(r) > 0\}$ has a unique maximal element, call it r_0. Moreover, $\rho(r_0)$ is an output bundle in $\mathbf{G}_A(\mathsf{last}(r_0))$ and there exists unique $s' \in S_A$ such that $\langle a, s' \rangle \in \mathsf{Supp}(\rho(r_0))$.*

Proof. Since $\mathbf{L}_{\sigma,\rho}(\alpha a)$ is nonzero, there is $r' \in \mathsf{tr}^{-1}_{\min}(\alpha a)$ with $\mathbf{Q}_{\sigma,\rho}(r') > 0$. Then, by the definition of $\mathbf{Q}_{\sigma,\rho}$, we know that r' is of the form $r_0.\rho(r_0).a.s'$, where $\rho(r_0)$ is an output bundle in $\mathbf{G}_A(\mathsf{last}(r_0))$ and s' is the unique state with $\langle a, s' \rangle \in \mathsf{Supp}(\rho(r_0))$. Also, we have $\mathsf{tr}(r_0) = \alpha$ and $\mathbf{Q}_{\sigma,\rho}(r_0) > 0$, hence the set $\{r \in \mathsf{tr}^{-1}(\alpha) \mid \mathbf{Q}_{\sigma,\rho}(r) > 0\}$ is nonempty. It remains to show r_0 is indeed the unique maximal element of this set.

By Proposition 11.3.3, this set is linearly ordered by prefix. Suppose it contains some proper extension of r_0. Let r'' denote the shortest such proper extension. Then r'' is of the form $r_0.\rho(r_0).b.s''$ with $b \in H_A$ and $\langle b, s'' \rangle \in \mathsf{Supp}(\rho(r_0))$. Since $\langle a, s' \rangle$ is also in $\mathsf{Supp}(\rho(r_0))$, this yields a contradiction to Axiom (P2). Therefore r_0 must be maximal. □

11.4 Parallel Composition and Compositionality

In Chapter 9, we defined parallel composition for PIOAs. Here we extend that definition to probabilistic systems and prove that the implementation relation introduced in Section 11.3 above is compositional.

11.4.1 Composition of Probabilistic Systems

Let us recall the intuition behind the notion of probabilistic systems: given a probabilistic system $\langle A, \mathcal{S} \rangle$, we think of the I/O schedulers in $\mathcal{S}$ as "acceptable" schedules for A. When we compose two systems $\langle A, \mathcal{S} \rangle$ and $\langle B, \mathcal{T} \rangle$, the underlying PIOA is defined to be $A \upuparrows B$ (cf. Section 9.2). The acceptable schedules for $\mathcal{A} \upuparrows B$ are those generated, in some appropriate sense, from schedules in $\mathcal{S}$

and $\mathcal{T}$, respectively. Thus, our notion of parallel composition for probabilistic systems is also along the lines of "schedule-and-compose" as in Chapter 10.

To carry out this development formally, we need to first make sure that all axioms of the present chapter are preserved under the parallel operator $\uparrow\uparrow$.

Proposition 11.4.1. *Let A and B be compatible PIOAs. Assume both are input-enabled and satisfy Axioms (P1)– (P3). Then the same hold for $A \uparrow\uparrow B$.*

Proof. For input enabling, let a be an input action of $A \uparrow\uparrow B$. If a is in the signature of A, then every state s of A enables an a-transition. Similarly for B. By the definition of $\mathbf{R}_{A\uparrow\uparrow B}$ (Definition 9.2.1), every state of $A \uparrow\uparrow B$ enables an a-transition.

Axiom (P1) also follows easily from the definition of $\mathbf{R}_{A\uparrow\uparrow B}$. For Axiom (P2), note that every hidden action a of $\mathbf{R}_{A\uparrow\uparrow B}$ is in the signature of either A or B, but not both. Without loss of generality, suppose a is a hidden action of A. By the definition of $\mathbf{G}_{A\uparrow\uparrow B}$ (Definition 9.2.2), B must stutter in every a-transition of $A \uparrow\uparrow B$. Since A satisfies Axiom (P2), this implies $A \uparrow\uparrow B$ also satisfies Axiom (P2).

Finally, note that every output transition bundle f of $A \uparrow\uparrow B$ is generated by an output transition bundle g of either A or B, but not both. Without loss, suppose g is an output bundle of A. Since A satisfies Axiom (P3) and B satisfies Axiom (P1), it is easy to see that f satisfies the condition in Axiom (P3). □

Next, we make precise the meaning of "generated" I/O schedulers, using projection operators defined in Section 9.2.

Definition 11.4.1. Let A and B be compatible PIOAs. Assume both are input-enabled and satisfy Axioms (P1)–(P3). Let $\langle \sigma, \rho \rangle$, $\langle \sigma_A, \rho_A \rangle$ and $\langle \sigma_B, \rho_B \rangle$ be determinate I/O schedulers for $A \uparrow\uparrow B$, A and B, respectively. Then $\langle \sigma, \rho \rangle$ is said to be *generated* by $\langle \sigma_A, \rho_A \rangle$ and $\langle \sigma_B, \rho_B \rangle$ if the following hold for all $r \in \mathsf{Bran}(A \uparrow\uparrow B)$.

- For all $a \in I_{A\uparrow\uparrow B}$, if $\sigma(r, a) \neq \perp$ then:
  - if $a \in I_A$, then $\sigma_A(\mathsf{proj}_A(r), a)$ is defined and equals $\mathsf{proj}_A(\sigma(r, a))$;
  - similarly if $a \in I_B$.
- If $\rho(r)$ is a hidden transition, then:
  - if $\rho(r)$ is generated by a hidden transition in A, then $\rho_A(\mathsf{proj}_A(r))$ is defined and equals $\mathsf{proj}_A(\rho(r))$;
  - similarly if $\rho(r)$ is generated by a hidden transition in B.
- If $\rho(r)$ is an output transition bundle, then the following hold.

- If $\rho(r)$ is generated by an output bundle in A, then $\rho_A(\mathsf{proj}_A(r))$ is defined and equals $\mathsf{proj}_A(\rho(r))$. Moreover, for all $a \in I_B$, if there exists $t \in S_{A \uparrow\uparrow B}$ with $\langle a, t\rangle \in \mathsf{Supp}(\rho(r))$, then $\sigma_B(\mathsf{proj}_B(r), a)$ is defined and equals $\mathsf{proj}_{a,B}(\rho(r))$.
- Similarly if $\rho(r)$ is generated by an output bundle in B.

Notice that the same pair of I/O schedulers $\langle \sigma_A, \rho_A\rangle$ and $\langle \sigma_B, \rho_B\rangle$ may generate many different I/O schedulers in the composite. This is because the choices between A and B may be resolved differently by an implicit global adversary. This is a sharp contrast to the composition of switched PIOAs, where a token structure ensures that every pair of local I/O schedulers generate a unique I/O scheduler for the composite (cf. Chapter 10).

We are now ready to define parallel composition for probabilistic systems.

Definition 11.4.2. Let $\mathcal{A} = \langle A, \mathcal{S}\rangle$ and $\mathcal{B} = \langle B, \mathcal{T}\rangle$ be probabilistic systems. Assume that:

– both are input-enabled and satisfy Axioms (P1)–(P3); and

– every $\langle \sigma, \rho\rangle \in \mathcal{S}$ is determinate and the same holds for $\mathcal{T}$.

The systems $\mathcal{A}$ and $\mathcal{B}$ are said to be *compatible* if A and B are as PIOAs. In that case, their *parallel composite*, denoted $\mathcal{A}\|\mathcal{B}$, is the probabilistic system $\langle A \uparrow\uparrow B, \mathcal{U}\rangle$, where $\mathcal{U}$ is the set of all determinate I/O schedulers $\langle \sigma, \rho\rangle$ for $A \uparrow\uparrow B$ such that $\langle \sigma, \rho\rangle$ is generated by some $\langle \sigma_A, \rho_A\rangle \in \mathcal{S}$ and $\langle \sigma_B, \rho_B\rangle \in \mathcal{T}$ in the sense of Definition 11.4.1.

Using commutativity and associativity of $\uparrow\uparrow$, it is routine to check that $\|$ is also commutative and associative. To illustrate some essential features of $\|$, we revisit Example 11.2.1, replacing Coin_0 with a specification that does *not* exhibit invisible probabilistic branching.

Example 11.4.1. Consider Coin_1 shown in Figure 11.5. Notice, the states of

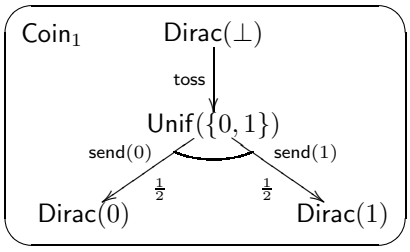

Figure 11.5: Automaton Coin_1.

Coin_1 are represented by discrete probability distributions. Intuitively, Coin_1

models a randomized algorithm with a boolean state variable bit, which is initially $\perp$. After the toss action, bit is assigned either 0 or 1, each with probability $\frac{1}{2}$; hence the end state of toss is represented by the uniform distribution on $\{0, 1\}$, denoted $\mathsf{Unif}(\{0, 1\})$.

This modeling style captures the idea that, if the outcome of toss is considered "local" (or "private"), then as far as the adversary is concerned the value of bit is random at any time *after* toss and *before* $\mathsf{send}(-)$. Once the $\mathsf{send}(-)$ action takes place, the value of bit is known to the adversary and therefore no longer random. (This interpretation is similar to the notion of "belief states" for POMDPs [KLA98].)

Now we compose Coin_1 with automata A and B of Example 11.2.1, using the new composition operator $\|$. Since all three automata are purely probabilistic, the associated probabilistic systems each contain just one schedule.

We argue that the total correlation illustrated in Figure 11.3 is no longer possible. To see this, we try to recreate scheduling decisions made by the adversary $\mathcal{A}$:

- First, the toss transition is scheduled.
- Immediately afterwards, a write action occurs (either write_A or write_B). Since the toss transition does not cause any branching, there is a unique execution with trace toss; namely,

  $$\langle s_A^0,\, s_B^0,\, \mathsf{Dirac}(\perp)\rangle.f.\,\mathsf{toss}\,.\langle s_A^0,\, s_B^0,\, \mathsf{Unif}(\{0,1\})\rangle.$$

  (Again, f here is the transition bundle generated by the toss transition in Coin_1, with both A and B stuttering.) Therefore, at this point, the adversary chooses either write_A or write_B, but not both.
- The remaining write transition is scheduled.
- Finally, the $\mathsf{send}(-)$ transition is scheduled.

This gives rise to two possible execution trees, illustrated in Figure 11.6. Neither exhibits a total correlation between the temporal ordering of write_A and write_B and the message content in $\mathsf{send}(-)$.

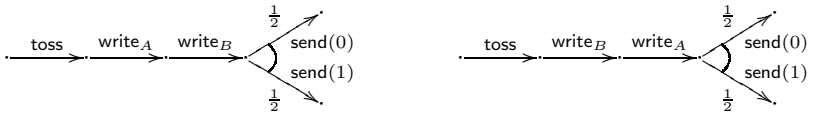

Figure 11.6: Two possible executions of $A\|B\|\,\mathsf{Coin}_1$.

We examine one more possible scenario: send(−) occurs before either of the write actions. In this case, the value of bit is revealed to the adversary through the branching structure induced by the send(−) transition. As a result, the adversary *can* use the message content to decide whether A or B wins. This is depicted in Figure 11.7.

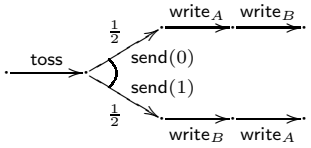

Figure 11.7: Random outcome available to adversary *after* announcement.

The scenario in Figure 11.7 shows that the adversary model associated with our parallel operator $\|$ is *dynamic*, in that the adversaries can in fact use dynamic random outcomes. We have simply delayed its knowledge by restricting the branching structures of our specifications.

Thus, we advocate the following for practical modeling: probabilistic branching should be avoided unless an externally visible effect is intended. For example, suppose we allow *intentionally* the possibility that the outcome of toss produces an immediate effect on the system dynamics. Then the two branches of toss should be labeled with different actions, as shown in Figure 11.8 below. It is not hard to see that an execution tree similar to the one in Figure 11.3 is possible in $A\|B\|$ Coin$_2$.

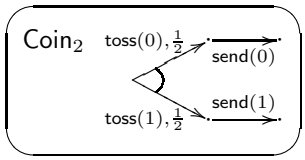

Figure 11.8: Automaton Coin$_2$.

This also explains our view that, in some situations (e.g., the original specification Coin$_0$), the combination of trace-style semantics and perfect-information scheduling conveys conflicting intentions.

- On the one hand, the branching of toss is abstracted away in the trace-style semantics. This suggests that such branching is local information and is considered "irrelevant" with respect to the evolution of other components. That is, such branching produces no effects outside the current component.
- On the other hand, adversaries have access to the branching of toss and may use that information to schedule other components. This models

situations in which the outcome of toss immediately produces an effect on the dynamics of the whole system.

11.4.2 Auxiliary Results

First we prove two "decomposition" results: one for execution trees and one for likelihood assignments. These are analogous to Lemmas 10.4.5 and 10.4.6 for switched PIOAs.

Proposition 11.4.2. *Let A and B be compatible PIOAs. Assume both are input-enabled and satisfy Axioms (P1)–(P3). Let I denote $I_{A\Uparrow B}$ and O denote $O_{A\Uparrow B}$. Let $\langle \sigma, \rho\rangle$, $\langle \sigma_A, \rho_A\rangle$ and $\langle \sigma_B, \rho_B\rangle$ be determinate I/O schedulers for $A \Uparrow B$, A and B, respectively. Suppose $\langle \sigma, \rho\rangle$ is generated by $\langle \sigma_A, \rho_A\rangle$ and $\langle \sigma_B, \rho_B\rangle$ as in Definition 11.4.1. Then the following hold.*

(i) For all $r \in \mathsf{Bran}(A \Uparrow B)$, if $\mathbf{Q}_{\sigma,\rho}(r) > 0$, then

$$\mathbf{Q}_{\sigma,\rho}(r) = \mathbf{Q}_{\sigma_A,\rho_A}(\mathsf{proj}_A(r)) \cdot \mathbf{Q}_{\sigma_B,\rho_B}(\mathsf{proj}_B(r)).$$

(ii) For every $\alpha \in (I \cup O)^{<\omega}$, if $\mathbf{L}_{\sigma,\rho}(\alpha) > 0$, then

$$\mathbf{L}_{\sigma,\rho}(\alpha) = \mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha)) \cdot \mathbf{L}_{\sigma_B,\rho_B}(\mathsf{proj}_B(\alpha)).$$

Proof. We prove Item (i) by induction on the length of r. The base case is trivial, since every execution tree assigns 1 to the empty execution.

Consider nonempty r' with $\mathbf{Q}_{\sigma,\rho}(r') > 0$. We have three cases.

- $r' = r.a.\sigma(r,a).s'$ for some $a \in I$ and $s' \in \mathsf{Supp}(\sigma(r,a))$. If $a \in I_A$, then $\mathsf{proj}_A(r')$ is of the form $\mathsf{proj}_A(r).a.\mathsf{proj}_A(\sigma(r,a)).\mathsf{proj}_A(s')$; otherwise, it is simply $\mathsf{proj}_A(r)$. Since $\langle \sigma, \rho\rangle$ is generated by $\langle \sigma_A, \rho_A\rangle$ and $\langle \sigma_B, \rho_B\rangle$, we have $\mathsf{proj}_A(\sigma(r,a)) = \sigma_A(\mathsf{proj}_A(r), a))$.

  Let c_A denote $\sigma_A(\mathsf{proj}_A(r), a)(\mathsf{proj}_A(s'))$ if $a \in I_A$ and 1 otherwise. Similarly for c_B. By the definition of $\mathbf{R}_{A\Uparrow B}$ (Definition 9.2.1), we know that $\sigma(r,a)(s') = c_A \cdot c_B$. Applying the induction hypothesis, we have:

$$\begin{aligned}
\mathbf{Q}_{\sigma,\rho}(r') &= \mathbf{Q}_{\sigma,\rho}(r) \cdot \sigma(r,a)(s') \\
&= \mathbf{Q}_{\sigma,\rho}(r) \cdot c_A \cdot c_B \\
&= \mathbf{Q}_{\sigma_A,\rho_A}(\mathsf{proj}_A(r)) \cdot \mathbf{Q}_{\sigma_B,\rho_B}(\mathsf{proj}_B(r)) \cdot c_A \cdot c_B \\
&= \mathbf{Q}_{\sigma_A,\rho_A}(\mathsf{proj}_A(r')) \cdot \mathbf{Q}_{\sigma_B,\rho_B}(\mathsf{proj}_B(r'))
\end{aligned}$$

- $r' = r.\rho(r).a.s'$, where $\langle a, s'\rangle \in \mathsf{Supp}(\rho(r))$ and $\rho(r)$ is generated by some $g \in \mathbf{G}_A(\mathsf{last}(\mathsf{proj}_A(r)))$. Since $\langle \sigma, \rho\rangle$ is generated by $\langle \sigma_A, \rho_A\rangle$ and $\langle \sigma_B, \rho_B\rangle$, we have $\rho_A(\mathsf{proj}_A(r)) = \mathsf{proj}_A(\rho(r)) = g$ and hence

$$\mathsf{proj}_A(r') = \mathsf{proj}_A(r).\rho_A(\mathsf{proj}_A(r)).a.\mathsf{proj}_A(s').$$

Let c_A denote $\rho_A(\mathsf{proj}_A(r))(a, \mathsf{proj}_A(s'))$.

If $a \in I_B$, then $\mathsf{proj}_B(r')$ is of the form $\mathsf{proj}_B(r).a.\mathsf{proj}_{a,B}(\rho(r)).\mathsf{proj}_B(s')$; otherwise, it is simply $\mathsf{proj}_B(r)$. Moreover, since $\langle \sigma, \rho \rangle$ is generated by $\langle \sigma_A, \rho_A \rangle$ and $\langle \sigma_B, \rho_B \rangle$, we have $\mathsf{proj}_{a,B}(\rho(r)) = \sigma_B(\mathsf{proj}_B(r), a)$ in case $a \in I_B$. Let c_B denote $\sigma_B(\mathsf{proj}_B(r), a)(\mathsf{proj}_B(s'))$ if $a \in I_B$ and 1 otherwise.

By the definition of $\mathbf{G}_{A \upuparrows B}$ (Definition 9.2.2), we know that $\rho(r)(a, s') = c_A \cdot c_B$. Applying the induction hypothesis, we have:

$$\begin{aligned}
\mathbf{Q}_{\sigma,\rho}(r') &= \mathbf{Q}_{\sigma,\rho}(r) \cdot \rho(r)(a, s') \\
&= \mathbf{Q}_{\sigma,\rho}(r) \cdot c_A \cdot c_B \\
&= \mathbf{Q}_{\sigma_A,\rho_A}(\mathsf{proj}_A(r)) \cdot \mathbf{Q}_{\sigma_B,\rho_B}(\mathsf{proj}_B(r)) \cdot c_A \cdot c_B \\
&= \mathbf{Q}_{\sigma_A,\rho_A}(\mathsf{proj}_A(r')) \cdot \mathbf{Q}_{\sigma_B,\rho_B}(\mathsf{proj}_B(r'))
\end{aligned}$$

– $r' = r.\rho(r).a.s'$, where $\langle a, s' \rangle \in \mathsf{Supp}(\rho(r))$ and $\rho(r)$ is generated by some $g \in \mathbf{G}_B(\mathsf{last}(\mathsf{proj}_B(r)))$. This is symmetric to the previous case.

This concludes the proof of Item (i).

For Item (ii), suppose we have α with $\mathbf{L}_{\sigma,\rho}(\alpha) > 0$. By Proposition 11.3.3, we may choose a unique branch $r \in \mathsf{tr}^{-1}_{\min}(\alpha)$ such that $\mathbf{L}_{\sigma,\rho}(\alpha) = \mathbf{Q}_{\sigma,\rho}(r) > 0$. By Item (i), we have

$$\mathbf{L}_{\sigma,\rho}(\alpha) = \mathbf{Q}_{\sigma,\rho}(r) = \mathbf{Q}_{\sigma_A,\rho_A}(\mathsf{proj}_A(r)) \cdot \mathbf{Q}_{\sigma_B,\rho_B}(\mathsf{proj}_B(r)).$$

Therefore, $\mathbf{Q}_{\sigma_A,\rho_A}(\mathsf{proj}_A(r)) > 0$. By Items (i) and (iii) of Proposition 11.3.3, we have

$$\begin{aligned}
&\mathbf{Q}_{\sigma_A,\rho_A}(\mathsf{proj}_A(r)) \\
&= \mathbf{L}_{\sigma_A,\rho_A}(\mathsf{tr}(\mathsf{proj}_A(r))) = \mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\mathsf{tr}(r))) = \mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha)).
\end{aligned}$$

Similarly for B. Therefore

$$\mathbf{L}_{\sigma,\rho}(\alpha) = \mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha)) \cdot \mathbf{L}_{\sigma_B,\rho_B}(\mathsf{proj}_B(\alpha)).$$

□

We have some more lemmas that will be used to prove compositionality. Intuitively, Lemma 11.4.3 says the following: if some output action a of component A occurs with nonzero probability, then we may infer that an output bundle of A is executed and hence every other action in the support of that bundle also occurs with nonzero probability. Moreover, none of the actions occurring with nonzero probability is in O_B.

Lemma 11.4.3. *Let A, B, I, O, $\langle \sigma, \rho \rangle$, $\langle \sigma_A, \rho_A \rangle$ and $\langle \sigma_B, \rho_B \rangle$ be given as in Proposition 11.4.2. Let $\alpha \in (I \cup O)^{<\omega}$ and $a \in O_A$ be given. Assume that $\mathbf{L}_{\sigma,\rho}(\alpha a) > 0$. For every $a' \in O_A$, we have $\mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha)a') > 0$ implies $\mathbf{L}_{\sigma,\rho}(\alpha a') > 0$. Moreover, for every $b \in O_B$, $\mathbf{L}_{\sigma,\rho}(\alpha b) = 0$.*

Proof. By Lemma 11.3.5, we may choose unique maximal $r_0 \in \mathsf{Bran}(A \Uparrow B)$ and $s' \in S_{A \Uparrow B}$ with

- $\mathsf{tr}(r_0) = \alpha$ and $\mathbf{Q}_{\sigma,\rho}(r_0) > 0$,
- $\rho(r_0)$ is an output bundle in $\mathbf{G}_{A \Uparrow B}(\mathsf{last}(r_0))$; and
- $\langle a, s' \rangle \in \mathsf{Supp}(\rho(r_0))$.

Since $a \in O_A$, $\rho(r_0)$ must be generated by an output bundle in A. By Definition 11.4.1, $\rho(r_0)$ is generated by $\rho_A(\mathsf{proj}_A(r_0))$. Note that $\mathsf{tr}(\mathsf{proj}_A(r_0)) = \mathsf{proj}_A(\mathsf{tr}(r_0)) = \mathsf{proj}_A(\alpha)$. Moreover, by Proposition 11.4.2 and the fact that $\mathbf{Q}_{\sigma,\rho}(r_0) > 0$, we have $\mathbf{Q}_{\sigma_A,\rho_A}(\mathsf{proj}_A(r_0)) > 0$.

Now let $a' \in O_A$ be given and assume that $\mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha)a') > 0$. By Lemma 11.3.5, we may choose unique maximal $r_A \in \mathsf{Bran}(A)$ with $\mathsf{tr}(r_A) = \mathsf{proj}_A(\alpha)$ and $\mathbf{Q}_{\sigma_A,\rho_A}(r_A) > 0$. By maximality of r_A, it must be the case that $\mathsf{proj}_A(r_0) \sqsubseteq r_A$. Since $\rho_A(\mathsf{proj}_A(r_0))$ is an output bundle, this implies $\mathsf{proj}_A(r_0) = r_A$ and therefore $\rho_A(\mathsf{proj}_A(r_0)) = \rho_A(r_A)$.

Note that Lemma 11.3.5 also guarantees the existence of $t_A \in S_A$ with $\langle a', t_A \rangle \in \mathsf{Supp}(\rho_A(r_A)) = \mathsf{Supp}(\rho_A(\mathsf{proj}_A(r_0)))$. Since $\rho(r_0)$ is generated by $\rho_A(\mathsf{proj}_A(r_0))$, there is $t \in S_{A \Uparrow B}$ with $\langle a', t \rangle \in \mathsf{Supp}(\rho(r_0))$, with $\mathsf{proj}_A(t) = t_A$. Then we have $\mathbf{Q}_{\sigma,\rho}(r_0.\rho(r_0).a'.t) > 0$ and $\mathsf{tr}(r_0.\rho(r_0).a'.t) = \alpha a'$. This implies $\mathbf{L}_{\sigma,\rho}(\alpha a') > 0$.

Finally, $\mathbf{L}_{\sigma,\rho}(\alpha b) = 0$ for every $b \in O_B$ simply because $\rho(r_0)$ is generated by an output bundle in A. □

Lemma 11.4.4 says, if component A has a nonempty input signature but it refuses all inputs indefinitely, then A no longer exhibits visible behavior. Moreover, if such A is composed with B, then the joint execution may continue only if $I_A \subseteq O_B$ and B does not attempt to synchronize with A.

Lemma 11.4.4. *Let A, B, I, O, $\langle \sigma, \rho \rangle$, $\langle \sigma_A, \rho_A \rangle$ and $\langle \sigma_B, \rho_B \rangle$ be given as in Proposition 11.4.2. Let $\alpha \in (I \cup O)^{<\omega}$ be given. Assume that I_A is nonempty but $\mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha)a) = 0$ for all $a \in I_A$.*

(i) For all $a' \in O_A$, $\mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha)a') = 0$.

(ii) Suppose there is $b \in I \cup O$ with $\mathbf{L}_{\sigma,\rho}(\alpha b) > 0$. Then $I_A \subseteq O_B$. Moreover, for every $b' \in I \cup O$, $\mathbf{L}_{\sigma,\rho}(\alpha b') > 0$ implies $b' \notin I_A \cup O_A$.

Proof. For Item (i), suppose for the sake of contradiction there is $a' \in O_A$ with $\mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha)a') > 0$. By Lemma 11.3.5, we may choose $r_0 \in \mathsf{Bran}(A)$ such that $\mathsf{tr}(r_0) = \mathsf{proj}_A(\alpha)$, $\mathbf{Q}_{\sigma_A,\rho_A}(r_0) > 0$, and $\rho_A(r_0)$ is an output bundle.

Since $\langle \sigma_A, \rho_A \rangle$ is determinate, we know that $\sigma_A(r_0, a) \neq \bot$ for all $a \in I_A$. By assumption, I_A is nonempty, hence we may choose $a \in I_A$. Let t denote the unique state in $\mathsf{Supp}(\sigma(r_0, a))$. Then $\mathbf{Q}_{\sigma_A,\rho_A}(r_0.a.\sigma(r_0, a).t) > 0$, contradicting the assumption that $\mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha)a) = 0$ for all $a \in I_A$.

For Item (ii), let such b be given. Choose $r' \in \mathsf{tr}^{-1}_{\min}(\alpha b)$ in $A \uparrow\uparrow B$ with $\mathbf{Q}_{\sigma,\rho}(r') > 0$. Let r be the one-step prefix of r'. Then either $\sigma(r, b) \neq \bot$ or $\rho(r) \neq \bot$. In both cases, the definition of determinate I/O schedulers implies that $\sigma(r, a) \neq \bot$ for all $a \in I$. Therefore, $\mathbf{L}_{\sigma,\rho}(\alpha a) > 0$ for all $a \in I$.

Suppose there exists $a \in I_A \cap I$. Applying Proposition 11.4.2, this implies $\mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha)a) > 0$, contradicting the assumption on $\mathbf{L}_{\sigma_A,\rho_A}$. Therefore, $I_A \cap I$ is empty and thus $I_A \subseteq O_B$.

Finally, let $b' \in I \cup O$ be given and assume that $\mathbf{L}_{\sigma,\rho}(\alpha b') > 0$. By Proposition 11.4.2, we know that $\mathbf{L}_{\sigma_A,\rho_A}(\mathsf{proj}_A(\alpha b')) > 0$. If $b' \in O_A$, then $\mathsf{proj}_A(\alpha b') = \mathsf{proj}_A(\alpha)b'$, contradicting Item (i) of the present lemma. Similarly, $b' \in I_A$ contradicts the assumption on $\mathbf{L}_{\sigma_A,\rho_A}$. Therefore, $b' \notin I_A \cup O_A$. □

11.4.3 Compositionality Theorem

We now state and prove our main theorem. Namely, the external behavior semantics for probabilistic systems is compositional.

Theorem 11.4.5. *Let $\mathcal{A} = \langle A, \mathcal{S}\rangle$, $\mathcal{B} = \langle B, \mathcal{T}\rangle$ and $\mathcal{C} = \langle C, \mathcal{U}\rangle$ be probabilistic systems satisfying the following.*

- *A, B and C are input-enabled and satisfy Axioms (P1)–(P3).*
- *Every I/O scheduler in $\mathcal{S}$ is determinate, and the same holds for $\mathcal{T}$ and $\mathcal{U}$.*
- *A and B are comparable, and both are compatible with C.*

Then we have $\mathcal{A} \leq_{\mathbf{L}} \mathcal{B}$ implies $\mathcal{A}\|\mathcal{C} \leq_{\mathbf{L}} \mathcal{B}\|\mathcal{C}$.

The rest of Section 11.4.3 is devoted to the proof of Theorem 11.4.5. First, we fix some notations and lay out the proof structure.

By the assumptions of Theorem 11.4.5, $\mathcal{A}\|\mathcal{C}$ and $\mathcal{B}\|\mathcal{C}$ are comparable. Let $I := I_{A\uparrow\uparrow C} = I_{B\uparrow\uparrow C}$ and $O := O_{A\uparrow\uparrow C} = O_{B\uparrow\uparrow C}$.

Let $\langle \sigma_0, \rho_0\rangle$ be a determinate I/O scheduler for $A \uparrow\uparrow C$ that is generated by some $\langle \sigma_A, \rho_A\rangle \in \mathcal{S}$ and $\langle \sigma_C, \rho_C\rangle \in \mathcal{U}$. For brevity, let $\mathbf{L}_0$, $\mathbf{L}_A$, $\mathbf{L}_C$ denote $\mathbf{L}_{\sigma_0,\rho_0}$, $\mathbf{L}_{\sigma_A,\rho_A}$, $\mathbf{L}_{\sigma_C,\rho_C}$, respectively. Similarly for $\mathbf{Q}_0$, $\mathbf{Q}_A$ and $\mathbf{Q}_C$.

By assumption, there is $\langle \sigma_B, \rho_B\rangle \in \mathcal{T}$ such that $\mathbf{L}_{\sigma_B,\rho_B} = \mathbf{L}_A$. Let $\mathbf{L}_B$ denote $\mathbf{L}_{\sigma_B,\rho_B}$ and $\mathbf{Q}_B$ denote $\mathbf{Q}_{\sigma_B,\rho_B}$. Our goal is to construct a determinate I/O scheduler $\langle \sigma, \rho\rangle$ for $B \uparrow\uparrow C$ such that $\langle \sigma, \rho\rangle$ is generated by $\langle \sigma_B, \rho_B\rangle$ and $\langle \sigma_C, \rho_C\rangle$. This will be done recursively.

Notice that, given $r \in \mathsf{Bran}(B \uparrow\uparrow C)$, if $\langle \sigma, \rho\rangle$ has been specified at every proper prefix of r, then $\mathbf{Q}_{\sigma,\rho}(r)$ is well defined. This holds even for the empty branch $\underline{s^0_{B\uparrow\uparrow C}}$ because $\mathbf{Q}_{\sigma,\rho}(\underline{s^0_{B\uparrow\uparrow C}}) = 1$ by definition.

Along with the recursive construction of $\langle \sigma, \rho \rangle$, we prove the following claim inductively.

(I) For every $r \in \mathsf{Bran}(B \uparrow\uparrow C)$, if $\langle \sigma, \rho \rangle$ has been specified at every proper prefix of r and $\mathbf{Q}_{\sigma,\rho}(r) > 0$, then

$$\mathbf{Q}_{\sigma,\rho}(r) = \mathbf{Q}_B(\mathsf{proj}_B(r)) \cdot \mathbf{Q}_C(\mathsf{proj}_C(r)).$$

Notice this holds trivially for the empty branch $\underline{s^0_{B \uparrow\uparrow C}}$.

Recursive Construction of $\langle \sigma, \rho \rangle$

We now proceed with the construction. Let $r \in \mathsf{Bran}(B \uparrow\uparrow C)$ be given and assume $\langle \sigma, \rho \rangle$ is defined at every prefix of r. The following list of clauses is used to determine $\langle \rho, \sigma \rangle$. By default, a later clause is entered only if all previous clauses are not applicable. (That is, these clauses are prioritized in the order given below.)

(1) $\mathbf{Q}_{\sigma,\rho}(r) = 0$, or $\mathbf{L}_0(\mathsf{tr}(r)a) = 0$ for all $a \in I \cup O$.

Then $\rho(r) := \bot$ and $\sigma(r, a) := \bot$ for all $a \in I$.

(2) $\rho_B(\mathsf{proj}_B(r))$ is a hidden transition and there exists $a \in I_B \cup O_B$ with $\mathbf{L}_0(\mathsf{tr}(r)a) > 0$.

We set $\rho(r)$ to be the unique hidden transition in $\mathbf{G}_{B \uparrow\uparrow C}(\mathsf{last}(r))$ generated by $\rho_B(\mathsf{proj}_B(r))$ (where C stutters). Moreover, $\sigma(r, a) := \bot$ for all $a \in I$.

(3) $\rho_C(\mathsf{proj}_C(r))$ is a hidden transition and there exists $a \in I_C \cup O_C$ with $\mathbf{L}_0(\mathsf{tr}(r)a) > 0$.

Then $\rho(r)$ is set to be the unique hidden transition in $\mathbf{G}_{B \uparrow\uparrow C}(\mathsf{last}(r))$ generated by $\rho_C(\mathsf{proj}_C(r))$ (where B stutters). Moreover, $\sigma(r, a) := \bot$ for all $a \in I$.

(4) All remaining situations.

We consider first $\sigma(r, a)$ for all $a \in I$ and then $\rho(r)$. In both cases, we need to prove additional claims using the fact that none of the previous three clauses apply.

For σ, we show that $I \cap I_B \neq \emptyset$ implies $\sigma_B(\mathsf{proj}_B(r), a) \neq \bot$ for all $a \in I_B$. Assume $I \cap I_B$ is nonempty. Since Clause (2) does not apply, we have three cases:

- $\rho_B(\mathsf{proj}_B(r)) = \bot$. If in addition $\sigma_B(\mathsf{proj}_B(r), a) = \bot$ for all $a \in I_B$, we may apply Proposition 11.3.3 to conclude that $\mathbf{L}_B(\mathsf{tr}(\mathsf{proj}_B(r))a) = 0$ for all $a \in I_B$. By assumption on A and B, this implies: for all $a \in I_A$,

$$\mathbf{L}_A(\mathsf{proj}_A(\mathsf{tr}(r))a) = \mathbf{L}_B(\mathsf{proj}_B(\mathsf{tr}(r))a) = \mathbf{L}_B(\mathsf{tr}(\mathsf{proj}_B(r))a) = 0.$$

On the other hand, since Clause (1) does not apply, we may choose $b \in I \cup O$ with $\mathbf{L}_0(\mathsf{tr}(r)b) > 0$. Therefore, by Item (ii) of Lemma 11.4.4, we know that $I_A \subseteq O_C$, thus $I \cap I_B = I \cap I_A = \emptyset$. This is a contradiction.

- $\rho_B(\mathsf{proj}_B(r))$ is an output transition bundle. Then the desired claim follows from the fact that $\langle \sigma_B, \rho_B \rangle$ is determinate.
- For all $a \in I_B \cup O_B$, $\mathbf{L}_0(\mathsf{tr}(r)a) = 0$. By assumption, $I \cap I_B$ is nonempty, therefore there is $a \in I$ such that $\mathbf{L}_0(\mathsf{tr}(r)a) = 0$. By Lemma 11.3.4, we have $\mathbf{L}_0(\mathsf{tr}(r)a) = 0$ for all $a \in I$. By Lemma 11.4.4, this implies $\mathbf{L}_0(\mathsf{tr}(r)a) = 0$ for all $a \in O$. These two statements contradict the fact that Clause (1) does not apply.

By a similar argument, we have $I \cap I_C \neq \emptyset$ implies $\sigma_C(\mathsf{proj}_C(r), a) \neq \bot$ for all $a \in I_C$. Thus, for every $a \in I$, we may define μ_B to be $\sigma_B(\mathsf{proj}_B(r), a)$ if $a \in I_B$ and $\mathsf{Dirac}(\mathsf{last}(\mathsf{proj}_B(r)))$ otherwise. Similarly for μ_C. Then we set $\sigma(r, a)$ to be $\mu_B \times \mu_C \in \mathbf{R}_{B \Uparrow C}(\mathsf{last}(r))$.

Next we specify $\rho(r)$. If $\mathbf{L}_0(\mathsf{tr}(r)a) = 0$ for all $a \in O$, then $\rho(r) := \bot$. Otherwise, we may choose $a_r \in O$ such that $\mathbf{L}_0(\mathsf{tr}(r)a_r) > 0$. By Proposition 11.4.2, we have

$$\mathbf{L}_A(\mathsf{proj}_A(\mathsf{tr}(r)a_r)) \cdot \mathbf{L}_C(\mathsf{proj}_C(tr(r)a_r)) = \mathbf{L}_0(\mathsf{tr}(r)a_r) > 0.$$

Therefore $\mathbf{L}_C(\mathsf{proj}_C(tr(r)a_r)) > 0$ and

$$\mathbf{L}_B(\mathsf{proj}_B(\mathsf{tr}(r)a_r)) = \mathbf{L}_A(\mathsf{proj}_A(\mathsf{tr}(r)a_r)) > 0.$$

Also, we apply the induction hypothesis for Claim (I) to conclude that

$$\mathbf{Q}_B(\mathsf{proj}_B(r)) \cdot \mathbf{Q}_C(\mathsf{proj}_C(r)) = \mathbf{Q}_{\sigma,\rho}(r) > 0,$$

and hence $\mathbf{Q}_B(\mathsf{proj}_B(r)) > 0$ and $\mathbf{Q}_C(\mathsf{proj}_C(r)) > 0$.

Notice, a_r is locally controlled by either B or C. We consider the case in which $a_r \in O_A = O_B$. The other case (i.e., $a_r \in O_C$) follows similarly, with the roles of B and C interchanged.

Note that $\mathsf{tr}(\mathsf{proj}_B(r)) = \mathsf{proj}_B(\mathsf{tr}(r))$. Since $a_r \in O_B$,

$$\mathbf{L}_B(\mathsf{proj}_B(\mathsf{tr}(r))a_r) = \mathbf{L}_B(\mathsf{proj}_B(\mathsf{tr}(r)a_r)) > 0.$$

Then Lemma 11.3.5 applies to B, $\langle \sigma_B, \rho_B \rangle$, $\mathsf{proj}_B(\mathsf{tr}(r))$ and a_r. Choose $r_0 \in \mathsf{Bran}(B)$ as in Lemma 11.3.5. Since we have shown earlier that $\mathbf{Q}_B(\mathsf{proj}_B(r)) > 0$, we may use the choice of r_0 to conclude that $\mathsf{proj}_B(r) \sqsubseteq r_0$.

Also, since Clause (2) does not apply, either $\rho_B(\mathsf{proj}_B(r))$ is not a hidden transition, or $\mathbf{L}_0(\mathsf{tr}(r)a) = 0$ for all $a \in I_B \cup O_B$. The latter does not hold because by assumption $\mathbf{L}_0(\mathsf{tr}(r)a_r) > 0$ and $a_r \in O_B$. Therefore $\rho_B(\mathsf{proj}_B(r))$ is not a hidden transition. This implies $\mathsf{proj}_B(r) = r_0$.

Next we prove that, if there exists $a_C \in I_C \cap O_B$ and $t \in S_B$ with $\langle a_C, t\rangle \in \mathsf{Supp}(\rho_B(r_0))$, then $\sigma_C(\mathsf{proj}_C(r), a) \neq \bot$ for every $a \in I_C$. Let such a_C and t be given. Then $\mathbf{Q}_B(r_0.\rho_B(r_0).a_C.t) > 0$ and hence

$$\mathbf{L}_A(\mathsf{proj}_A(\mathsf{tr}(r))a_C) = \mathbf{L}_B(\mathsf{proj}_B(\mathsf{tr}(r))a_C) = \mathbf{L}_B(\mathsf{tr}(r_0)a_C) > 0.$$

Now we have: (i) $a_r \in O_A$, (ii) $a_C \in O_A$, (iii) $\mathbf{L}_0(\mathsf{tr}(r)a_r) > 0$, and (iv) $\mathbf{L}_A(\mathsf{proj}_A(\mathsf{tr}(r))a_C) > 0$. Therefore, we may apply Lemma 11.4.3 to conclude that $\mathbf{L}_0(\mathsf{tr}(r)a_C) > 0$.

For contradiction, suppose $\sigma_C(\mathsf{proj}_C(r), a) = \bot$ for some $a \in I_C$. Then, by the definition of determinate I/O schedulers, $\sigma_C(\mathsf{proj}_C(r), a) = \bot$ for all $a \in I_C$. Since Clause (3) does not apply, we have three cases:

- $\rho_C(\mathsf{proj}_C(r)) = \bot$. Since we also have $\sigma_C(\mathsf{proj}_C(r), a) = \bot$ for all $a \in I_C$, we may apply Proposition 11.3.3 to conclude that: for all $a \in I_C$,
$$\mathbf{L}_C(\mathsf{proj}_C(\mathsf{tr}(r))a) = \mathbf{L}_C(\mathsf{tr}(\mathsf{proj}_C(r))a) = 0.$$
On the other hand, since Clause (1) does not apply, we may choose $b \in I \cup O$ with $\mathbf{L}_0(\mathsf{tr}(r)b) > 0$. Therefore, by Item (ii) of Lemma 11.4.4, we know that $\mathbf{L}_0(\mathsf{tr}(r)b') = 0$ for all $b' \in I_C \cup O_C$. This is a contradiction, because by assumption $a_C \in I_C$ and we have shown that $\mathbf{L}_0(\mathsf{tr}(r)a_C) > 0$.
- $\rho_C(\mathsf{proj}_C(r))$ is an output transition bundle. Since $\langle \sigma_C, \rho_C\rangle$ is determinate, it follows that $\sigma_C(\mathsf{proj}_C(r), a) \neq \bot$ for all $a \in I_C$. This is a contradiction.
- For all $a \in I_C \cup O_C$, $\mathbf{L}_0(\mathsf{tr}(r)a) = 0$. This is a contradiction, because by assumption $a_C \in I_C$ and we have shown that $\mathbf{L}_0(\mathsf{tr}(r)a_C) > 0$.

This completes the proof that, if there exists $a_C \in I_C \cap O_B$ and $t \in S_B$ with $\langle a_C, t\rangle \in \mathsf{Supp}(\rho_B(r_0))$, then $\sigma_C(\mathsf{proj}_C(r), a) \neq \bot$ for every $a \in I_C$.

Now we may set $\rho(r)$ to be the unique bundle in $\mathbf{G}_{B\uparrow\uparrow C}(\mathsf{last}(r))$ generated by $\rho_B(r_0)$ and the family

$$\{\sigma_C(\mathsf{proj}_C(r), a) \mid a \in I_C \cap O_B \text{ and } \exists t \in S_B\ \langle a, t\rangle \in \mathsf{Supp}(\rho_B(r_0))\}.$$

This completes the construction of $\langle \sigma, \rho\rangle$.

Inductive Proof of Claim (I)

For each of the four clauses in the construction of $\langle \sigma, \rho\rangle$, we need to prove Claim (I) for every one-step extension r' of r. Let such r' be given and suppose $\mathbf{Q}_{\sigma,\rho}(r') > 0$. Observe that the proof for Clause (1) is trivial.

Proof for Clause (2) By the definition of $\mathbf{Q}_{\sigma,\rho}$ and Axiom (2), r' must be of the form $r.\rho(r).a.s'$, where $\langle a,\, s'\rangle$ is the unique member of $\mathsf{Supp}(\rho(r))$. Observe that

$$\rho(r)(a,s') = \rho_B(\mathsf{proj}_B(r))(a,\mathsf{proj}_B(s'))) = 1,$$

therefore $\mathbf{Q}_{\sigma,\rho}(r') = \mathbf{Q}_{\sigma,\rho}(r)\cdot\rho(r)(a,s') = \mathbf{Q}_{\sigma,\rho}(r)$ and

$$\begin{aligned}\mathbf{Q}_B(\mathsf{proj}_B(r')) &= \mathbf{Q}_B(\mathsf{proj}_B(r).\rho_B(\mathsf{proj}_B(r)).a.\mathsf{proj}_B(s'))\\ &= \mathbf{Q}_B(\mathsf{proj}_B(r))\cdot\rho_B(\mathsf{proj}_B(r))(a,\mathsf{proj}_B(s'))\\ &= \mathbf{Q}_B(\mathsf{proj}_B(r)).\end{aligned}$$

Then, by the induction hypothesis, we have

$$\begin{aligned}\mathbf{Q}_{\sigma,\rho}(r') &= \mathbf{Q}_{\sigma,\rho}(r)\\ &= \mathbf{Q}_B(\mathsf{proj}_B(r))\cdot\mathbf{Q}_C(\mathsf{proj}_C(r))\\ &= \mathbf{Q}_B(\mathsf{proj}_B(r'))\cdot\mathbf{Q}_C(\mathsf{proj}_C(r'))\end{aligned}$$

Proof for Clause (3) The same as for Clause (2), but with B and C interchanged.

Proof for Clause (4) Here we have three cases.

(i) r' is of the form $r.a.\sigma(r,a).s'$ for some $a\in I$ and $s'\in\mathsf{Supp}(\sigma(r,a))$. By Axiom (P1), $\sigma(r,a)(s')=1$, thus $\mathbf{Q}_{\sigma,\rho}(r') = \mathbf{Q}_{\sigma,\rho}(r)$.
If $a\in I_B$, then by the definition of $\sigma(r,a)$ we have:

$$\begin{aligned}\mathsf{proj}_B(r') &= \mathsf{proj}_B(r).a.\mathsf{proj}_B(\sigma(r,a)).\mathsf{proj}_B(s')\\ &= \mathsf{proj}_B(r).a.\sigma_B(\mathsf{proj}_B(r),a).\mathsf{proj}_B(s').\end{aligned}$$

Again by Axiom (P1), we have $\sigma(\mathsf{proj}_B(r),a)(\mathsf{proj}_B(s')) = 1$ and thus $\mathbf{Q}_B(\mathsf{proj}_B(r')) = \mathbf{Q}_B(\mathsf{proj}_B(r))$.
If $a\notin I_B$, then $\mathsf{proj}_B(r')$ coincides with $\mathsf{proj}_B(r)$ and we also have $\mathbf{Q}_B(\mathsf{proj}_B(r')) = \mathbf{Q}_B(\mathsf{proj}_B(r))$.
Applying the same argument to C, we obtain

$$\mathbf{Q}_C(\mathsf{proj}_C(r')) = \mathbf{Q}_C(\mathsf{proj}_C(r)).$$

Therefore, by the induction hypothesis, we have

$$\begin{aligned}\mathbf{Q}_{\sigma,\rho}(r') &= \mathbf{Q}_{\sigma,\rho}(r)\\ &= \mathbf{Q}_B(\mathsf{proj}_B(r))\cdot\mathbf{Q}_C(\mathsf{proj}_C(r))\\ &= \mathbf{Q}_B(\mathsf{proj}_B(r'))\cdot\mathbf{Q}_C(\mathsf{proj}_C(r')).\end{aligned}$$

(ii) r' is of the form $r.\rho(r).a.s'$ for some $a \in O_B \cup H_B$ and $\langle a, s' \rangle \in \mathsf{Supp}(\rho(r))$. By the construction of ρ and the definition of the generative transition structure $\mathbf{G}_{B \Uparrow C}$ (Definition 9.2.2), we know that

$$\rho(r)(a, s') = c \cdot \rho_B(\mathsf{proj}_B(r))(a, \mathsf{proj}_B(s')),$$

where c equals $\sigma_C(\mathsf{proj}_C(r), a)(\mathsf{proj}_C(s'))$ if $a \in I_C$ and 1 otherwise. In either case, we have $\mathbf{Q}_C(\mathsf{proj}_C(r')) = c \cdot \mathbf{Q}_C(\mathsf{proj}_C(r))$.
On the other hand, we may infer the following from the construction of $\rho(r)$:

$$\begin{aligned}\mathsf{proj}_B(r') &= \mathsf{proj}_B(r).\mathsf{proj}_B(\rho(r)).a.\mathsf{proj}_B(s') \\ &= \mathsf{proj}_B(r).\rho_B(\mathsf{proj}_B(r)).a.\mathsf{proj}_B(s').\end{aligned}$$

Therefore,

$$\mathbf{Q}_B(\mathsf{proj}_B(r')) = \mathbf{Q}_B(\mathsf{proj}_B(r)) \cdot \rho_B(\mathsf{proj}_B(r))(a, \mathsf{proj}_B(s')).$$

Now we may apply the induction hypothesis to obtain:

$$\begin{aligned}&\mathbf{Q}_{\sigma,\rho}(r') \\ &= \mathbf{Q}_{\sigma,\rho}(r) \cdot \rho(r)(a, s') \\ &= (\mathbf{Q}_B(\mathsf{proj}_B(r)) \cdot \mathbf{Q}_C(\mathsf{proj}_C(r))) \cdot (c \cdot \rho_B(\mathsf{proj}_B(r))(a, \mathsf{proj}_B(s'))) \\ &= (\mathbf{Q}_B(\mathsf{proj}_B(r)) \cdot \rho_B(\mathsf{proj}_B(r))(a, \mathsf{proj}_B(s'))) \cdot (c \cdot \mathbf{Q}_C(\mathsf{proj}_C(r))) \\ &= \mathbf{Q}_B(\mathsf{proj}_B(r')) \cdot \mathbf{Q}_C(\mathsf{proj}_C(r')).\end{aligned}$$

(iii) r' is of the form $r.\rho(r).a.s'$ for some $a \in O_C \cup H_C$ and $\langle a, s' \rangle \in \mathsf{Supp}(\rho(r))$. The proof is the same as in the previous case, with B and C interchanged.

This concludes the proof of Claim (I).

Wrapping Up

It is routine to check that $\langle \sigma, \rho \rangle$, thus constructed, is a determinate I/O scheduler for $B \Uparrow C$ and is generated by $\langle \sigma_B, \rho_B \rangle$ and $\langle \sigma_C, \rho_C \rangle$. It remains to prove $\mathbf{L}_{\sigma,\rho} = \mathbf{L}_0$.

Lemma 11.4.6. *For every $\alpha \in (I \cup O)^{<\omega}$, $\mathbf{L}_{\sigma,\rho}(\alpha) > 0$ implies $\mathbf{L}_0(\alpha) > 0$.*

Proof. This claim holds trivially for the empty trace, since by definition every likelihood assignment assigns 1 to the empty trace. Consider a trace α' of the form $\alpha a'$ and suppose $\mathbf{L}_{\sigma,\rho}(\alpha') > 0$. By Proposition 11.4.2 and the assumptions of Theorem 11.4.5, we have $\mathbf{L}_A(\mathsf{proj}_A(\alpha')) = \mathbf{L}_B(\mathsf{proj}_B(\alpha')) > 0$ and $\mathbf{L}_C(\mathsf{proj}_C(\alpha')) > 0$.

Moreover, we may choose minimal $r' \in \mathsf{Bran}(B \Uparrow C)$ with $\mathsf{tr}(r') = \alpha'$ and $\mathbf{Q}_{\sigma,\rho}(r') > 0$. Let r denote the unique one-step prefix of r'. Then we have $\mathsf{tr}(r) = \alpha$ and $\mathbf{Q}_{\sigma,\rho}(r) > 0$.

Consider the construction of $\langle \sigma, \rho \rangle$ at r. Clearly, Clause (1) does not apply. For the sake of contradiction, suppose Clause (2) applies. Then $\rho_B(\mathsf{proj}_B(r))$ is a hidden transition and $\rho(r)$ is a hidden transition induced by $\rho_B(\mathsf{proj}_B(r))$. Since a' is not a hidden action, we may conclude that $a' \in I$. However, in Clause (2), $\sigma(r, a)$ is defined to be $\perp$ for every $a \in I$. In particular, $\sigma(r, a') = \perp$, contradicting the fact that $\mathbf{Q}_{\sigma,\rho}(r') > 0$.

A similar argument eliminates Clause (3), therefore we may focus on Clause (4). We have three cases.

- $a' \in I$. Since Clause (1) does not apply, there is $b \in I \cup O$ with $\mathbf{L}_0(\alpha b) > 0$. Applying Lemma 11.3.4, we know that $\mathbf{L}_0(\alpha a) > 0$ for all $a \in I_A$. In particular, $\mathbf{L}_0(\alpha a') > 0$.
- $a' \in O_A = O_B$. In that case, we know r' is of the form $r.\rho(r).a'.s'$ for some s' with $\langle a', s' \rangle \in \mathsf{Supp}(\rho(r))$. Since $a' \in O_B$, $\rho(r)$ is generated by an output bundle of B.

  Since $\rho(r) \neq \perp$, we know that $\mathbf{L}_0(\alpha a) = \mathbf{L}_0(\mathsf{tr}(r)a) > 0$ for some $a \in O$. Choose the same a_r used in the construction of $\rho(r)$ in Clause (4). Then, we may infer that a_r is also in $O_A = O_B$.

  Recall from earlier that $\mathbf{L}_A(\mathsf{proj}_A(\alpha')) > 0$. This implies

  $$\mathbf{L}_A(\mathsf{proj}_A(\alpha)a') = \mathbf{L}_A(\mathsf{proj}_A(\alpha')) > 0,$$

  therefore we may apply Lemma 11.4.3 to α, a_r and a' and conclude that $\mathbf{L}_0(\alpha') = \mathbf{L}_0(\alpha a') > 0$.
- $a' \in O_C$. As in the previous case, we argue that $\rho(r)$ is generated by an output bundle of C. Moreover, the action a_r used in the construction of $\rho(r)$ in Clause (4) is also in O_C.

  Recall from earlier that $\mathbf{L}_C(\mathsf{proj}_C(\alpha')) > 0$. Again by Lemma 11.4.3 we have $\mathbf{L}_0(\alpha') = \mathbf{L}_0(\alpha a') > 0$.

□

Lemma 11.4.7. *For every* $\alpha \in (I \cup O)^{<\omega}$, $\mathbf{L}_0(\alpha) > 0$ *implies* $\mathbf{L}_{\sigma,\rho}(\alpha) > 0$.

Proof. We proceed by induction on the length of α. The base case is trivial, since by definition every likelihood assignment assigns 1 to the empty trace. For the inductive step, consider α' of the form $\alpha a'$.

Suppose $\mathbf{L}_0(\alpha') > 0$. Then $\mathbf{L}_0(\alpha) > 0$. By Proposition 11.4.2 and the assumptions of Theorem 11.4.5, we also have $\mathbf{L}_A(\mathsf{proj}_A(\alpha')) = \mathbf{L}_B(\mathsf{proj}_B(\alpha')) > 0$ and $\mathbf{L}_C(\mathsf{proj}_C(\alpha')) > 0$.

By the induction hypothesis, $\mathbf{L}_{\sigma,\rho}(\alpha) > 0$. Then the set

$$X := \{r \in \mathsf{Bran}(B \uparrow\uparrow C) \mid \mathsf{tr}(r) = \alpha \text{ and } \mathbf{Q}_{\sigma,\rho}(r) > 0\}$$

is nonempty. By Proposition 11.3.3, X is linearly ordered by prefix and has a unique minimal element, call it r_0.

Suppose for the sake of contradiction that X is infinite. Then either $\mathsf{proj}_B(X)$ or proj_C is infinite. We treat only the case in which $\mathsf{proj}_B(X)$ is infinite. The other is completely analogous.

Let r_0' be the shortest extension of r_0 such that $\rho(r_0')$ is a hidden transition generated by some hidden transition in B. Such r_0' exists because $\mathsf{proj}_B(X)$ is nonempty. Then Clause (2) is used in the construction of $\rho(r_0')$ and we may choose $b_0 \in I_B \cup O_B$ with $\mathbf{L}_0(\alpha b_0) = \mathbf{L}_0(\mathsf{tr}(r_0')b_0) > 0$. By Proposition 11.4.2, this implies

$$\mathbf{L}_B(\mathsf{proj}_B(\alpha)b_0) = \mathbf{L}_B(\mathsf{proj}_B(\alpha b_0)) > 0.$$

On the other hand, since $\mathbf{Q}_{\sigma,\rho}(r) > 0$ for every $r \in X$, we may apply Proposition 11.4.2 to conclude that $\mathbf{Q}_B(r_B) > 0$ for every $r_B \in \mathsf{proj}_B(X)$. Moreover, since $\mathsf{tr}(r) = \alpha$ for every $r \in X$, we have $\mathsf{tr}(r_B) = \mathsf{proj}_B(\alpha)$ for every $r_B \in \mathsf{proj}_B(X)$. This implies $\mathsf{proj}_B(X)$ is a subset of the following set:

$$X_B := \{r_B \in \mathsf{Bran}(B) \mid \mathsf{tr}(r_B) = \mathsf{proj}_B(\alpha) \text{ and } \mathbf{Q}_B(r_B) > 0\}.$$

Thus X_B is also infinite.

Since $\langle \sigma_B, \rho_B \rangle$ is determinate, we may conclude that $\sigma_B(r_B, b) = \bot$ for all $b \in I_B$. Hence $\mathbf{L}_B(\mathsf{proj}_B(\alpha)b) = 0$ for every $b \in I_B$. Moreover, by Lemma 11.3.5, $\mathbf{L}_B(\mathsf{proj}_B(\alpha)b) = 0$ for every $b \in O_B$. This contradicts our earlier claim that $\mathbf{L}_B(\mathsf{proj}_B(\alpha)b_0) > 0$. Therefore X must be finite.

Let r' denote the maximal element of X and consider the construction of $\langle \sigma, \rho \rangle$ at r'. Clause (1) does not apply since $\mathbf{Q}_{\sigma,\rho}(r') > 0$ and $\mathbf{L}_0(\alpha a') > 0$. Clauses (2) and (3) do not apply because of maximality of r'. Thus, we may focus on Clause (4). We have three cases.

- $a' \in I$. In Clause (4), $\sigma(r', a) \neq \bot$ for all $a \in I$. Let t be the unique state in $\mathsf{Supp}(\sigma(r', a'))$. Then $\mathbf{Q}_{\sigma,\rho}(r'.a'.\sigma(r', a').t) > 0$ and hence $\mathbf{L}_{\sigma,\rho}(\alpha a') > 0$.
- $a' \in O_B$. Then $\mathbf{L}_B(\mathsf{proj}_B(\alpha)a') = \mathbf{L}_B(\mathsf{proj}_B(\alpha')) > 0$.

  By Lemma 11.4.3 and the fact that $\mathbf{L}_0(\alpha a') > 0$, we may infer that $\mathbf{L}_0(\alpha c) = 0$ for all $c \in O_C$. Therefore, by the construction in Clause (4), $\rho(r')$ is an output bundle generated by some output bundle in B. Thus, there is some $b \in O_B$ such that $\mathbf{L}_{\sigma,\rho}(\alpha b) > 0$. Now we apply Lemma 11.4.3 again to conclude that $\mathbf{L}_{\sigma,\rho}(\alpha a') > 0$.
- $a' \in O_C$. The same as in the previous case, but with B and C interchanged.

□

The following is a corollary of Lemma 11.4.6, Lemma 11.4.7 and Proposition 11.4.2.

Corollary 11.4.8. *For every* $\alpha \in (I \cup O)^{<\omega}$, $\mathbf{L}_{\sigma,\rho}(\alpha) = \mathbf{L}_0(\alpha)$.

Proof. By virtue of Lemmas 11.4.6 and 11.4.7, we may assume both $\mathbf{L}_{\sigma,\rho}(\alpha) > 0$ and $\mathbf{L}_0(\alpha) > 0$. Then by Proposition 11.4.2 and the assumptions of Theorem 11.4.5, we have

$$\mathbf{L}_{\sigma,\rho}(\alpha) = \mathbf{L}_B(\mathsf{proj}_B(\alpha)) \cdot \mathbf{L}_C(\mathsf{proj}_C(\alpha)) = \mathbf{L}_A(\mathsf{proj}_A(\alpha)) \cdot \mathbf{L}_C(\mathsf{proj}_C(\alpha)) = \mathbf{L}_0(\alpha).$$

□

We have now completed the proof of Theorem 11.4.5.

11.5 Conclusions

In this chapter, we presented a probabilistic modeling framework, along with a compositional trace-style semantics. The defining axioms of this new framework is motivated by the notion of local-oblivious scheduling, which prevents the adversary from learning local coin tosses "too early".

Our notions of external behavior and parallel composition are similar to those introduced in Chapter 10. In particular, we follow the same motto of "schedule-and-compose", where local nondeterministic choices must be fully resolved before a global parallel composition takes place. This key idea ensures that our trace-style semantics is compositional.

Many interesting ideas remain to be worked out. For instance, in the formal methods literature, it is typical that adversaries/policies/schedulers depend entirely on the "past" (e.g., a state sequence and/or actions that have already been executed). In the literature of distributed algorithms, adversaries may be able to detect differences in enabled operations, that is, those that have not yet been completed (cf. [Cha96, AB04]). This is used to model situations such as slowdown in hardware access due to contention. It would be interesting to develop more formally this notion of "future-dependent" adversaries.

Also, we have experienced quite some difficulty in finding a consistent way to assign probabilities to various interleavings in a parallel composition. In our view, this is due largely to the fact that interleaving is inherently timing related, and yet our basic framework is untimed, abstracting away from properties such as "some internal computations require more time than others." This may occur, for example, when division operations are performed on randomly chosen values, so that internal random choices can in fact affect the outcome of interleaving.

Therefore, we believe that our formal modeling will benefit from a move to a timed setting, where relative timing can be treated explicitly.

Finally, we mention two possible extensions to the technical work presented here. First, we would like to develop an algorithm that translate a given PIOA specification into one that satisfies our three partial-information axioms. This is because we recognize the fact that it may be less natural to write a specification while conforming to those axioms. We expect the translation algorithm to be similar to algorithms that compute the belief-state MDP from a given POMDP [KLA98]. It is likely that we need to impose certain restrictions on the original PIOA spec, e.g., an upperbound on the number of consecutive hidden transitions.

Moreover, we would like to define a notion of probabilistic simulation for our new framework, as a sound method for proving behavioral inclusion. Some preliminary results (not yet published) have already been obtained for the simpler case of closed PIOAs, that is, those without input actions.

Part III

A Randomized Consensus Algorithm

12

Randomized Wait-Free Consensus

We present a randomized algorithm for asynchronous wait-free consensus using multi-writer multi-reader shared registers. This algorithm is based on earlier work by Chor, Israeli and Li (CIL) and is correct under the assumption that processes can perform a random choice and a write operation in one atomic step. The expected total work for our algorithm is shown to be $O(N \log(\log N))$, compared with $O(N^2)$ for the CIL algorithm, and $O(N \log N)$ for the best known weak adversary algorithm. We also model check instances of our algorithm using the probabilistic model checking tool PRISM.

12.1 Introduction

Distributed consensus refers to a class of problems in which a set of parallel processes exchange messages in order to agree on a common preference. Initially, each process is given an input value from a fixed, finite domain and, at the end of the algorithm, each non-faulty process outputs a decision value. Correctness requirements are typically formulated as follows.

- *Validity*: the output of any non-faulty process must have been the input of some process.
- *Agreement*: all non-faulty processes decide on the same value.
- *Termination*: every non-faulty process decides after a finite number of steps.

As shown in [FLP85], there exists no deterministic algorithm that solves distributed consensus in a setting of asynchronous communication with undetected process failure. Nonetheless, many efficient solutions exist under stronger assumptions (e.g., partial synchrony [DLS88] and failure detection [ACT00]) or weaker correctness requirements (e.g., probabilistic termination [CIL87]).

Our algorithm falls into the category of *randomized consensus algorithms*, where processes may use coin tosses to determine their course of actions. In this setting, termination is weakened to a probabilistic statement: the set of all

non-terminating executions has probability 0. We refer to [Asp03] for a comprehensive overview on randomized consensus.

The first randomized consensus algorithm was proposed by Ben-Or [BO83] and was designed for a message-passing system. A few years later, Chor, Israeli and Li published the first randomized consensus algorithm using shared [CIL87, CIL94] using shared memory . It satisfies the following termination condition.

- *Probabilistic wait-free termination*: with probability 1, each non-faulty process decides after a finite number of steps.

We adopt the same requirement. In fact, the logical structure of our algorithm closely resembles that in [CIL94], while we borrow ideas from [Cha96] to reduce the amount of shared and local data. We shall refer to [CIL94] as the original CIL algorithm and our own as the modified CIL algorithm.

Adversary Models and Work Bounds

To prove probabilistic termination, we must reason about probability distributions on the set of executions. This is done by specifying the so-called *adversaries*, which are fictitious entities designed to model scheduling uncertainties in a distributed environment. Mathematically, an adversary is a function mapping each finite execution to an available next operation. Such a function resolves all non-deterministic choices among parallel processes, thereby inducing a probability distribution on the set of executions. One can then ask if probabilistic termination is satisfied according to this distribution. By quantifying over all possible adversaries (of a certain strength), we obtain worst-case guarantees similar to those in a non-probabilistic setting.

The strength of an adversary varies according to the amount of information it can extract from a finite history. The *strong* adversaries have access to complete history of all processes and shared registers. Some weaker forms, such as *write-oblivious* and *value-oblivious*, delay the adversary's knowledge of outcomes of internal coin tosses. The *oblivious* adversaries are the weakest, unable to observe any random outcomes and their subsequent effects on system dynamics.

Clearly, a stronger adversary model permits more possibilities and therefore renders consensus more difficult. Consensus against strong adversaries is shown to be $\Omega(N^2/\log^2 N)$ in expected total work, where N is the number of processes participating in the algorithm [Asp98]. The best known algorithms achieve expected $O(N^2 \log N)$ total work [BR91] and $O(N \log^2 N)$ per process [AW96]. Against write-oblivious adversaries, one can achieve expected $O(\log N)$ per process work and $O(N \log N)$ total work [Aum97]. Against value-oblivious adversaries, the fastest algorithm is $O(N \log N\, e^{\sqrt{\log N}})$ in a single-writer single-reader (SWSR) setting [AKL99][1].

[1] This is faster than other value-oblivious algorithms because SWSR is a weak primitive. More discussion can be found in Section 12.7.

Our adversary model takes the form of an atomicity assumption: processes can perform a random choice and a write operation in one atomic step. In particular, the process increments its round number if and only if the coin lands heads; then immediately it writes 1 to the memory location $\mathsf{mem}(v, r)$, where v is the current preference and r is the round number *after* the coin toss. This amounts to saying that the adversary cannot distinguish between the two locations $\mathsf{mem}(v, r)$ and $\mathsf{mem}(v, r + 1)$.

The original CIL algorithm relies on a similar assumption[2] and achieves expected $O(N^2)$ total work [CIL94]. In the present paper, we replace the single-writer multiple-reader (SWMR) registers of [CIL94] with multi-writer multi-reader (MWMR) registers, thereby reducing the expected total work bound to $O(N \log(\log N))$.

Since our adversaries are value-sensitive, every non-faulty process must perform at least one read operation, otherwise we can easily construct an execution that violates the agreement property. Therefore, expected total work in this model is $\Omega(N)$, which is almost matched by our upper bound of $O(N \log(\log N))$.

We have adopted the worst case expected total work as our complexity measure, mainly because it is more natural to reason about the collective effect of all processes on the shared memory. In fact, per process work in our case is comparable to total work: if all but one process suffer crash failures, the lone survivor carries the total work burden and performs expected $\Omega(N)$ tosses in order to pull far enough ahead for termination. In this sense, our algorithm is less efficient than [Cha96, Aum97], where polylogarithmic upper bounds are given for per process work.

Probabilistic Model Checking

We model check instances of our algorithm using PRISM, which can check PCTL (*Probabilistic Computation Tree Logic*) formulas against an MDP (*Markov Decision Process*) [PRI, BK98]. This tool has been applied to many randomized algorithms, including the consensus algorithm of Aspnes and Herlihy [AH90, KNS01] and Byzantine agreement [KN02].

Consensus algorithms are often hard to model check, because the state space grows exponentially with the number of participating processes. In [KNS01], PRISM is applied only to a shared-coin subroutine, while full correctness relies on verification using Cadence SMV, as well as higher level manual proofs. Unfortunately, the structure of our algorithm does not provide such convenient isolation of probabilistic reasoning. Nevertheless, we are able to build models of binary consensus with up to 4 processes and verify relevant properties.

[2] Since [CIL94] uses SWMR memory registers, while we use MWMR registers, the same atomicity assumption has different meanings in the two settings. The version in [CIL94] says the adversary cannot distinguish between the values r and $r + 1$ as they are written to the same memory location.

In Section 12.6, we briefly describe these models and give a summary of PRISM results. In Section 12.7, we discuss our learning experience with PRISM and some prospects in improving feasibility of model checking.

Overview

Section 12.2 describes in greater detail our computational setting and assumptions. Section 12.3 presents the algorithm and correctness proofs are given in Sections 12.4 and 12.5. Section 12.6 is devoted to model checking and Section 12.7 contains closing discussions.

12.2 System Model

We consider a system of N processes interacting asynchronously via shared memory objects. Each process P_i is given as input an initial preference p_i^0, which belongs to a fixed, finite domain. Without loss of generality, this preference domain is assumed to be $\mathbb{Z}_K$ for some natural number constant $K \geq 2$. As a convention, we write $\mathbb{Z}_K$ for $\{0, \ldots, K-1\}$ and $\mathbb{Z}_K^+$ for $\{1, \ldots, K-1\}$.

We take a state-based view of our system. The *local state* of a process is determined by a valuation of all of its local variables, plus a program counter indicating the next line of code to be executed. In fact, it is trivial to include the program counter as an explicit variable, so that local state is fully determined by valuation of local variables. This is done in our PRISM models.

The *global state* is then determined by local states of all N processes, together with contents of shared MWMR atomic registers. These registers are by definition *linearizable* [HW90], meaning that each memory operation can be viewed as taking place at a particular instant in time, as opposed to an interval between *invocation* and *response*. Under this assumption, each read access returns the value written by the last write access in the linearized execution history.

A process executes a possibly infinite sequence of discrete steps, each consisting of a change in local state and/or a memory operation. It may also exhibit a limited form of non-deterministic behavior: crashing at any point of its execution. A crashed process may never recover and re-enter the algorithm.

An execution of the entire system is obtained by interleaving executions of individual processes, where scheduling among processes is determined by an adversary that satisfies the atomicity assumption stated in Section 12.1. That is, if a process is scheduled to toss a coin, it must be allowed to write to the memory before another process is given a turn. The worst-case complexity is measured in terms of the expected number of read and write operations taken by all processes, quantifying over all admissible adversaries.

12.3 Modified CIL Algorithm

As in many other consensus algorithms [BO83, CIL94, AH90, Cha96], we make use of a *round* structure. During each round, a process goes through a possibly infinite sequence of *phases*, each of which is a complete pass through the main **while**-loop.

In original CIL, the shared memory is configured into an array of N many SWMR registers, one for every process. Each $\mathsf{register}_i$ contains two pieces of information: round number r_i and preference value p_i. At the beginning of each phase, process P_i copies the contents of all $\mathsf{register}_j$ $(i \neq j)$ and stores them locally. These entries are then examined to decide the next action of P_i: output a decision value and terminate, toss a coin to advance to the next round, or jump to a higher round.

The initial copying of each phase is the main source of inefficiency in original CIL, because the copied data contain more information than necessary for decision making. For example, P_i need not know exactly which P_j is in a higher round, as long as it knows *some* P_j is. This observation is precisely the motivation of our move from SWMR memory to MWMR memory. Instead of a *race among processes*, we envision a *race among preference values*, so that processes participate anonymously and the number of read operations in the main loop is reduced from $O(N)$ to $O(1)$. Moreover, consensus is achieved with high probability using only $O(\log N)$ registers containing one bit each.

Our MWMR shared memory is configured into K arrays of bits, each of length $R+2$, where $R := 2\lceil \log N \rceil$. (Recall that K is the size of the preference domain and is a constant, while N is the number of participating processes.) In other words, we have $\mathsf{mem} : \mathbb{Z}_{R+2} \times \mathbb{Z}_K \to \{0, 1\}$, and the entries can be interpreted as follows.

- For all $r \in Z^+_{R+1}$ and $v \in \mathbb{Z}_K$, $\mathsf{mem}(r, v) = 1$ if and only if value v has reached round r (i.e., some process holds/held preference v while in round r). These entries are initialized to 0.
- We assume every value v participates in the race from round 0, therefore $\mathsf{mem}(0, v)$ is initialized to 1. This prevents a process from deciding (erroneously) in round 1 before all processes "wake up" and join the protocol[3].
- Round-$(R+1)$ entries are initialized to 0 and are used for marking decision values. That is, if a process decides on value v, it writes 1 to $\mathsf{mem}(R+1, v)$.

Each process P_i maintains a current preference p_i and a round number r_i. Intuitively, P_i "believes" that p_i is a leading value and r_i is the highest round reached by p_i. If P_i detects any value v in a round higher than r_i, it updates its "belief" by running a subroutine Jump. In this way, lagging values are quickly

[3] As noted in [CH05], original CIL contains this initialization error.

abandoned by active processes and are eventually *eliminated* from the race. (This notion is made precise in Definition 12.4.1 in Section 12.4.) Therefore the number of contending preference values never increases and the algorithm terminates when that number decreases to 1. If P_i sees p_i at least two rounds ahead in the race, the algorithm guarantees that every other contending value has been eliminated, therefore P_i can safely terminate with p_i.

Notice, biased coin tosses are used to break ties in the lead pack, so that with probability 1 the number of contending preferences eventually reaches 1. This technique is quite different from the more common approach of shared coin subroutines, in which processes cast randomly generated votes to obtain a weak shared-coin [AH90, BR91].

Although every non-faulty process is guaranteed (with probability 1) to terminate after a finite number of steps, the round in which it terminates can become arbitrarily high. This requires an unbounded number of registers and is infeasible. Therefore we stop our algorithm when it reaches a certain round without successful termination, in which case we switch to a slower algorithm that requires bounded memory. We call this the *exit* algorithm. For convenience, the original CIL algorithm is chosen for this purpose[4]. We will show that any exit algorithm is invoked with probability at most $\frac{1}{N}$, therefore the higher cost of original CIL does not affect overall expected complexity.

Figure 12.1(a) contains the pseudocode for process P_i. The numbered lines can be described informally as follows.

(1) Check if some process has decided.

(2) If so, decide for the same value.

(3) Check if a value other than p_i has reached round $r_i - 1$.

(4) If not, write 1 to $\mathsf{mem}(R+1, p_i)$ and terminate with output p_i.

(5) Otherwise, if round R is reached, run the original CIL algorithm.

(6) Otherwise, check if some value has reached round $r_i + 1$.

(7) If not, advance p_i to the next round with probability $\frac{1}{2N}$.

(8) Otherwise, run subroutine Jump to find a leading value.

Note that the atomicity assumption discussed in Section 12.1 applies at Line (7). This prevents the adversary from selectively delaying write operations of processes who are ready to advance its preference to the next round.

Figures 12.1(b) and 12.1(c) contain the subroutines ReadMem and Jump, respectively. The former is used to read from the shared memory, while the later

[4]Technically, original CIL requires registers with unbounded size. However, according to [CIL94], the probability of non-termination is already extremely small (2^{-56}) when each register is 128 bits.

```
ModifiedCIL(i, p_i^0)
local variables
  // round number
  r_i ∈ Z_{R+2},
  // preference
  p_i ∈ Z_K,
  // decision value
  d_i ∈ Z_{K+1},
  // values read from memory
  ahead_i, behind_i ∈ Z_{K+1}
begin
  p_i := p_i^0;  r_i := 0;
  while r_i ≤ R do
(1) d_i := ReadMem(R + 1, K);
(2) if d_i ≠ K then return d_i;
    if r_i > 0 then {
(3)   behind_i := ReadMem(r_i − 1, p_i);
(4)   if behind_i = K then {
        mem(R + 1, p_i) := 1;
        return p_i
        }
(5)   elseif r_i = R then return
        OriginalCIL(i, p_i)
      }
(6) ahead_i := ReadMem(r_i + 1, K);
    if ahead_i = K then {
(7)   with probability 1/(2N) do
        r_i := r_i + 1;
      mem(r_i, p_i) := 1
      }
(8) else ⟨r_i, p_i⟩ := Jump(r_i + 1, ahead_i)
  od
end
```

(a) Main Algorithm.

```
ReadMem(r, p)
local variables
  // counter
  k ∈ Z_K,
  // preference value found
  v ∈ Z_{K+1},
begin
  k := 0;  v := K;
  while k < K and v = K do
    if mem(r, k) = 1 and k ≠ p then
      v := k;
    k := k + 1
  od
  return v
end
```

(b) Subroutine ReadMem.

```
Jump(r, p)
local variables
  // confirmed round and preference
  r′ ∈ Z_{R+1}, p′ ∈ Z_K,
  // current round and preference
  l ∈ Z^+_{R+1}, u ∈ Z_{K+1},
  // counter
  c ∈ Z_{R+1},
begin
  if r ≥ R then return ⟨r, p⟩;
  r′ := r;  p′ := p;  c := ⌈log(R − r)⌉;
  while c > 0 do
    l := r′ + 2^{c−1};
    if l ≤ R then {
      u := ReadMem(l, K);
      if u ≠ K then {
        r′ := l;  p′ := u
        }
      }
    c := c − 1
  od
  return ⟨r′, p′⟩
end
```

(c) Subroutine Jump.

Figure 12.1: Modified CIL Algorithm

is used to find a leading value. When called with parameters r and p, ReadMem scans one-by-one the r-th entry of every bit vector, except for the p-th. In other words, we would like to know if any process has reached round r with preference other than p. It returns the first k such that both $k \neq p$ and, at the time of read access, $\mathsf{mem}(r, k) = 1$. If no such k is encountered, ReadMem returns K.

In every pass through the **while**-loop of Figure 12.1(a), ReadMem is called with at most three round numbers: $R+1$, $r_i - 1$, and $r_i + 1$. This does not reveal the highest round ever reached by any value. Therefore, a separate subroutine Jump is run when the process sees itself behind. This is a key difference between our algorithm and original CIL: in exchange for fewer read operations in the main loop, more work is needed for slower processes to catch up.

The subroutine Jump can be implemented in various ways. A simple solution is to increment r_i by 1 and then call ReadMem once to find the least k such that $\mathsf{mem}(r_i, k) = 1$. This requires a constant amount of work per invocation of Jump and is implemented in our PRISM models. However, a process lagging way behind may need to step through the main loop as many as R times in order to catch up. Hence we opt for a faster implementation, which is essentially a binary search on mem. This involves $O(\log(\log N))$ operations per invocation of Jump, but a process can correctly locate a leading value in one complete phase (provided no further progress is made in the mean time).

12.4 Validity and Agreement

In this section, we treat all coin tosses as non-deterministic choices. Let s_0 denote the initial state of our system, where all N processes as well as the shared memory have been properly initialized. A *path* of the system is a finite sequence of states $s_0 s_1 \dots s_m$ where, for all $j \in \mathbb{Z}_m$, s_{j+1} can be obtained from s_j by allowing exactly one non-faulty process to execute its next instruction. A state s is *reachable* if there is a path ending in s. Finally, a value $k \in \mathbb{Z}_K$ is said to be *valid* if there is $i \in \mathbb{Z}_N$ such that k equals the input p_i^0 to process P_i.

We use record notation to indicate valuation of variables. For example, $s.r_i$ denotes the round number of P_i in state s. If P_i is running a subroutine (e.g., ReadMem), we add subscript i to variables of that subroutine (e.g., $s.k_i$ and $s.v_i$). This should not cause any confusion because each process runs at most one instance of any subroutine at any given point of the execution.

First we state some properties about mem and subroutines ReadMem and Jump. Lemma 12.4.1 says that an entry in mem never changes from 1 to 0, and Lemma 12.4.2 says that the return value of ReadMem is correct (although it may be out-of-date). Similarly, Lemma 12.4.3 states the correctness of Jump.

Lemma 12.4.1. *Let $r \in \mathbb{Z}_{R+2}$, $v \in \mathbb{Z}_K$ and a path $s_0 \dots s_m$ be given. Suppose there is $j \in \mathbb{Z}_{m+1}$ with $s_j.\mathsf{mem}(r, v) = 1$. Then $s_{j'}.\mathsf{mem}(r, v) = 1$ for all $j \leq j' \leq m$.*

Proof. A process writes to the shared memory only if it executes Lines (4) or (7) in Figure 12.1(a). In either case, the value 1 is written. Therefore, once $\mathsf{mem}(r, v)$ becomes 1, it retains that value in the rest of the path. □

Lemma 12.4.2. *Let $r \in \mathbb{Z}_{R+2}$, $p, v \in \mathbb{Z}_{K+1}$ and a path $s_0 \dots s_m$ be given. If the last step is $\mathsf{ReadMem}(r, p)$ returning $v \neq K$, then $s_m.\mathsf{mem}(r, v) = 1$.*

Proof. The return value v of ReadMem is set to a value other than K only if the **if-then** clause is executed. Let s_j $(0 \leq j \leq m)$ be the state from which this instance of ReadMem reads from $\mathsf{mem}(r, v)$. Clearly, $s_j.\mathsf{mem}(r, v) = 1$. By Lemma 12.4.1, this also holds in s_m. □

Lemma 12.4.3. *Let $r, r'' \in \mathbb{Z}_{R+1}$, $p, p'' \in \mathbb{Z}_K$ and a path $s_0 \dots s_m$ be given. Suppose the last step is $\mathsf{Jump}(r, p)$ returning $\langle r'', p'' \rangle$. If $\mathsf{mem}(r, p) = 1$ when $\mathsf{Jump}(r, p)$ is called, then $s_m.\mathsf{mem}(r'', p'') = 1$.*

Proof. We prove that $\mathsf{mem}(r', p') = 1$ is an invariant of the **while**-loop in Jump. By assumption, the claim holds for initial values $r' = r$ and $p' = p$. Noticed that $\langle r', p' \rangle$ is updated only if the **if-then** clause is executed, in which case $v \neq K$. Since v is the return value of $\mathsf{ReadMem}(l, K)$, we have by Lemma 12.4.2 that $\mathsf{mem}(l, v) = 1$, hence $\mathsf{mem}(r', p') = 1$ still holds after the update. Let s_j be the state immediately after the last update of $\langle r', p' \rangle$. Then we know $s_j.\mathsf{mem}(r'', p'') = 1$. By Lemma 12.4.1, this also holds in s_m. □

Lemma 12.4.4 below states that mem correctly reflects the preference history of participating processes. Validity is then proven to be an invariant (Theorem 12.4.5).

Lemma 12.4.4. *Let a path $s_0 \dots s_m$ be given.*

(i) For all $i \in \mathbb{Z}_N$, $s_m.r_i \leq R$ implies $s_m.\mathsf{mem}(s_m.r_i, s_m.p_i) = 1$.

(ii) For all $r \in \mathbb{Z}^+_{R+2}$ and $v \in \mathbb{Z}_K$, $s_m.\mathsf{mem}(r, v) = 1$ implies there exist $i \in \mathbb{Z}_N$ and $j \in \mathbb{Z}_{m+1}$ such that $s_j.p_i = v$.

Proof. We proceed by induction on the length of paths. For the initial state s_0, recall that round-0 entries are initialized to 1 and all other entries 0, therefore the two claims hold trivially.

Now we consider a path $s_0 \dots s_m s_{m+1}$. Suppose the last step is taken by process P_i. Let r denote $s_m.r_i$ and v denote $s_m.p_i$. Notice that only Lines (4), (7) and (8) in Figure 12.1(a) update variables r_i, p_i and mem.

– Line (4). By Lemma 12.4.1, Item (i) is trivial because r_i is not updated. Item (ii) holds because the only entry of interest is $\mathsf{mem}(R + 1, s_m.p_i)$.

– Line (7). If the coin toss fails, there is no state change other than the program counter of P_i. Suppose the coin toss succeeds. For Item (i), simply note that

$$s_{m+1}.\mathsf{mem}(s_{m+1}.r_i, s_{m+1}.p_i) = s_{m+1}.\mathsf{mem}(r+1, v) = 1$$

because the two updates in Line (7) are assumed to be simultaneous. On the other hand, Item (ii) also holds because $s_{m+1}.p_i = v$.

– Line (8). Item (i) follows from Lemma 12.4.3. Item (ii) follows from the induction hypothesis.

□

Theorem 12.4.5. *The following claims hold in every reachable state s.*

(i) For every $i \in \mathbb{Z}_N$, $s.p_i$ is valid.

(ii) For every $r \in \mathbb{Z}^+_{R+2}$ and $v \in \mathbb{Z}_K$, $s.\mathsf{mem}(r, v) = 1$ implies v is valid.

(iii) For every $i \in \mathbb{Z}_N$, if $s.d_i \neq K$ then $s.d_i$ is valid. Similarly for $s.\mathsf{ahead}_i$ and $s.\mathsf{behind}_i$.

Proof. We prove these claims simultaneously using induction on the length of paths. First consider the initial state s_0. For Item (i), every $s_0.p_i$ is valid because it is set to the input value p_i^0. Item (ii) holds trivially because all but round-0 entries are initialized to 0. Item (iii) is also trivial because all relevant variables are initialized to K.

Now we consider a path $s_0 \ldots s_m s_{m+1}$. Suppose the last is taken by process P_i. We examine Figure 12.1(a) line by line for all possible actions of P_i.

– Line (1). In this case, one update is possible: d_i is set to the return value v of $\mathsf{ReadMem}(R+1, K)$. Suppose v is not K. Then we can apply Lemma 12.4.2 to conclude that $s_{m+1}.\mathsf{mem}(R+1, v) = 1$. Since mem is not updated in the last step, this also holds in s_m. Applying the induction hypothesis, we conclude that $s_{m+1}.d_i = v$ is valid.

– Line (2). In this case, P_i terminates by returning the value $s_m.d_i$ and there are no variable updates. We simply apply the induction hypothesis.

– Lines (3) and (6). Similar to Line (2).

– Line (4). In this case, $\mathsf{mem}(R+1, s_m.p_i)$ is set to 1. By the induction hypothesis, $s_m.p_i$ is valid. Therefore Items (ii) hold in s_{m+1}. Item (iii) follows from the inductive hypothesis.

– Line (7). Similar to Line (4).

– Line (8). $\langle r_i, p_i \rangle$ is set to the return values of Jump. Notice that this update is executed only if $s_m.\mathsf{ahead}_i \neq K$. Therefore, we can conclude that in Line (6) ReadMem returned a value v other than K. Moreover, notice that from then on r_i and ahead_i have not be updated. Therefore, by Lemmas 12.4.1 and 12.4.2, we know that $\mathsf{mem}(s_m.r_i + 1, s_m.\mathsf{ahead}_i) = 1$ at the time Jump is called. Applying Lemma 12.4.3, we have

$$s_{m+1}.\mathsf{mem}(s_{m+1}.r_i, s_{m+1}.p_i) = 1.$$

Since mem is not updated in the last step, we have

$$s_m.\mathsf{mem}(s_{m+1}.r_i, s_{m+1}.p_i) = 1.$$

Applying the induction hypothesis, we conclude that $s_{m+1}.p_i$ is valid.

□

Corollary 12.4.6. *The modified CIL algorithm in Figure 12.1 is valid, assuming the exit algorithm (in this case, the original CIL algorithm) is valid.*

Next we prove agreement. A key ingredient is a predicate Φ on global states.

Definition 12.4.1. Let $v, v' \in \mathbb{Z}_K$ and $r \in \mathbb{Z}^+_{R+1}$ be given. We say that v *eliminates* v' *in round* r *in global state* s (denoted $s \models \Phi(v, v', r)$) just in case $s.\mathsf{mem}(r, v) = 1$ and $s.\mathsf{mem}(r-1, v') = 0$.

We state a string of lemmas leading to the claim that no two processes terminating by Line (4) do so with conflicting decision values (Lemma 12.4.10). First, we have the fact that, if an entry $\mathsf{mem}(r, v)$ is marked 1, then every entry $\mathsf{mem}(r', v)$ with $r' \leq r$ is also marked 1. This is Lemma 12.4.7 below.

Lemma 12.4.7. *Let s be a reachable state. For all $r \in \mathbb{Z}_{R+1}$ and $v \in \mathbb{Z}_K$, if $s.\mathsf{mem}(r, v) = 1$ then $s.\mathsf{mem}(r', v) = 1$ for all $r' \leq r$.*

Proof. We proceed by induction on the length of paths. Clearly this holds at the initial state s_0. Consider a path of the form $s_0 \ldots s_{m+1}$ and assume the property holds for s_m. The only case of interest is when $\mathsf{mem}(r, v)$ changes from 0 to 1 as the result of some process P_i executes Line (7) from s_m. In that case, we have $s_m.r_i = r - 1$ and $s_m.p_i = v$. By Lemma 12.4.4, we may infer that $s_m.\mathsf{mem}(r-1, v) = 1$. By the induction hypothesis, $s_m.\mathsf{mem}(r', v) = 1$ for all $r' \leq r - 1$. Using Lemma 12.4.1, we conclude that $s_{m+1}.\mathsf{mem}(r', v) = 1$ for all $r' \leq r$. □

Lemma 12.4.8 says, if a value v' is eliminated by another value v in round r, then no process subsequently reaches round r with preference v'. Intuitively, this holds because $\mathsf{mem}(r, v')$ changes from 0 to 1 only when some process P_i preferring v' executes Line (7). Using the definition of Φ and Lemma 12.4.7, we argue that implies that such a process does not reach Line (7).

Lemma 12.4.8. *Let $v, v' \in \mathbb{Z}_K$ and $r \in \mathbb{Z}^+_{R+1}$ be given. Consider a path $s_0 \dots s_m$ such that $s_j \models \Phi(v, v', r)$ for some $j \in \mathbb{Z}_{m+1}$. Then, for all $j' \in \{j, \dots, m\}$, $s_{j'}.\mathsf{mem}(r, v') = 0$.*

Proof. By the definition of Φ, we have $s_j.\mathsf{mem}(r-1, v') = 0$ and $s_j.\mathsf{mem}(r, v) = 1$. Using Lemma 12.4.7, we infer that $s_j.\mathsf{mem}(r, v') = 0$. For contradiction, suppose that the claim doesn't hold. We focus on the least $j' \geq j$ with $s_{j'}.\mathsf{mem}(r, v') = 1$. Then it must be the case that $s_{j'-1}.\mathsf{mem}(r, v') = 0$ and some process P_i executes Line (7) from $s_{j'-1}$. Moreover, $s_{j'-1}.r_i = r - 1$ and $s_{j'-1}.p_i = v'$.

On the other hand, using Lemma 12.4.4 and the fact that $s_j.\mathsf{mem}(r-1, v') = 0$, we know either $s_j.r_i < r - 1$ or $s_j.p_i \neq v'$. Therefore, P_i must have entered the current phase *after* s_j. Since $\mathsf{mem}(r, v)$ is 1 in every state following s_j, the invocation of ReadMem in Line (6) of the current phase of P_i must have returned a value other than K. This contradicts the claim that P_i executes Line (7) in the current phase. □

Finally, if a process P_i terminates by Line (4) with value v in round r, then every other v' must have been eliminated by v in round r at some earlier state. This is proven in Lemma 12.4.9.

Lemma 12.4.9. *Consider a path $s_0 \dots s_{m+1}$. Suppose that in the last step some process P_i terminates by executing Line (4). Let r denote $s_m.r_i$ and v denote $s_m.p_i$. For every $v' \neq v$, there is $j' \in \mathbb{Z}_{m+1}$ such that $s_{j'} \models \Phi(v, v', r)$.*

Proof. Let $v' \neq v$ be given and let s_j denote the first state in which P_i enters the current phase. Thus $s_j.r_i = r$ and $s_j.p_i = v$. By Lemma 12.4.4 and Lemma 12.4.1, we have $s_{j''}.\mathsf{mem}(r, v) = 1$ for all $j'' \in \{j, \dots, m\}$.

Since Line (4) is executed, r must be non-zero and the invocation of ReadMem in Line (3) must have returned K. Let $s_{j'}$ be the state from which ReadMem reads from $\mathsf{mem}(r-1, v')$. Since the return value of ReadMem is K, we may infer that $s_{j'}.\mathsf{mem}(r-1, v') = 0$. Moreover, we have $j' > j$ and hence $s_{j'}.\mathsf{mem}(r, v) = 1$. Therefore $s_{j'} \models \Phi(v, v', r)$. □

We are now ready for Lemma 12.4.10. The basic idea is, if two distinct decision values exist, then they must have eliminated each other. By Lemma 12.4.8, this is a contradiction.

Lemma 12.4.10. *Let a path $s_0 \dots s_m$ and $j, j' \in \mathbb{Z}_{m+1}$ be given. Assume that process P_i terminates by Line (4) with output v from state s_j and some other process $P_{i'}$ does the same with output v' from state $s_{j'}$. Then $v = v'$.*

Proof. For the sake of contradiction, suppose $v \neq v'$. Let r and r' denote the final round numbers of P_i and $P_{i'}$, respectively. Without loss of generality, assume that $r \leq r'$. By Lemma 12.4.9, we know that $s_j \models \Phi(v, v', r)$, therefore

$s_j.\mathsf{mem}(r, v) = 1$ and $s_j.\mathsf{mem}(r-1, v') = 0$. On the other hand, $s_{j'}.r_{i'} = r'$ and $s_{j'}.p_{i'} = v'$, so by Lemma 12.4.4 we have $s_{j'}.\mathsf{mem}(r', v') = 1$.

First we consider the case in which $j < j'$. By Lemma 12.4.8, we know that $s_{j'}.\mathsf{mem}(r, v') = 0$. Applying Lemma 12.4.7, we have $s_{j'}.\mathsf{mem}(r', v') = 0$, which yields a contradiction.

Next we consider the case in which $j' < j$. By Lemma 12.4.1, we may infer that $s_j.\mathsf{mem}(r', v') = 1$. By Lemma 12.4.7, this implies $s_j.\mathsf{mem}(r-1, v') = 1$, contradicting the fact that $s_j \models \Phi(v, v', r)$. □

It remains to consider termination by Line (2). Lemma 12.4.11 below implies that every process terminating by Line (2) must be preceded by a process terminating by Line (4) with the same decision. Using this claim, Theorem 12.4.12 is easily reduced to Lemma 12.4.10.

Lemma 12.4.11. *Let $v \in \mathbb{Z}_K$ and a path $s_0 \dots s_m$ be given. Assume that $s_m.\mathsf{mem}(R+1, v) = 1$. There is $j \in \mathbb{Z}_{m+1}$ such that some process P_i terminates with decision value v by executing Line (4) from s_j.*

Proof. Let j be the index for the first state in this path such that $s_j.\mathsf{mem}(R+1, v) = 1$. Since $\mathsf{mem}(R+1, v)$ is initialized to 0, we know that $j > 0$. Let P_i be the first process writing to $\mathsf{mem}(R+1, v)$ from s_{j-1}. Then P_i must have terminated with decision value v by executing Line (4) from s_{j-1}. □

Theorem 12.4.12. *Let a path $s_0 \dots s_m$ be given. Assume that process P_i terminates by executing either Line (2) or Line (4) from state s_j ($j \in \mathbb{Z}_{m+1}$) and its decision value is v. Similarly for $P_{i'}$, $s_{j'}$ and v'. Then $v = v'$.*

Proof. We claim that there exist $j'' \in \mathbb{Z}_{m+1}$ and $i'' \in \mathbb{Z}_N$ such that $P_{i''}$ terminates with decision value v by executing Line (4) from $s_{j''}$. If P_i terminates by Line (4), then we simply set $i'' := i$ and $j'' := j$. Otherwise, P_i terminates by Line (2) and the invocation of ReadMem in Line (1) of the last phase of P_i must have returned $v \neq K$. By Lemma 12.4.2 and Lemma 12.4.1, we know that $s_m.\mathsf{mem}(R+1, v) = 1$. We can then choose j'' and i'' using Lemma 12.4.11.

The same claim also holds for v'. Now we apply Lemma 12.4.10 to infer that $v = v'$. □

12.5 Probabilistic Termination and Expected Complexity

Let us first consider the amount of work required during each phase of the algorithm. (Recall that a phase is an entire pass through the **while**-loop in Figure 12.1(a)). Notice each phase involves at most (i) three calls to ReadMem, (ii) one write operation and (iii) one call to Jump. Each call to ReadMem requires $O(1)$ read operations, because the size K of the preference domain is a constant in our analysis. Therefore, aside from Jump, each phase involves constant work.

Consider the **while**-loop in Jump. Each pass through this loop involves at most one call to ReadMem. Furthermore, this loop is executed at most $\log R+1$ times. Since $R = 2^{\lceil \log N \rceil}$ by definition, each call to Jump requires $O(\log(\log N))$ read operations. This is then also the cost of a complete phase. Later on, we will prove that the expected number of complete phases until at least one process terminates successfully is $O(N)$ and hence the expected number of read/write operations is $O(N \log(\log N))$ (Lemma 12.5.5).

For any state s, let $s.r_{\max}$ denote the highest round reached by any process in state s. In other words, $s.r_{\max} := \max_{i \in \mathbb{Z}_N} s.r_i$. Since the two updates in Line (7) of Figure 12.1(a) are performed in a single step, $s.r_{\max}$ is also the largest r such that $s.\mathsf{mem}(r, v) = 1$ for some value $v \in \{0, \ldots, K-1\}$.

Lemma 12.5.1 below says, if no value advances to round $r_{\max}+1$, a lagging process can catch up to round $r_{\max}$ in one complete phase. First we argue that a lagging process reaches Line (8) in its first complete phase after s_m. Then, based on the **while**-loop in Jump, we construct a nested sequence of intervals shrinking to the singleton $\{s_m.r_{\max}\}$. Therefore $s_m.r_{\max}$ is the round number returned by Jump.

Lemma 12.5.1. *Let $s_0 \ldots s_m \ldots s_{m'}$ be a path with $m < m'$. Assume that $s_j.r_{\max} = s_m.r_{\max}$ for every $j \in \{m, \ldots, m'\}$. Moreover, assume that P_i completes a phase between s_m and $s_{m'}$ without crashing, successfully terminating or switching to the exit algorithm. Then $s_{m'}.r_i = s_m.r_{\max}$.*

Proof. For readability, write $r_{\max}$ for $s_m.r_{\max}$ and r for $s_m.r_i + 1$. Consider the first complete phase executed by P_i between s_m and s'_m. Without loss of generality, assume that s_m is the first state in that phase and that $r \le r_{\max}$.

By assumption, P_i does not crash, terminate, or exit. Therefore it reaches Line (6) in this phase. By Lemma 12.4.1 and Lemma 12.4.7, $r \le r_{\max}$ implies there is $v \in \mathbb{Z}_K$ such that $s_j.\mathsf{mem}(r, v) = 1$ for all $j \in \{m, \ldots, m'\}$. Hence the invocation of ReadMem in Line (6) returns a value other than K and P_i executes Line (8). It remains to show Jump returns $r_{\max}$ for the round number.

Note that Jump returns r if $r \ge R$, in which case $r = R = r_{\max}$. Otherwise, let c denote $\lceil \log(R - r) \rceil$. The **while**-loop of Jump calculates the following sequence $\{r'_0, \ldots, r'_c\}$ of natural numbers: r'_0 is r and r'_{i+1} is

- r'_i, if $r'_i + 2^{c-i} > R$ or $\mathsf{ReadMem}(r'_i + 2^{c-i}, K)$ returns K;
- $r'_i + 2^{c-i}$, otherwise.

From this we obtain a nested sequence of intervals:

$$[r'_0, r'_0 + 2^c),\ \ldots\ , [r'_i, r'_i + 2^{c-i}),\ \ldots,\ [r'_c, r'_c + 2^0).$$

It is easy to see that $r_{\max}$ belongs to every one of these intervals and, since the last is a singleton, we know $r'_c = r_{\max}$. This is precisely the round number returned by Jump. □

In Lemma 12.5.2, we show that the probability of at least one process terminating successfully within the next two rounds is bounded below by a constant, provided $r_{\max}$ is at most $R-2$. Moreover, this termination takes place before $15N$ complete phases are executed. The proof resembles the analysis given in [CIL94]; namely, we consider two events:

- E_1 is "a success occurs before $5N$ attempts to move from r to $r+1$ are made and all subsequent such attempts fail," and
- E_2 is "a success occurs before $5N$ attempts to move from $r+1$ to $r+2$ are made."

We argue that the conjunction of E_1 and E_2 implies at least one process terminates successfully in round $r+2$ before $15N$ complete phases are executed. Moreover, the probability of both E_1 and E_2 occurring is at least 0.511, using the fact that $\{(1-\frac{1}{n})^n\}_{n=2}^{\infty}$ increases to the limit $\frac{1}{e}$.

Lemma 12.5.2. *Suppose* ModifiedCIL *starts from a reachable state s. Let r denote $s.r_{\max}$ and suppose $r \leq R-2$. Then, with probability greater than 0.511, at least one process terminates successfully in a round no higher than $r+2$. Moreover, at most $15N$ complete phases are executed between s and the successful termination.*

Proof. By assumption, at least one process survives throughout the execution of the algorithm. Therefore, if no successful termination ever takes place, the algorithm stops only if all surviving processes reach round R and switch to the exit algorithm. Without loss of generality, we assume that no successful termination occurs in round $r+1$ or lower.

Consider the first complete phase following s. There are two cases.

- It is executed by a process P_i in round $< r$. By Lemma 12.5.1, P_i reaches round r by the end of this phase.
- It is executed by a process P_i in round r. Then P_i reaches Line (7) in this phase and, with probability $\frac{1}{2N}$, P_i advances to round $r+1$.

Suppose that either the first case applies, or the second case applies but P_i fails to advance to round $r+1$. Then the same case distinction can be made on the next complete phase. This repeats until all surviving processes have reached round r and, after that point, every complete phase involves a coin toss to advance to round $r+1$ until a success occurs. Moreover, since a lagging process catches up to round r in one complete phase, at most N complete phases following s are executed by processes in round strictly lower than r.

Consider the event E_1, in which "a success occurs before $5N$ attempts to move from r to $r+1$ are made" and "all subsequent attempts to move from r to $r+1$ fail." Notice the first condition is equivalent to "it is not the case that all of the

first $5N$ attempts to move from r to $r+1$ fail," which occurs with probability $1-(1-\frac{1}{2N})^{5N}$. By the reasoning above, this first success occurs before $6N$ complete phases are executed following s.

Let P_i be the successful process, thus the first to reach round $r+1$. By our atomicity assumption, $\mathsf{mem}(r+1, s.p_i)$ is set to 1 as soon as P_i reaches round $r+1$. After that point, every other $P_{i'}$ tosses a coin at most once to advance from r to $r+1$. This is because in the subsequence phase $P_{i'}$ sees it's no longer leading and therefore does not execute Line (7). As a result, the probability of "all subsequent attempts to move from r to $r+1$ fail" is at least $(1-\frac{1}{2N})^{N-1}$ and hence the probability of E_1 is at least $(1-(1-\frac{1}{2N})^{5N})(1-\frac{1}{2N})^{N-1}$. Moreover, after P_i reaches round $r+1$, at most $2N-2$ complete phases are executed by processes in round r or lower: at most $N-1$ failed coin tosses to move from r to $r+1$ and at most $N-1$ phases to catch up to $r+1$.

By assumption, no successful termination takes place until a process has reached round $r+2$. Thus, every complete phase executed by a process in round $r+1$ is a coin toss to move to round $r+2$, until a success occurs. Let E_2 denote the event that "a success occurs before $5N$ attempts to move from $r+1$ to $r+2$ are made." The probability of E_2 given E_1 is then $1-(1-\frac{1}{2N})^{5N}$. Similarly, this success occurs before $6N+(2N-2)+5N = 13N-2$ complete phases are executed following s and, after this success, at most $2N-2$ complete phases are executed by processes in round $r+1$ or lower.

Therefore, given E_1 and E_2, at least one process executes a complete phase in round $r+2$ before $15N$ complete phases are executed following s. Due to E_1, no process reaches round $r+1$ with preference value other than $s.p_i$. Therefore the first process to complete a phase in round $r+2$ sees no disagreement in round $r+1$ or higher. It then terminates successfully by Line (4). It remains to consider the probability of both E_1 and E_2 occurring. Recall that $\{(1-\frac{1}{n})^n\}_{n=2}^{\infty}$ is an increasing sequence with limit $\frac{1}{e}$. Therefore $\{(1-\frac{1}{2n})^{n-1}\}_{n=2}^{\infty}$ is a decreasing sequence with limit $\frac{1}{\sqrt{e}}$ and $\{1-(1-\frac{1}{2n})^{5n}\}_{n=2}^{\infty}$ is a decreasing sequence with limit $1-\frac{1}{e^{2.5}}$. Therefore, the probability of both E_1 and E_2 occurring is at least

$$(1-(1-\frac{1}{2N})^{5N})^2 \cdot (1-\frac{1}{2N})^{N-1} \geq (1-\frac{1}{e^{2.5}})^2 \cdot \frac{1}{\sqrt{e}} > 0.511.$$

□

Notice Lemma 12.5.2 applies only to executions starting in round $R-2$ or lower. The next lemma covers rounds $R-1$ and R, assuming a decision is reached without switching to the exit algorithm.

Lemma 12.5.3. *Suppose* ModifiedCIL *starts from a reachable state s. Let r denote $s.r_{\max}$ and suppose $R-2 < r \leq R$. Assuming the exit algorithm is not invoked, the (conditional) probability that at least one process terminates successfully before $15N$ complete phases are executed after s is greater than* 0.511.

Proof. We use arguments similar to those in the proof of Lemma 12.5.2. First suppose $r = R$. Then at most $N - 1$ complete phases are executed before a process completes a phase in round R. Suppose P_i is the first to do so. If P_i does not terminate by Line (4) in that phase, it must be the case that $\mathsf{mem}(R-1, v_1) = \mathsf{mem}(R-1, v_2) = 1$ for some $v_1 \neq v_2$. Then P_i, as well as every other process that reaches round R, invokes the exit algorithm. By assumption, this is not the case and hence P_i terminates by Line (4) in that phase. Therefore, with probability 1, at least one process terminates before N complete phases are executed.

If $r = R-1$, then at most $N-1$ complete phases are executed before a process completes a phase in round $R-1$. Similar to the previous case ($r = R$), if the first process completing a phase in round $R-1$ does not terminate by Line (4) in that phase, every process reaching round $R-1$ will try to advance to round R by Line (7), until one of them succeeds. The probability of at least one success before $4N$ attempts are made is $1 - (1 - \frac{1}{2N})^{4N}$, which is bounded below by $(1 - \frac{1}{e^2}) > 0.864$. After that success, the problem reduces to the case where $r = R$ and successful termination is guaranteed before N complete phases. Therefore, with probability at least 0.864, at least one process terminates before $6N$ complete phases are executed. □

We now prove probabilistic wait-free termination.

Theorem 12.5.4. *If the exit algorithm is wait-free and satisfies probabilistic termination, the same holds for* ModifiedCIL.

Proof. By correctness of the exit algorithm, we may focus on the case in which the exit algorithm is not invoked. Consider execution blocks of $15N$ complete phases each. By Lemma 12.5.2 and Lemma 12.5.3, the probability of successful termination within each block is at least 0.511. Thus, with probability 1, the algorithm terminates successfully after a finite number of blocks. Since we have made no assumption on the number of surviving processes, the algorithm is wait-free. □

Next, we turn to complexity considerations. First we consider the case in which the exit algorithm is not invoked. As before, the execution is divided into blocks of $15N$ complete phases and we argue that the expected number of blocks is at most 2. Hence the expected number of complete phases is $O(N)$. Since each phase involves $O(\log(\log N))$ elementary operations, the expected number of elementary operations $O(N \log(\log N))$.

Lemma 12.5.5. *Assume that the exit algorithm is not invoked. The expected number of elementary* read/write *operations until at least one process terminates successfully is* $O(N \log(\log N))$.

Proof. Again we consider execution blocks of $15N$ complete phases each. The expected number of blocks is:

$$\sum_{n=1}^{\infty}(n \cdot 0.511 \cdot (1-0.511)^{n-1}) = \sum_{n=0}^{\infty}((n+1) \cdot 0.511 \cdot 0.489^n).$$

Factoring out 0.511 and rearranging the summands, we have

$$\begin{aligned}
&\sum_{n=0}^{\infty}((n+1) \cdot 0.511 \cdot 0.489^n) \\
&= 0.511 \cdot (\sum_{n=0}^{\infty} 0.489^n + \sum_{n=1}^{\infty} 0.489^n + \sum_{n=2}^{\infty} 0.489^n + \ldots) \\
&= 0.511 \cdot (\sum_{n=0}^{\infty} 0.489^n + 0.489 \sum_{n=0}^{\infty} 0.489^n + 0.489^2 \sum_{n=0}^{\infty} 0.489^n + \ldots) \\
&= 0.511 \cdot (\sum_{n=0}^{\infty} 0.489^n)^2 \\
&= 0.511 \cdot (\frac{1}{0.511})^2 = \frac{1}{0.511} < 2.
\end{aligned}$$

Thus the expected number of complete phases is at most $30N$. Moreover, there are at most $N-1$ incomplete phases. Since each phase involves $O(\log(\log N))$ elementary operations, the expected number of elementary operations is at most $O(N \log(\log N))$. □

If the exit algorithm is in fact invoked, we consider the expected number of elementary operations both before and after switching. The former is shown to be $O(N(\log N)(\log(\log N)))$ and, as proven in [CIL94], the latter is $O(N^2)$. This implies the overall expected complexity in this case is $O(N^2 \log(\log N))$.

Lemma 12.5.6. *Suppose the exit algorithm is the original CIL algorithm and is invoked. The expected number of elementary* read/write *operations until at least one process terminates successfully is* $O(N^2 \log(\log N))$.

Proof. In this case the algorithm steps through all R rounds without successful termination. Using a similar calculation as in the proof of Lemma 12.5.5, the expected number of coin tosses to move from r to $r+1$ is

$$\sum_{n=1}^{\infty} n(\frac{1}{2N})(1-\frac{1}{2N})^{n-1} = 2N.$$

Following each success, at most $N-1$ phases are executed by processes lagging behind. Therefore, the expected number of complete phases before switching to original CIL is at most $3NR \leq 6N(\log N + 1)$. The expected number of elementary operations before switching is then $O(N(\log N)(\log(\log N)))$.

In [CIL94], it is shown that the expected number of elementary operations for the original CIL algorithm is $O(N^2)$. Therefore, the overall expected number of elementary operations is $O(N^2 \log(\log N))$. □

Lemma 12.5.7. *Suppose the* ModifiedCIL *starts from the initial state s_0. The probability of failing to reach a decision in or before round R is at most $1/N$.*

Proof. By Lemma 12.5.2, this probability is at most $(1 - 0.511)^{\frac{R}{2}}$. Since $R = 2\lceil \log N \rceil$, we have

$$(1 - 0.511)^{\frac{R}{2}} \leq (1 - 0.511)^{\log N} < (0.5)^{\log N} = \frac{1}{N}.$$

□

Putting together Lemmas 12.5.5, 12.5.6, and 12.5.7, we conclude that the expected complexity of ModifiedCIL is $O(N \log(\log N))$.

Theorem 12.5.8. *Suppose* ModifiedCIL *starts from the initial state s_0 and the exit algorithm is original CIL. The expected number of elementary* read/write *operations until at least one process terminates successfully is $O(N \log(\log N))$.*

12.6 Model Checking

It is quite straightforward to specify our algorithm in PRISM's state-based input language. Each process is modeled as a *module* and the shared memory is modeled using global variables. Two more global variables are used to keep track of process failures and the number of completed phases.

We consider binary consensus (i.e., $K = 2$) with $N = 2, 3, 4$ processes. Processes are assumed to disagree initially, therefore validity is trivial. Agreement is satisfied in all models constructed. For probabilistic termination, we ask PRISM to compute the (exact) minimum probability of at least one process terminating successfully, given an allowance of $R = 2\lceil \log N \rceil$ rounds and $15N \cdot \frac{R}{2} = 15N\lceil \log N \rceil$ complete phases. This result is compared against our analytic lower bound of $1 - \frac{1}{N}$.

Below we present (portions of) our PRISM model. Figures 12.2 shows the front matter, including declarations of global constants and global variables. The key word "nondeterministic" indicates that the underlying model is that of Markov decision processes. Figures 12.3 and 12.4 contain the PRISM code for a process whose initial preference is 0. All other processes can be obtained from this one using syntactic replacements. Model checking is then performed on the parallel composition of all process modules.

In the case of $N = 4$, the model becomes too complex (with $2\lceil \log N \rceil = 4$ rounds and $15N\lceil \log N \rceil = 120$ complete phases). However, we discover that the analytic bound of $1 - \frac{1}{N} = 0.750$ is already met when we restrict to 40 complete

```
//////////////////////////////////////////
// PRISM model for the modified CIL algorithm
// 4 processes
// bias = 1/2N = 0.125
// binary consensus, maximum round number R = 4 = 2 log N
// initial configuration: round, preference
// process 0 : 0,0
// process 1 : 0,1
// process 2 : 0,0
// process 3 : 0,1

//////////////////////////////////////////
nondeterministic

// number of processes
const int N=4;

// probability of advancing to next round prob = 1/2N
const double bias=0.125;

// number of preference values
const int K=2;

// protocol starts at round 0
// stops if decision is reached or round R is reached without decision
// R = 2 log N
const int R=4;

// maximum number of complete phases 15N*R/2=120
// can't build the model with that many, so try something lower
const int numPhases=40;

//////////////////////////////////////////
// shared memory
// mem(i,b): a process in round i prefers/preferred b
// initial disagreement, no need to have mem00 and mem01
// mem(R+1,b) is used to indicate decision

global mem10 : [0..1] init 0;
global mem11 : [0..1] init 0;
global mem20 : [0..1] init 0;
global mem21 : [0..1] init 0;
global mem30 : [0..1] init 0;
global mem31 : [0..1] init 0;
global mem40 : [0..1] init 0;
global mem41 : [0..1] init 0;
global mem50 : [0..1] init 0;
global mem51 : [0..1] init 0;

// number of non-faulty processes
global NF : [1..N] init N;

// number of complete phases remaining
global q : [0..numPhases] init numPhases;
```

Figure 12.2: PRISM Model: Front Matter.

```
////////////////////////////////////////
// process0, preferring 0
// REMARK
// Jump is not modeled due to limitations in the PRISM language
// a lagging process walks one step at a time

module proc0

  // preference value
  p0 : [0..1] init 0;

  // round number R=4
  r0 : [0..R] init 0;

  // status
  s0 : [0..7] init 1;
  // 0 crashed or switching to exit algorithm
  // 1 read mem(R+1,0)
  // 2 read mem(R+1,1)
  // 3 read mem(r0-1,1-p0)
  // 4 read mem(r0+1,0)
  // 5 read mem(r0+1,1)
  // 6 random-write
  // 7 decided

  ////////////////////////////////////////////
  // crash failure
  [] (NF>1) & (s0!=0) -> (NF'=NF-1) & (s0'=0);

  // read from memory
  // read mem(R+1,0)
  [] (s0=1) & (mem50=1) & (q>=1) -> (s0'=7) & (p0'=0) & (q'=q-1);
  [] (s0=1) & (mem50=0) & (q>=1) -> (s0'=2);

  // read mem(R+1,1)
  [] (s0=2) & (mem51=1) & (q>=1) -> (s0'=7) & (p0'=1) & (q'=q-1);
  [] (s0=2) & (mem51=0) & (q>=1) -> (s0'=3);

  // read mem(r0-1,1-p0)
  // round-0 entries all initialized to 1
  [] ((r0=0)—(r0=1)) & (s0=3) & (q>=1) -> (s0'=4);
  [] (r0=2) & (s0=3) & (p0=0) & (mem11=0) & (q>=1) -> (s0'=7) & (mem50'=1) & (q'=q-1);
  [] (r0=2) & (s0=3) & (p0=1) & (mem10=0) & (q>=1) -> (s0'=7) & (mem51'=1) & (q'=q-1);
  [] (r0=2) & (s0=3) & (p0=0) & (mem11=1) & (q>=1) -> (s0'=4);
  [] (r0=2) & (s0=3) & (p0=1) & (mem10=1) & (q>=1) -> (s0'=4);
  [] (r0=3) & (s0=3) & (p0=0) & (mem21=0) & (q>=1) -> (s0'=7) & (mem50'=1) & (q'=q-1);
  [] (r0=3) & (s0=3) & (p0=1) & (mem20=0) & (q>=1) -> (s0'=7) & (mem51'=1) & (q'=q-1);
  [] (r0=3) & (s0=3) & (p0=0) & (mem21=1) & (q>=1) -> (s0'=4);
  [] (r0=3) & (s0=3) & (p0=1) & (mem20=1) & (q>=1) -> (s0'=4);
  [] (r0=4) & (s0=3) & (p0=0) & (mem31=0) & (q>=1) -> (s0'=7) & (mem50'=1) & (q'=q-1);
  [] (r0=4) & (s0=3) & (p0=1) & (mem30=0) & (q>=1) -> (s0'=7) & (mem51'=1) & (q'=q-1);
  [] (r0=4) & (s0=3) & (p0=0) & (mem31=1) & (q>=1) -> (s0'=0) & (q'=q-1);
  [] (r0=4) & (s0=3) & (p0=1) & (mem30=1) & (q>=1) -> (s0'=0) & (q'=q-1);

  // read mem(r0+1,0)
  // This doesn't happen in round 4
  [] (r0=0) & (s0=4) & (mem10=1) & (q>=1) -> (r0'=r0+1) & (p0'=0) & (s0'=1) & (q'=q-1);
  [] (r0=0) & (s0=4) & (mem10=0) & (q>=1) -> (s0'=5);
  [] (r0=1) & (s0=4) & (mem20=1) & (q>=1) -> (r0'=r0+1) & (p0'=0) & (s0'=1) & (q'=q-1);
  [] (r0=1) & (s0=4) & (mem20=0) & (q>=1) -> (s0'=5);
  [] (r0=2) & (s0=4) & (mem30=1) & (q>=1) -> (r0'=r0+1) & (p0'=0) & (s0'=1) & (q'=q-1);
  [] (r0=2) & (s0=4) & (mem30=0) & (q>=1) -> (s0'=5);
  [] (r0=3) & (s0=4) & (mem40=1) & (q>=1) -> (r0'=r0+1) & (p0'=0) & (s0'=1) & (q'=q-1);
  [] (r0=3) & (s0=4) & (mem40=0) & (q>=1) -> (s0'=5);
```

Figure 12.3: PRISM Model: Process with Preference 0, Part I.

```
////////////////////////////////////////
// process0, preferring 0, continued

  // read mem(r0+1,1)
  // This doesn't happen in round 4
  [] (r0=0) & (s0=5) & (mem11=1) & (q>=1) -> (r0'=r0+1) & (p0'=1) & (s0'=1) & (q'=q-1);
  [] (r0=0) & (s0=5) & (mem11=0) & (q>=1) -> (s0'=6);
  [] (r0=1) & (s0=5) & (mem21=1) & (q>=1) -> (r0'=r0+1) & (p0'=1) & (s0'=1) & (q'=q-1);
  [] (r0=1) & (s0=5) & (mem21=0) & (q>=1) -> (s0'=6);
  [] (r0=2) & (s0=5) & (mem31=1) & (q>=1) -> (r0'=r0+1) & (p0'=1) & (s0'=1) & (q'=q-1);
  [] (r0=2) & (s0=5) & (mem31=0) & (q>=1) -> (s0'=6);
  [] (r0=3) & (s0=5) & (mem41=1) & (q>=1) -> (r0'=r0+1) & (p0'=1) & (s0'=1) & (q'=q-1);
  [] (r0=3) & (s0=5) & (mem41=0) & (q>=1) -> (s0'=6);

  // leading with competition, toss coin to advance
  // This doesn't happen in round 4
  [] (r0=0) & (p0=0) & (s0=6) & (q>=1) ->
    quad bias:(r0'=r0+1)&(mem10'=1)&(s0'=1)&(q'=q-1) + (1.0-bias):(s0'=1)&(q'=q-1);
  [] (r0=0) & (p0=1) & (s0=6) & (q>=1) ->
        bias:(r0'=r0+1)&(mem11'=1)&(s0'=1)&(q'=q-1) + (1.0-bias):(s0'=1)&(q'=q-1);
  [] (r0=1) & (p0=0) & (s0=6) & (q>=1) ->
        bias:(r0'=r0+1)&(mem20'=1)&(s0'=1)&(q'=q-1) + (1.0-bias):(s0'=1)&(q'=q-1);
  [] (r0=1) & (p0=1) & (s0=6) & (q>=1) ->
        bias:(r0'=r0+1)&(mem21'=1)&(s0'=1)&(q'=q-1) + (1.0-bias):(s0'=1)&(q'=q-1);
  [] (r0=2) & (p0=0) & (s0=6) & (q>=1) ->
        bias:(r0'=r0+1)&(mem30'=1)&(s0'=1)&(q'=q-1) + (1.0-bias):(s0'=1)&(q'=q-1);
  [] (r0=2) & (p0=1) & (s0=6) & (q>=1) ->
        bias:(r0'=r0+1)&(mem31'=1)&(s0'=1)&(q'=q-1) + (1.0-bias):(s0'=1)&(q'=q-1);
  [] (r0=3) & (p0=0) & (s0=6) & (q>=1) ->
        bias:(r0'=r0+1)&(mem40'=1)&(s0'=1)&(q'=q-1) + (1.0-bias):(s0'=1)&(q'=q-1);
  [] (r0=3) & (p0=1) & (s0=6) & (q>=1) ->
        bias:(r0'=r0+1)&(mem41'=1)&(s0'=1)&(q'=q-1) + (1.0-bias):(s0'=1)&(q'=q-1);

endmodule
```

Figure 12.4: PRISM Model: Process with Preference 0, Part II.

phases. This suggests that we have made some overly conservative estimates while deriving the analytic bound.

The table below summarizes our model checking results. We use PRISM version 2.1, running on a 1.4 GHz Pentium M machine with 500 Mb memory under Linux 2.6. The MTBDD engine is used with a CUDD memory limit of 400 Mb. Other parameters remain at default settings. All relevant files, including model checking logs, can be found in [Che05a].

N	R	#Phases	Model		Agreement	Termination		
			#States	Time(s)	Time(s)	Time(s)	MinProb	AnalyticBd
2	2	30	42,320	4	0.025	6	0.745	0.511
3	4	90	12,280,910	213	0.094	2,662	0.971	0.667
4	2	60	45,321,126	429	0.078	602	0.755	0.511
4	4	40	377,616,715	5224	3.926	55,795	0.765	0.750

12.7 Conclusions

We have given a simple algorithm that solves asynchronous wait-free consensus in expected $O(N \log(\log N))$ total work. We follow a value-based (as opposed to process-based) approach and make use of MWMR atomic registers. This strategy, also adopted in [Cha96, Aum97], leads to a significant reduction in data handling and hence more efficient consensus algorithms. As a pleasant side-effect, the reduction in both global and local data makes model checking significantly more feasible, for it helps to avoid the typical state explosion problem.

MWMR memory is often regarded as a stronger primitive than SWMR memory. Indeed, there are optimal implementations of MWMR from physical SWMR registers using linear time and logarithmic space [IS92]. However, if one makes comparisons from the basis of SWSR, then MWMR and SWMR become roughly the same: when implemented from SWSR, both require linear time and logarithmic space. Moreover, it is argued in [BPSV00] that SWMR memory requires the hidden assumption of *naming*: existence of distinct identifiers known to all. In that sense, MWMR is a weaker primitive compared to SWMR. This idea is echoed by the fact that, unlike the original CIL algorithm, our version allows processes to participate anonymously.

The MWMR strategy has another advantage, namely, flexibility in memory usage. We have shown that, with high probability, consensus can be reached using $O(\log N)$ many single-bit MWMR registers. (That is, the main algorithm succeeds and thus the exit algorithm is not invoked.) This can be seen as a temporary reprieve from the lower bound of $\Omega(\sqrt{N})$ for the space requirement of randomized consensus [FHS98]. In practice, one may be willing to accept a small probability of failing to reach consensus, in which case we can remove the exit algorithm altogether. The main algorithm can be repeated to increase the

success probability, and memory is allocated only as needed.

To our best knowledge, our algorithm is faster (in terms of expected total work) than all other algorithms for dynamic adversaries, with the exception of [AKL99]. The algorithm of [AKL99] works in a SWSR setting and is shown to be $O(N \log N\, e^{\sqrt{\log N}})$ in expected total work. To have a fair comparison, we should take into account an emulation of MWMR from SWSR, thus adding a linear slowdown to our complexity result.

However, the algorithm of [AKL99] is based on a primitive called *cooperative sharing*, in which processes propagate knowledge by reading and writing entire knowledge sets into registers. These knowledge sets grow throughout the execution and may eventually contain the identifiers and input values of all N processes. Therefore polynomial-size registers are necessary. In contrast, we require constant-size registers. Even if our registers are provided by an emulation from SWSR, it is sufficient to have logarithmic-size registers [IS92].

For future work, we want to improve the per process work bound of our algorithm. In [AW96], a similar improvement is achieved by allowing fast processes to cast votes of increasing weights. However, their proofs rely on properties of Martingale processes and cannot be adapted immediately to our setting. At this time, we do not know if per process work is inherently high in our setting (e.g., $\Omega(\frac{N}{f(N)})$, where f is a polylogarithmic function).

Another possibility for future work is to consider *contention cost*, which measures the amount of conflict in memory access [AB04]. The contention cost for ModifiedCIL is high because, in a roughly synchronous execution, all N processes try to access a constant number of registers at the same time. It would be interesting to modify the algorithm further to reduce contention.

Finally, we comment on model checking using PRISM. Although the current limit seems to be 4 processes, we conjecture a vast improvement using a symmetry reduction option, which is under development by the PRISM team. Before symmetry reduction is available, manual abstraction can be used to increase feasibility. That is, we manually construct an abstraction that captures core ideas of an algorithm, while significantly decreasing the model size. We experimented with such an abstraction of original CIL, by focusing on the shared memory and filtering out local states of processes. Having done so, we were in fact able to handle up to 10 processes. However, it is non-trivial to prove soundness of the abstraction. Standard techniques such as probabilistic simulation are available for this purpose, but substantial investment of time is required.

Overall, PRISM allows us to conduct experiments during the development stage of an algorithm, with minimal learning effort. Although in most cases it still cannot handle large instances of a full algorithm, it is perfectly feasible to model check a subroutine or an abstract version. This already provides valuable information, especially to those who simply wish to gain more insight into an algorithm.

Part IV

Conclusions

13

Conclusions

This PhD project began with the following question: "Can one define a compositional trace-style semantics for probabilistic automata?" This quickly led to an even more fundamental question: "How should one define parallel composition for probabilistic automata?" In order to answer these questions, we researched the literature extensively and tried to understand the various treatments of nondeterminism and scheduling.

An interesting observation is that different research communities have different traditions and preferences for scheduling mechanisms. In the formal methods community, for example, one tends to invent scheduling mechanisms that are mathematically simple, so that scheduling does not add to the complexity of verification. In contrast, the community of distributed computing is more willing to sacrifice simplicity of scheduling for better performance of algorithms. Various scheduling assumptions are introduced in order to sidestep lower bound results, despite the fact that these assumptions complicate correctness proofs. Yet another tradition is found in the area of cryptographic protocols, where scheduling is designated to an adversarial entity.

These observations had some significant influences over our work. On the one hand, we would like our formalisms to be mathematically simple and, if possible, to resemble existing formalisms. This increases the likelihood that our work can be integrated with other efforts in probabilistic verification, especially those in tool development. On the other hand, we would like to be able to model the various scheduling assumptions that are used in distributed computing, as well as adversarial scheduling for cryptographic protocols. Ideally, this should be done in a systematic manner, so that minimal effort is required to switch from one scheduling assumption to another.

After several attempts, we have come to favor a two-leveled approach. On the lower level, we aim for a basic framework that uses a simple scheduling mechanism and supports compositional reasoning. Moreover, it is important to keep in mind the possibility of mechanization, since correctness proofs of distributed algorithms are notoriously labor intensive.

Then, within this basic framework, one can implement a scheduler component, very much like the arbiter automaton described in Section 10.5. This scheduler component can be viewed as a parameter that specifies the desired scheduling

assumption. In other words, to switch from one assumption to another, one simply needs to replace the scheduler component. Such flexibility will be the main appeal of this two-leveled modeling paradigm.

We believe that the switched PIOA framework of Chapter 10 is very suitable for implementing a two-leveled paradigm. On the lower level, the notion of (local) I/O schedulers is a simple extension of the familiar notion of perfect-information schedulers. On the higher level, one can use arbiter automata to specify which part of the execution history may influence scheduling decisions.

The framework proposed in Chapter 11 is less flexible in this respect, because we tried to hardwire a particular scheduling assumption (i.e., local-oblivious scheduling) using a number of axioms. Nonetheless, it serves to confirm our claim that compositionality can indeed be achieved if local scheduling is clearly separated from global scheduling. The same idea may very well be carried out in other formalisms in order to achieve compositionality.

To conclude, we believe we have made significant progress on the topic of compositionality. Although our proposals may still be too complex for practical applications and tool implementation, we have identified the key issues that made compositionality difficult to achieve. We hope that our ideas will inspire further research in probabilistic verification and will contribute to the advances of this field.

Bibliography

[AB04] Y. Aumann and M.A. Bender, *Efficient low-contention asynchronous consensus with the value-oblivious adversary scheduler*, Distributed Computing (2004), Accepted in 2004.

[Abr88] K. Abrahamson, *On achieving consensus using a shared memory*, Proceedings PODC'88, ACM Press New York, 1988, pp. 291–302.

[ABZ97] Y. Aumann, M. Bender, and L. Zhang, *Efficient execution of nondeterministic parallel programs on asynchronous systems*, Information and Computation **139** (1997), no. 1, 1–16.

[ACT00] M.K. Aguilera, W. Chen, and S. Toueg, *Failure detection and consensus in the crash recovery model*, Distributed Computing **13** (2000), no. 2, 99–125.

[Agg94] S. Aggarwal, *Time optimal self-stabilizing spanning tree algorithms*, Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1994, Available as Technical Report MIT/LCS/TR-632.

[AH90] J. Aspnes and M. Herlihy, *Fast randomized consensus using shared memory*, Journal of Algorithms **11** (1990), no. 3, 441–461.

[AH99] R. Alur and T. Henzinger, *Reactive modules*, Formal Methods in System Design **15** (1999), no. 1, 7–48.

[AKL99] Y. Aumann and A. Kapah-Levy, *Cooperative sharing and asynchronous consensus using single-read single-writer registers*, Proceedings of the 10th ACM-SIAM Annual Symposium on Discrete Algorithms (SODA), 1999, pp. 61–70.

[AM99] J.H. Anderson and M. Moir, *Wait-free synchronization in multiprogrammed systems: integrating priority-based and quantum-based scheduling*, Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, 1999, pp. 123–132.

[Asp98] J. Aspnes, *Lower bounds for distributed coin-flipping and randomized consensus*, Journal of the ACM **45** (1998), no. 3, 415–450.

[Asp00] ———, *Fast deterministic consensus in a noisy environment*, Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, 2000, pp. 299–309.

[Asp03] ———, *Randomized protocols for asynchronous consensus*, Distributed Computing **16** (2003), no. 2-3, 165–175.

[Aum97] Y. Aumann, *Efficient asynchronous consensus with the weak adversary scheduler*, Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, 1997, pp. 209–218.

[AW96] J. Aspnes and O. Waarts, *Randomized consensus in expected $O(n \log^2 n)$ operations per process*, SIAM Journal on Computing **25** (1996), no. 5, 1024–1044.

[BBK87] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop, *On the consistency of Koomen's fair abstraction rule*, Theoretical Computer Science **51** (1987), no. 1/2, 129–176.

[BdA95] A. Bianco and L. de Alfaro, *Model checking of probabilistic and nondeterministic systems*, Proceedings Foundations of Software Technology and Theoretical Computer Science, LNCS, vol. 1026, 1995, pp. 499–513.

[Ber95] D.P. Bertsekas, *Dynamic programming and optimal control*, vol. 1–2, Athena Scientific, Belmont, MA, 1995.

[BK86] J.A. Bergstra and J.W. Klop, *Verification of an alternating bit protocol by means of process algebra*, Mathematical Methods of Specification and Synthesis of Software Systems '85, Math, Mathematical Research, vol. 31, Akademie-Verlag, 1986, pp. 9–23.

[BK98] C. Baier and M. Kwiatkowska, *Model checking for a probabilistic branching time logic with fairness*, Distributed Computing **11** (1998), no. 3, 125–155.

[BO83] M. Ben-Or, *Another advantage of free choice: completely asynchronous agreement protocols*, Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, 1983, pp. 27–30.

[BPSV00] H. Buhrman, A. Panconesi, R. Silvestri, and P.M.B. Vitányi, *On the importance of having an identity or is consensus really universal?*, Proceedings of the 14th International Conference on Distributed Computing, LNCS, vol. 1914, Springer-Verlag, 2000, pp. 134–148.

[BPW03] M. Backes, B. Pfitzmann, and M. Waidner, *A composable cryptographic library with nested operations*, Proceedings 10th ACM conference on Computer and Communications Security

(CCS) (S. Jajodia, V. Atluri, and T. Jaeger, eds.), ACM, 2003, Extended version at the Cryptology ePrint archive, `http://eprint.iacr.org/2003/015/`, pp. 220–230.

[BPW04a] ———, *A general composition theorem for secure reactive systems*, First Theory of Cryptography Conference (TCC 2004) (Cambridge, MA, USA) (M. Naor, ed.), LNCS, vol. 2951, Springer-Verlag, February 2004, pp. 336–354.

[BPW04b] ———, *Secure asynchronous reactive systems*, Cryptology ePrint Archive Report 2004/082, 2004.

[BR91] G. Bracha and O. Rachman, *Randomized consensus in expected $O(n^2 \log n)$ operations*, Proceedings of the 5th International Workshop on Distributed Algorithms, LNCS, vol. 579, 1991, pp. 143–150.

[Bro87] F.P. Brooks, *No silver bullet: essence and accidents of software engineering*, Computer **20** (1987), no. 4, 10–19.

[BSdV03] F. Bartels, A. Sokolova, and E. de Vink, *A hierarchy of probabilistic system types*, Proceedings CMCS 2003, ENTCS, vol. 82, Elsevier, 2003.

[Can01] R. Canetti, *Universally composable security: a new paradigm for cryptographic protocols*, Proceedings of the 42nd IEEE Symposium on Foundations of Computing, 2001, pp. 136–145.

[CB90] G. Casella and R.L. Berger, *Statistical inference*, Duxbury Press, Belmont, 1990.

[CCK+05] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala, *Using probabilistic I/O automata to analyze an oblivious transfer protocol*, Tech. Report MIT-LCS-TR-1001a, MIT CSAIL, 2005.

[CCK+06a] ———, *Levels-of-abstraction proofs for security protocols*, Submitted to CONCUR 06, 2006.

[CCK+06b] ———, *Task-structured probabilistic I/O automata*, Tech. Report MIT-CSAIL-TR-2006-023, MIT CSAIL, March 2006, Available at `http://hdl.handle.net/1721.1/32525`. Extended abstract submitted to ICALP '06.

[CDSY99] R. Cleaveland, Z. Dayar, S.A. Smolka, and S. Yuen, *Testing preorders for probabilistic processes*, Information and Computation **154** (1999), no. 2, 93–148.

[CH04] L. Cheung and J. Hughes, *Concise graphs and functional bisimulations*, ENTCS **100** (2004), 5–29, Proceedings CMCIM/GETCO 2003, Marseille, France.

[CH05] L. Cheung and M. Hendriks, *Causal dependencies in parallel composition of stochastic processes*, Tech. Report ICIS-R05020, Institute for Computing and Information Sciences, University of Nijmegen, 2005.

[Cha96] T.D. Chandra, *Polylog randomized wait-free consensus*, Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, 1996, pp. 166–175.

[Che05a] L. Cheung, *Collection of PRISM models of the modified CIL algorithm*, 2005, Available at `http://www.niii.ru.nl/~lcheung/mcil/`.

[Che05b] ———, *Randomized wait-free consensus using an atomicity assumption*, Proceedings 9th International Conference on Principles of Distributed Systems (OPODIS 2005), December 2005.

[Che05c] ———, *Randomized wait-free consensus using an atomicity assumption*, Tech. Report ICIS-R05035, Institute for Computing and Information Sciences, University of Nijmegen, 2005, Available at `http://www.niii.ru.nl/~lcheung/cilTR.pdf`.

[Che06] ———, *Reconciling nondeterministic and probabilistic choices*, Ph.D. thesis, ICIS, Radboud University Nijmegen, 2006, To appear.

[Chr90] I. Christoff, *Testing equivalence and fully abstract models of probabilistic processes*, Proceedings CONCUR 90, Lecture Notes in Computer Science, vol. 458, Springer-Verlag, 1990.

[CIL87] B. Chor, A. Israeli, and M. Li, *On processor coordination using asynchronous hardware*, Proceedings PODC'87, 1987, pp. 86–97.

[CIL94] ———, *Wait-free consensus using asynchronous hardware*, SIAM Journal on Computing **23** (1994), no. 4, 701–712.

[CLSV04a] L. Cheung, N.A. Lynch, R. Segala, and F.W. Vaandrager, *Switched probabilistic i/o automata*, Proceedings First International Colloquium on Theoretical Aspects of Computing (ICTAC2004), Lecture Notes in Computer Science, Springer-Verlag, 2004.

[CLSV04b] ———, *Switched probabilistic i/o automata*, Tech. Report NIII-R0437, University of Nijmegen, September 2004.

[CLSV06] ———, *Switched pioa: Parallel composition via distributed scheduling*, Theoretical Computer Science (2006), Special Issue on FMCO 04.

[Coh80] D.L. Cohn, *Measure theory*, Birkhäuser, Boston, 1980.

[CSV06] L. Cheung, M.I.A. Stoelinga, and F.W. Vaandrager, *A testing scenario for probabilistic processes*, Tech. Report ICIS-R06002, ICIS, Radboud University Nijmegen, January 2006, Submitted to Journal of the ACM.

[CY90] C. Courcoubetis and M. Yannakakis, *Markov decision processes and regular events*, Proceedings ICALP 90, LNCS, vol. 443, 1990, pp. 336–349.

[dA99] L. de Alfaro, *The verification of probabilistic systems under memoryless partial-information policies is hard*, Proceedings PROBMIV 99, 1999, pp. 19–32.

[dAH01] L. de Alfaro and T.A. Henzinger, *Interface automata*, Proceedings of the Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT Symposium on the Foundation of Software Engineering (ESEC/FSE-01) (V. Gruhn, ed.), Software Engineering Notes, vol. 26, ACM Press New York, September 2001, pp. 109–120.

[dAHJ01] L. de Alfaro, T.A. Henzinger, and R. Jhala., *Compositional methods for probabilistic systems*, Proceedings CONCUR 01 (K.G. Larsen and M. Nielsen, eds.), Lecture Notes in Computer Science, vol. 2154, Springer-Verlag, 2001, pp. 351–365.

[DEP02] J. Desharnais, A. Edalat, and P. Panangaden, *Bisimulation for labeled Markov processes*, Information and Computation **179** (2002), no. 2, 163–193.

[DGJP03] J. Desharnais, V. Gupta, R. Jagadeesan, and P. Panangaden, *Approximating labelled Markov processes*, Information and Computation **184** (2003), 160–200.

[DGRV00] M.C.A. Devillers, W.O.D. Griffioen, J.M.T. Romijn, and F.W. Vaandrager, *Verification of a leader election protocol - formal methods applied to IEEE 1394*, Formal Methods in System Design **16** (2000), no. 3.

[DHK98] P. D'Argenio, H. Hermanns, and J.-P. Katoen, *On generative parallel composition*, Proceedings PROBMIV'98, ENTCS, vol. 22, 1998, pp. 105–122.

[DKMR05] A. Datta, R. Küsters, J.C. Mitchell, and A. Ramanathan, *On the relationships between notions of simulation-based security*, Proceedings TCC 2005, 2005, pp. 476–494.

[DLS88] C. Dwork, N. Lynch, and L. Stockmeyer, *Consensus in the presence of partial synchrony*, Journal of the ACM **35** (1988), no. 2, 288–323.

[DP90] B.A. Davey and H.A. Priestley, *Introduction to lattices and order*, Cambridge University Press, Cambridge, 1990.

[Eda95] A. Edalat, *Domain theory in stochastic processes*, Proceedings LICS 95, 1995, pp. 244–254.

[EGL85] S. Even, O. Goldreich, and A. Lempel, *A randomized protocol for signing contracts*, CACM **28** (1985), no. 6, 637–647.

[FHS98] F. Fich, M. Herlihy, and N. Shavit, *On the space complexity of randomized synchronization*, Journal of the ACM **45** (1998), no. 5, 843–862.

[FLP85] M. Fischer, N.A. Lynch, and M.S. Paterson, *Impossibility of distributed consensus with one faulty process*, Journal of the ACM **32** (1985), no. 2, 374–382.

[FOL] *Free On-Line Dictionary Of Computing (FOLDOC)*, `http://www.foldoc.org/foldoc/index.html`.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game*, Proceedings of the 19th Symposium on Theory of Computing (STOC), ACM, 1987, pp. 218–229.

[GN98] C. Gregorio-Rodrígez and M. Núñez, *Denotational semantics for probabilistic refusal testing*, Proceedings ProbMIV 98, Electronic Notes in Theoretical Computer Science, vol. 22, 1998.

[Gol01] O. Goldreich, *Foundations of cryptography: Basic tools*, vol. I, Cambridge University Press, 2001.

[Gol04] ———, *Foundations of cryptography: Basic applications*, vol. II, Cambridge University Press, 2004.

[Hav98] B.R. Haverkort, *Performance of computer communication systems*, John Wiley & Sons, Ltd, 1998.

[Hit02] C. Hitchcock, *Probabilistic causation*, The Stanford Encyclopedia of Philosophy Fall 2002, 2002, `http://plato.stanford.edu/archives/fall2002/entries/causation-probabilistic/`.

[HSP83] S. Hart, M. Sharir, and A. Pnueli, *Termination of probabilistic concurrent programs*, ACM Transactions on Programming Languages **5** (1983), 356–380.

[HW90] M.P. Herlihy and J.M. Wing, *Linearizability: a correctness condition for concurrent objects*, ACM TOPLAS **12** (1990), no. 3, 463–492.

[IS92] A. Israeli and A. Shaham, *Optimal multi-writer multi-reader atomic register*, Proceedings of the 11th Annual ACM Symposium on Principles of Distributed Computing, 1992, pp. 71–82.

[JL91] B. Jonsson and K.G. Larsen, *Specification and refinement of probabilistic processes*, Proceedings of the 6th IEEE Symposium on Logic in Computer Science, 1991, pp. 266–277.

[JLY01] B. Jonsson, K.G. Larsen, and W. Yi, *Handbook of process algebras*, ch. Probabilistic extensions of process algebras, Elsevier, 2001.

[JY02] B. Jonsson and W. Yi, *Testing preorders for probabilistic processes can be characterized by simulations*, Theoretical Computer Science **282** (2002), no. 1, 33–51.

[KF70] A.N. Kolmogorov and S.V. Fomin, *Introductory real analysis*, Dover Publications, Inc., New York, 1970.

[KLA98] L.P. Kaelbling, M.L. Littman, and A.R.Cassandra, *Planning and acting in partially observable stochastic domains*, Artificial Intelligence **101** (1998), 99–134.

[KN02] M. Kwiatkowska and G. Norman, *Verifying randomized Byzantine agreement*, Proc. Formal Techniques for Networked and Distributed Systems (FORTE'02), LNCS, vol. 2529, 2002, pp. 194–209.

[KNS01] M. Kwiatkowska, G. Norman, and R. Segala, *Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM*, Proceedings CAV'01, LNCS, vol. 2102, 2001, pp. 194–206.

[KS76] J.G. Kennedy and J.L. Snell, *Finite Markov chains*, Springer-Verlag, New York, 1976.

[LMMS98] P. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov, *A probabilistic poly-time framework for protocol analysis*, ACM Conference on Computer and Communications Security, 1998, pp. 112–121.

[LS91] K.G. Larsen and A. Skou, *Bisimulation through probabilistic testing*, Information and Computation **91** (1991), 1–28.

[LSS94] N.A. Lynch, I. Saias, and R. Segala, *Proving time bounds for randomized distributed algorithms*, Proceedings of the 13th Annual ACM Symposium on the Principles of Distributed Computing, 1994, pp. 314–323.

[LSV03] N.A. Lynch, R. Segala, and F.W. Vaandrager, *Compositionality for probabilistic automata*, Proceedings 14th International Conference on Concurrency Theory (CONCUR 2003) (R. Amadio and D. Lugiez, eds.), Lecture Notes in Computer Science, vol. 2761, Springer-Verlag, 2003, pp. 208–221.

[LT89] N.A. Lynch and M.R. Tuttle, *An introduction to input/output automata*, CWI Quarterly **2** (1989), no. 3, 219–246.

[Lyn96] N. Lynch, *Distributed algorithms*, Morgan Kaufmann Publishers, Inc., 1996.

[MMS03] P. Mateus, J.C. Mitchell, and A. Scedrov, *Composition of cryptographic protocols in a probabilistic polynomial-time process calculus*, Proceedings CONCUR 2003, 2003, pp. 323–345.

[NH84] R.De Nicola and M. Hennessy, *Testing equivalences for processes*, Theoretical Computer Science **34** (1984), 83–133.

[PRI] *PRISM web site*, `http://www.cs.bham.ac.uk/~dxp/prism`.

[PSL00] A. Pogosyants, R. Segala, and N.A. Lynch, *Verification of the randomized consensus algorithm of Aspnes and Herlihy: a case study.*, Distributed Computing **13** (2000), no. 3, 155–186.

[Put94] M.L. Puterman, *Markov decision process – discrete stochastic dynamic programming*, John Wiley & Sons, Inc., New York, NY, 1994.

[PW00] B. Pfitzman and M. Waidner, *Composition and integrity preservation of secure reactive systems*, Proceedings CCS 2000, 2000.

[PW01] ———, *A model for asynchronous reactive systems and its application to secure message transmission*, Proceedings of the IEEE Symposium on Research in Security and Privacy, 2001, pp. 184–200.

[PZ86] A. Pnueli and L. Zuck, *Verification of multiprocess probabilistic protocols*, Distributed Computing **1** (1986), no. 1, 53–72.

[PZ93] ———, *Probabilistic verification*, Information and Computation **103** (1993), 1–29.

[Rab81] M. Rabin, *How to exchange secrets by oblivious transfer*, Tech. report, Aiken Computation Laboratory, Harvard University, 1981.

[Rab82] ———, *The choice coordination problem*, Acta Informatica **17** (1982), 121–134.

[RMST04] A. Ramanathan, J.C. Mitchell, A. Scedrov, and V. Teague, *Probabilistic bisimulation and equivalence for security analysis of network protocols*, Proceedings FOSSACS 2004, 2004.

[Rud87] W. Rudin, *Real and complex analysis*, McGraw-Hill, Inc., Boston, 1987.

[SAGG+93] J. Søgaard-Andersen, S. Garland, J. Guttag, N. Lynch, and A. Pogosyants, *Computer-assisted simulation proofs*, Proceedings of the 4th Conference on Computer Aided Verification, CAV'93, 1993, pp. 305–319.

[SdV04] A. Sokolova and E.P. de Vink, *Probabilistic automata: system types, parallel composition and comparison*, Validation of Stochastic Systems, Lecture Notes in Computer Science, vol. 2925, Springer-Verlag, 2004, pp. 1–43.

[Seg95a] R. Segala, *Compositional trace-based semantics for probabilistic automata*, Proceedings CONCUR 95, Lecture Notes in Computer Science, vol. 962, 1995, pp. 234–248.

[Seg95b] ———, *Modeling and verification of randomized distributed real-time systems*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1995, Available as Technical Report MIT/LCS/TR-676.

[Seg96] ———, *Testing probabilistic automata*, Proceedings CONCUR 96, Lecture Notes in Computer Science, vol. 1119, 1996, pp. 299–314.

[SL95] R. Segala and N.A. Lynch, *Probabilistic simulations for probabilistic processes*, Nordic Journal of Computing **2** (1995), no. 2, 250–273.

[SM03] A. Sabelfeld and A.C. Myers, *Language-based information-flow security*, IEEE Journal on Selected Areas in Communications **21** (2003), no. 1, 5–19.

[Ste94] W.J. Stewart, *Introduction to the numerical solutions of Markov chains*, Princeton University Press, 1994.

[Sto02a] M.I.A. Stoelinga, *Alea jacta est: verification of probabilistic, real-time and parametric systems*, Ph.D. thesis, Department of Computer Science, University of Nijmegen, April 2002.

[Sto02b] ———, *An introduction to probabilistic automata*, Bulletin of the European Association for Theoretical Computer Science **78** (2002), 176–198.

[SV99] M.I.A. Stoelinga and F.W. Vaandrager, *Root contention in IEEE 1394*, Proceedings 5th International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems (J.-P. Katoen, ed.), Lecture Notes in Computer Science, vol. 1601, Springer-Verlag, 1999, pp. 53–74.

[SV03] ———, *A testing scenario for probabilistic automata*, Proceedings 30 ICALP, Lecture Notes in Computer Science, vol. 2719, Springer-Verlag, 2003, pp. 407–418.

[SVA04] K. Sen, M. Viswanathan, and G. Agha, *Statistical model checking of black-box probabilistic systems*, Computer-Aided Verification, LNCS, vol. 3114, 2004, pp. 202–215.

[Tri02] K.S. Trivedi, *Probability and statistics with reliability, queuing and computer science applications*, John Wiley & Sons, Inc., New York, 2002.

[Var85] M. Vardi, *Automatic verification of probabilistic concurrent finite-state programs*, Proceedings FOCS 85, 1985, pp. 327–338.

[Vat01] F. Vatan, *Distribution functions of probabilistic automata*, Proceedings STOC 01, 2001, pp. 684–693.

[vG01] R.J. van Glabbeek, *The linear time - branching time spectrum I. The semantics of concrete, sequential processes*, pp. 3–99, North-Holland, 2001.

[vGSS95] R.J. van Glabbeek, S.A. Smolka, and B. Steffen, *Reactive, generative, and stratified models of probabilistic processes*, Information and Computation **121** (1995), 59–80.

[Wik05] *Formal methods*, Wikipedia, November 2005, `http://en.wikipedia.org/w/index.php?title=Formal_methods&oldid=27592510`.

[WSS94] S.-H. Wu, S.A. Smolka, and E.W. Stark, *Composition and behaviors of probabilistic I/O automata*, Proceedings CONCUR 94 (B. Jonsson and J. Parrow, eds.), Lecture Notes in Computer Science, vol. 836, Springer-Verlag, 1994, pp. 513–528.

[YKNP04] H.L.S. Younes, M.Z. Kwiatkowska, G. Norman, and D. Parker, *Numerical vs. statistical probabilistic model checking: an empirical study*, Tools and Algorithms for the Construction and Analysis of Systems, Lecutre Notes in Computer Science, vol. 2988, 2004, pp. 46–60.

[YL92] W. Yi and K.G. Larsen, *Testing probabilistic and nondeterministic processes*, Testing and Verification **12** (1992), 47–61, Protocol Specification.

[You05] H.L.S. Younes, *Probabilistic verification for "black-box" systems*, Proceedings CAV 2005, 2005, pp. 253–265.

[YS02] H.L.S. Younes and R.G. Simmons, *Probabilistic verification of discrete event systems using acceptance sampling*, Computer-Aided Verification, Lecutre Notes in Computer Science, vol. 2404, 2002, pp. 223–235.

Samenvatting (*Dutch Summary*)

Dit proefschrift gaat over probabilistische verificatie. Het primaire doel was om modelleerraamwerken te ontwikkelen die kunnen worden gebruikt voor zowel de specificatie als de verificatie van gedistribueerde algorithmen die gebruik maken van randomisatie (kansen). De nadruk ligt op de semantische aspecten van zulke raamwerken; wij pogen bijvoorbeeld om een precieze wiskundige semantiek te geven aan de processen die definiëerbaar zijn in deze raamwerken, en wij bewijzen standaard stellingen die ons in staat stellen om over semantische objecten te redeneren en om deze te manipuleren.

In dit proefschrift worden kansen expliciet geïntroduceerd door de deelnemende partijen. Processen brengen bijvoorbeeld stemmen uit (iedere stem heeft een bepaalde kans) om consensus te bereiken, of ze kiezen met een bepaalde kans een buurman om informatie door te geven zonder het netwerk te overbelasten. Omdat de omgeving zeer onvoorspelbaar is worden nondeterministische keuzes gebruikt om onzekerheden met betrekking tot communicatie te modelleren. Kortom, onze modellen kunnen typisch zowel nondeterministische als probabilistische keuzes bevatten.

De aanwezigheid van nondeterminisme is gewenst vanuit het modelleerperspectief, maar leidt tot complicaties bij de semantische definities en analyse. Om namelijk goed gedefiniëerde kansdistributies te verkrijgen uit een specificatie die zowel nondeterministische als probabilistische keuzes bevat moeten deze twee "uit de knoop" worden gehaald. Dit wordt gewoonlijk gedaan door zogenaamde *tegenstanders/schedulers* die alle nondeterministische keuzes oplossen.

Wij zijn van mening dat het oplossen van de nondeterministische keuzes (dit wordt ook "scheduling" genoemd) een fundamenteel punt is in semantische studies omdat verschillende scheduling mechanismes ook verschillende kansdistributies aan een specificatie koppelen. Dit impliceert dat de eigenschappen van een specificatie af kunnen hangen van de manier waarop de nondeterministische keuzes worden opgelost. Daarom voelen wij de drang om meer aandacht aan de notie van tegenstanders te besteden.

In het bijzonder bestuderen wij mathematische eigenschappen van tegenstanders welke geformaliseerd zijn als functies die executiehistories aan beschikbare transities koppelen. Bovendien proberen wij te begrijpen hoe de verschillende definities van parallelle compositie zich vertalen in verschillende aannames over het gedrag van de tegenstanders. Dit geeft ons de mogelijkheid om een aantal

sleuteleigenschappen van tegenstanders te benoemen (zoals afhankelijkheid van historie) die invloed hebben op de compositionaliteit van trace-stijl semantiek. Als laatste proberen wij een link te leggen tussen de noties van tegenstanders zoals die in onze formele definities bevat zijn en degenen die worden gebruikt binnen gedistribueerde systemen zoals in de gebieden van security protocollen en gerandomiseerde consensus algorithmes.

Dit proefschrift bestaat uit drie delen. In Part I werken wij met Segala's (simple) *Probabilistic Automata (PA)* model [Seg95b] en bewijzen vele technische stellingen over tegenstanders en hun geïnduceerde kansdistributies. Deze resultaten zijn vervolgens gebruikt om de test-semantiek van Stoelinga en Vaandrager [SV03] uit te breiden. In Part II introduceren wij een variant van *Probabilistic Input/Output Automata (PIOA)* en gebruiken deze als basis voor twee gespecialiseerde modellen die beiden een compositionale trace-stijl semantiek hebben. Als laatste wordt in Part III een consensus algorithme gepresenteerd dat gebruik maakt van kansen. Het algorithme wordt handmatig correct bewezen en deels mechanisch geanalyseerd met de PRISM model checker [PRI].

(Special thanks to Martijn Hendriks, who translated this summary into the Dutch language.)

Curriculum Vitae

Ling Cheung was born on June 12, 1978, in Canton, China.

In year 2000, she received a Bachelor of Science degree in Civil and Environmental Engineering and a Bachelor of Science degree in Mathematics, both from Carnegie Mellon University in Pennsylvania, the United States.

In year 2001, she received a Master of Science degree in Mathematics, also from Carnegie Mellon University.

From September 2002 to August 2003, she was a PhD student in the Foundations group at the Institute for Computing and Information, Radboud University Nijmegen, the Netherlands. She was under the supervision of Prof. Henk Barendregt. In September 2003, she joined the Informatics for Technical Applications group, led by Prof. Frits Vaandrager.

Starting September 2006, she will be a postdoctoral researcher in the Theory of Computing group at the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, the United States.

Titles in the IPA Dissertation Series

J.O. Blanco. *The State Operator in Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1996-01

A.M. Geerling. *Transformational Development of Data-Parallel Algorithms*. Faculty of Mathematics and Computer Science, KUN. 1996-02

P.M. Achten. *Interactive Functional Programs: Models, Methods, and Implementation*. Faculty of Mathematics and Computer Science, KUN. 1996-03

M.G.A. Verhoeven. *Parallel Local Search*. Faculty of Mathematics and Computing Science, TUE. 1996-04

M.H.G.K. Kesseler. *The Implementation of Functional Languages on Parallel Machines with Distrib. Memory*. Faculty of Mathematics and Computer Science, KUN. 1996-05

D. Alstein. *Distributed Algorithms for Hard Real-Time Systems*. Faculty of Mathematics and Computing Science, TUE. 1996-06

J.H. Hoepman. *Communication, Synchronization, and Fault-Tolerance*. Faculty of Mathematics and Computer Science, UvA. 1996-07

H. Doornbos. *Reductivity Arguments and Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1996-08

D. Turi. *Functorial Operational Semantics and its Denotational Dual*. Faculty of Mathematics and Computer Science, VUA. 1996-09

A.M.G. Peeters. *Single-Rail Handshake Circuits*. Faculty of Mathematics and Computing Science, TUE. 1996-10

N.W.A. Arends. *A Systems Engineering Specification Formalism*. Faculty of Mechanical Engineering, TUE. 1996-11

P. Severi de Santiago. *Normalisation in Lambda Calculus and its Relation to Type Inference*. Faculty of Mathematics and Computing Science, TUE. 1996-12

D.R. Dams. *Abstract Interpretation and Partition Refinement for Model Checking*. Faculty of Mathematics and Computing Science, TUE. 1996-13

M.M. Bonsangue. *Topological Dualities in Semantics*. Faculty of Mathematics and Computer Science, VUA. 1996-14

B.L.E. de Fluiter. *Algorithms for Graphs of Small Treewidth*. Faculty of Mathematics and Computer Science, UU. 1997-01

W.T.M. Kars. *Process-algebraic Transformations in Context*. Faculty of Computer Science, UT. 1997-02

P.F. Hoogendijk. *A Generic Theory of Data Types*. Faculty of Mathematics and Computing Science, TUE. 1997-03

T.D.L. Laan. *The Evolution of Type Theory in Logic and Mathematics*. Faculty of Mathematics and Computing Science, TUE. 1997-04

C.J. Bloo. *Preservation of Termination for Explicit Substitution*. Faculty of Mathematics and Computing Science, TUE. 1997-05

J.J. Vereijken. *Discrete-Time Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1997-06

F.A.M. van den Beuken. *A Functional Approach to Syntax and Typing*. Faculty of Mathematics and Informatics, KUN. 1997-07

A.W. Heerink. *Ins and Outs in Refusal Testing*. Faculty of Computer Science, UT. 1998-01

G. Naumoski and W. Alberts. *A Discrete-Event Simulator for Systems Engineering*. Faculty of Mechanical Engineering, TUE. 1998-02

J. Verriet. *Scheduling with Communication for Multiprocessor Computation*. Faculty of Mathematics and Computer Science, UU. 1998-03

J.S.H. van Gageldonk. *An Asynchronous Low-Power 80C51 Microcontroller*. Faculty of Mathematics and Computing Science, TUE. 1998-04

A.A. Basten. *In Terms of Nets: System Design with Petri Nets and Process Algebra*. Faculty of Mathematics and Computing Science, TUE. 1998-05

E. Voermans. *Inductive Datatypes with Laws and Subtyping – A Relational Model*. Faculty of Mathematics and Computing Science, TUE. 1999-01

H. ter Doest. *Towards Probabilistic Unification-based Parsing*. Faculty of Computer Science, UT. 1999-02

J.P.L. Segers. *Algorithms for the Simulation of Surface Processes*. Faculty of Mathematics and Computing Science, TUE. 1999-03

C.H.M. van Kemenade. *Recombinative Evolutionary Search*. Faculty of Mathematics and Natural Sciences, UL. 1999-04

E.I. Barakova. *Learning Reliability: a Study on Indecisiveness in Sample Selection*. Faculty of Mathematics and Natural Sciences, RUG. 1999-05

M.P. Bodlaender. *Scheduler Optimization in Real-Time Distributed Databases*. Faculty of Mathematics and Computing Science, TUE. 1999-06

M.A. Reniers. *Message Sequence Chart: Syntax and Semantics*. Faculty of Mathematics and Computing Science, TUE. 1999-07

J.P. Warners. *Nonlinear approaches to satisfiability problems*. Faculty of Mathematics and Computing Science, TUE. 1999-08

J.M.T. Romijn. *Analysing Industrial Protocols with Formal Methods*. Faculty of Computer Science, UT. 1999-09

P.R. D'Argenio. *Algebras and Automata for Timed and Stochastic Systems*. Faculty of Computer Science, UT. 1999-10

G. Fábián. *A Language and Simulator for Hybrid Systems*. Faculty of Mechanical Engineering, TUE. 1999-11

J. Zwanenburg. *Object-Oriented Concepts and Proof Rules*. Faculty of Mathematics and Computing Science, TUE. 1999-12

R.S. Venema. *Aspects of an Integrated Neural Prediction System*. Faculty of Mathematics and Natural Sciences, RUG. 1999-13

J. Saraiva. *A Purely Functional Implementation of Attribute Grammars*. Faculty of Mathematics and Computer Science, UU. 1999-14

R. Schiefer. *Viper, A Visualisation Tool for Parallel Program Construction*. Faculty of Mathematics and Computing Science, TUE. 1999-15

K.M.M. de Leeuw. *Cryptology and Statecraft in the Dutch Republic*. Faculty of Mathematics and Computer Science, UvA. 2000-01

T.E.J. Vos. *UNITY in Diversity. A stratified approach to the verification of distributed algorithms*. Faculty of Mathematics and Computer Science, UU. 2000-02

W. Mallon. *Theories and Tools for the Design of Delay-Insensitive Communicating Processes*. Faculty of Mathematics and Natural Sciences, RUG. 2000-03

W.O.D. Griffioen. *Studies in Computer Aided Verification of Protocols*. Faculty of Science, KUN. 2000-04

P.H.F.M. Verhoeven. *The Design of the MathSpad Editor*. Faculty of Mathematics and Computing Science, TUE. 2000-05

J. Fey. *Design of a Fruit Juice Blending and Packaging Plant*. Faculty of Mechanical Engineering, TUE. 2000-06

M. Franssen. *Cocktail: A Tool for Deriving Correct Programs*. Faculty of Mathematics and Computing Science, TUE. 2000-07

P.A. Olivier. *A Framework for Debugging Heterogeneous Applications*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2000-08

E. Saaman. *Another Formal Specification Language*. Faculty of Mathematics and Natural Sciences, RUG. 2000-10

M. Jelasity. *The Shape of Evolutionary Search Discovering and Representing Search Space Structure*. Faculty of Mathematics and Natural Sciences, UL. 2001-01

R. Ahn. *Agents, Objects and Events a computational approach to knowledge, observation and communication*. Faculty of Mathematics and Computing Science, TU/e. 2001-02

M. Huisman. *Reasoning about Java programs in higher order logic using PVS and Isabelle*. Faculty of Science, KUN. 2001-03

I.M.M.J. Reymen. *Improving Design Processes through Structured Reflection*. Faculty of Mathematics and Computing Science, TU/e. 2001-04

S.C.C. Blom. *Term Graph Rewriting: syntax and semantics*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2001-05

R. van Liere. *Studies in Interactive Visualization*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2001-06

A.G. Engels. *Languages for Analysis and Testing of Event Sequences*. Faculty of Mathematics and Computing Science, TU/e. 2001-07

J. Hage. *Structural Aspects of Switching Classes*. Faculty of Mathematics and Natural Sciences, UL. 2001-08

M.H. Lamers. *Neural Networks for Analysis of Data in Environmental Epidemiology: A Case-study into Acute Effects of Air Pollution Episodes*. Faculty of Mathematics and Natural Sciences, UL. 2001-09

T.C. Ruys. *Towards Effective Model Checking*. Faculty of Computer Science, UT. 2001-10

D. Chkliaev. *Mechanical verification of concurrency control and recovery protocols*. Faculty of Mathematics and Computing Science, TU/e. 2001-11

M.D. Oostdijk. *Generation and presentation of formal mathematical documents*. Faculty of Mathematics and Computing Science, TU/e. 2001-12

A.T. Hofkamp. *Reactive machine control: A simulation approach using χ*. Faculty of Mechanical Engineering, TU/e. 2001-13

D. Bošnački. *Enhancing state space reduction techniques for model checking*. Faculty of Mathematics and Computing Science, TU/e. 2001-14

M.C. van Wezel. *Neural Networks for Intelligent Data Analysis: theoretical and experimental aspects*. Faculty of Mathematics and Natural Sciences, UL. 2002-01

V. Bos and J.J.T. Kleijn. *Formal Specification and Analysis of Industrial Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2002-02

T. Kuipers. *Techniques for Understanding Legacy Software Systems*. Faculty of Natural Sciences, Mathematics and Computer Science, UvA. 2002-03

S.P. Luttik. *Choice Quantification in Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-04

R.J. Willemen. *School Timetable Construction: Algorithms and Complexity*. Faculty of Mathematics and Computer Science, TU/e. 2002-05

M.I.A. Stoelinga. *Alea Jacta Est: Verification of Probabilistic, Real-time and Parametric Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-06

N. van Vugt. *Models of Molecular Computing*. Faculty of Mathematics and Natural Sciences, UL. 2002-07

A. Fehnker. *Citius, Vilius, Melius: Guiding and Cost-Optimality in Model Checking of Timed and Hybrid Systems*. Faculty of Science, Mathematics and Computer Science, KUN. 2002-08

R. van Stee. *On-line Scheduling and Bin Packing*. Faculty of Mathematics and Natural Sciences, UL. 2002-09

D. Tauritz. *Adaptive Information Filtering: Concepts and Algorithms*. Faculty of Mathematics and Natural Sciences, UL. 2002-10

M.B. van der Zwaag. *Models and Logics for Process Algebra*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-11

J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2002-12

L. Moonen. *Exploring Software Systems*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2002-13

J.I. van Hemert. *Applying Evolutionary Computation to Constraint Satisfaction and Data Mining*. Faculty of Mathematics and Natural Sciences, UL. 2002-14

S. Andova. *Probabilistic Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2002-15

Y.S. Usenko. *Linearization in μCRL*. Faculty of Mathematics and Computer Science, TU/e. 2002-16

J.J.D. Aerts. *Random Redundant Storage for Video on Demand*. Faculty of Mathematics and Computer Science, TU/e. 2003-01

M. de Jonge. *To Reuse or To Be Reused: Techniques for component composition and construction*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-02

J.M.W. Visser. *Generic Traversal over Typed Source Code Representations*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2003-03

S.M. Bohte. *Spiking Neural Networks*. Faculty of Mathematics and Natural Sciences, UL. 2003-04

T.A.C. Willemse. *Semantics and Verification in Process Algebras with Data and Timing*. Faculty of Mathematics and Computer Science, TU/e. 2003-05

S.V. Nedea. *Analysis and Simulations of Catalytic Reactions*. Faculty of Mathematics and Computer Science, TU/e. 2003-06

M.E.M. Lijding. *Real-time Scheduling of Tertiary Storage*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-07

H.P. Benz. *Casual Multimedia Process Annotation – CoMPAs*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-08

D. Distefano. *On Modelchecking the Dynamics of Object-based Software: a Foundational Approach*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2003-09

M.H. ter Beek. *Team Automata – A Formal Approach to the Modeling of Collaboration Between System Components*. Faculty of Mathematics and Natural Sciences, UL. 2003-10

D.J.P. Leijen. *The λ Abroad – A Functional Approach to Software Components*. Faculty of Mathematics and Computer Science, UU. 2003-11

W.P.A.J. Michiels. *Performance Ratios for the Differencing Method*. Faculty of Mathematics and Computer Science, TU/e. 2004-01

G.I. Jojgov. *Incomplete Proofs and Terms and Their Use in Interactive Theorem Proving*. Faculty of Mathematics and Computer Science, TU/e. 2004-02

P. Frisco. *Theory of Molecular Computing – Splicing and Membrane systems*. Faculty of Mathematics and Natural Sciences, UL. 2004-03

S. Maneth. *Models of Tree Translation*. Faculty of Mathematics and Natural Sciences, UL. 2004-04

Y. Qian. *Data Synchronization and Browsing for Home Environments*. Faculty of

Mathematics and Computer Science and Faculty of Industrial Design, TU/e. 2004-05

F. Bartels. *On Generalised Coinduction and Probabilistic Specification Formats*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-06

L. Cruz-Filipe. *Constructive Real Analysis: a Type-Theoretical Formalization and Applications*. Faculty of Science, Mathematics and Computer Science, KUN. 2004-07

E.H. Gerding. *Autonomous Agents in Bargaining Games: An Evolutionary Investigation of Fundamentals, Strategies, and Business Applications*. Faculty of Technology Management, TU/e. 2004-08

N. Goga. *Control and Selection Techniques for the Automated Testing of Reactive Systems*. Faculty of Mathematics and Computer Science, TU/e. 2004-09

M. Niqui. *Formalising Exact Arithmetic: Representations, Algorithms and Proofs*. Faculty of Science, Mathematics and Computer Science, RU. 2004-10

A. Löh. *Exploring Generic Haskell*. Faculty of Mathematics and Computer Science, UU. 2004-11

I.C.M. Flinsenberg. *Route Planning Algorithms for Car Navigation*. Faculty of Mathematics and Computer Science, TU/e. 2004-12

R.J. Bril. *Real-time Scheduling for Media Processing Using Conditionally Guaranteed Budgets*. Faculty of Mathematics and Computer Science, TU/e. 2004-13

J. Pang. *Formal Verification of Distributed Systems*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-14

F. Alkemade. *Evolutionary Agent-Based Economics*. Faculty of Technology Management, TU/e. 2004-15

E.O. Dijk. *Indoor Ultrasonic Position Estimation Using a Single Base Station*. Faculty of Mathematics and Computer Science, TU/e. 2004-16

S.M. Orzan. *On Distributed Verification and Verified Distribution*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2004-17

M.M. Schrage. *Proxima - A Presentation-oriented Editor for Structured Documents*. Faculty of Mathematics and Computer Science, UU. 2004-18

E. Eskenazi and A. Fyukov. *Quantitative Prediction of Quality Attributes for Component-Based Software Architectures*. Faculty of Mathematics and Computer Science, TU/e. 2004-19

P.J.L. Cuijpers. *Hybrid Process Algebra*. Faculty of Mathematics and Computer Science, TU/e. 2004-20

N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. Faculty of Mechanical Engineering, TU/e. 2004-21

E. Ábrahám. *An Assertional Proof System for Multithreaded Java -Theory and Tool Support-* . Faculty of Mathematics and Natural Sciences, UL. 2005-01

R. Ruimerman. *Modeling and Remodeling in Bone Tissue*. Faculty of Biomedical Engineering, TU/e. 2005-02

C.N. Chong. *Experiments in Rights Control - Expression and Enforcement*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-03

H. Gao. *Design and Verification of Lock-free Parallel Algorithms*. Faculty of Mathematics and Computing Sciences, RUG. 2005-04

H.M.A. van Beek. *Specification and Analysis of Internet Applications*. Faculty of Mathematics and Computer Science, TU/e. 2005-05

M.T. Ionita. *Scenario-Based System Architecting - A Systematic Approach to Developing Future-Proof System Architectures*. Faculty of Mathematics and Computing Sciences, TU/e. 2005-06

G. Lenzini. *Integration of Analysis Techniques in Security and Fault-Tolerance*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-07

I. Kurtev. *Adaptability of Model Transformations*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-08

T. Wolle. *Computational Aspects of Treewidth - Lower Bounds and Network Reliability*. Faculty of Science, UU. 2005-09

O. Tveretina. *Decision Procedures for Equality Logic with Uninterpreted Functions*. Faculty of Mathematics and Computer Science, TU/e. 2005-10

A.M.L. Liekens. *Evolution of Finite Populations in Dynamic Environments*. Faculty of Biomedical Engineering, TU/e. 2005-11

J. Eggermont. *Data Mining using Genetic Programming: Classification and Symbolic Regression*. Faculty of Mathematics and Natural Sciences, UL. 2005-12

B.J. Heeren. *Top Quality Type Error Messages*. Faculty of Science, UU. 2005-13

G.F. Frehse. *Compositional Verification of Hybrid Systems using Simulation Relations*. Faculty of Science, Mathematics and Computer Science, RU. 2005-14

M.R. Mousavi. *Structuring Structural Operational Semantics*. Faculty of Mathematics and Computer Science, TU/e. 2005-15

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Faculty of Mathematics and Computer Science, TU/e. 2005-16

T. Gelsema. *Effective Models for the Structure of pi-Calculus Processes with Replication*. Faculty of Mathematics and Natural Sciences, UL. 2005-17

P. Zoeteweij. *Composing Constraint Solvers*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-18

J.J. Vinju. *Analysis and Transformation of Source Code by Parsing and Rewriting*. Faculty of Natural Sciences, Mathematics, and Computer Science, UvA. 2005-19

M.Valero Espada. *Modal Abstraction and Replication of Processes with Data*. Faculty of Sciences, Division of Mathematics and Computer Science, VUA. 2005-20

A. Dijkstra. *Stepping through Haskell*. Faculty of Science, UU. 2005-21

Y.W. Law. *Key management and link-layer security of wireless sensor networks: energy-efficient attack and defense*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2005-22

E. Dolstra. *The Purely Functional Software Deployment Model*. Faculty of Science, UU. 2006-01

R.J. Corin. *Analysis Models for Security Protocols*. Faculty of Electrical Engineering, Mathematics & Computer Science, UT. 2006-02

P.R.A. Verbaan. *The Computational Complexity of Evolving Systems*. Faculty of Science, UU. 2006-03

K.L. Man and R.R.H. Schiffelers. *Formal Specification and Analysis of Hybrid Systems*. Faculty of Mathematics and Computer Science and Faculty of Mechanical Engineering, TU/e. 2006-04

M. Kyas. *Verifying OCL Specifications of UML Models: Tool Support and Compositionality*. Faculty of Mathematics and Natural Sciences, UL. 2006-05

M. Hendriks. *Model Checking Timed Automata - Techniques and Applications*. Faculty of Science, Mathematics and Computer Science, RU. 2006-06

J. Ketema. *Böhm-Like Trees for Rewriting*. Faculty of Sciences, VUA. 2006-07

C.-B. Breunesse. *On JML: topics in tool-assisted verification of JML programs*. Faculty of Science, Mathematics and Computer Science, RU. 2006-08

B. Markvoort. *Towards Hybrid Molecular Simulations*. Faculty of Biomedical Engineering, TU/e. 2006-09

S.G.R. Nijssen. *Mining Structured Data*. Faculty of Mathematics and Natural Sciences, UL. 2006-10

G. Russello. *Separation and Adaptation of Concerns in a Shared Data Space*. Faculty of Mathematics and Computer Science, TU/e. 2006-11

L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. Faculty of Science, Mathematics and Computer Science, RU. 2006-12