Switched PIOA: Parallel Composition via Distributed Scheduling

Ling Cheung [a,*,1]

[a] *Institute for Computing and Information Sciences,*
Radboud University Nijmegen,
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands

Nancy Lynch [b,2]

[b] *MIT Computer Science and Artificial Intelligence Laboratory,*
Cambridge, MA 02139, USA

Roberto Segala [c,3]

[c] *Dipartimento di Informatica, Università di Verona,*
Strada Le Grazie 15, 37134 Verona, Italy

Frits Vaandrager [a,1]

Abstract

This paper presents the framework of switched probabilistic input/output automata (or switched PIOA), 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 unrealistic schedules that arise under the more common centralized scheduling scheme. As a result, we are able to prove that our trace-style semantics is compositional.

Key words: Switched Probabilistic Input/Output Automata, Trace-style Semantics, Parallel Composition, Distributed Scheduling, Compositionality

1 Introduction

Over the past few decades, a large number of modeling frameworks have been adopted for the purpose of verifying and analyzing stochastic systems. Some of these frameworks, for example, *continuous-time Markov chains* [Ste94] and *labeled Markov processes* [DEP02], are designed to handle continuous probability distributions, thus finding many applications in the area of performance and reliability analysis [Hav98]. Others, such as *discrete-time Markov chains* [KS76] and *probabilistic automata* [Seg95], deal with discrete probability distributions and are popular in the verification of distributed algorithms and communication protocols [Agg94,LSS94,PSL00,SV99].

Designers of such frameworks are often presented with two challenges:

(i) defining a sensible notion of *parallel composition*;
(ii) defining a sensible notion of *semantic equivalence (or preorder)* that is compositional with respect to the proposed parallel operator.

Both notions are important tools for verification. Parallel composition underlies the so-called modular approaches to system development and analysis, where large and complex systems are decomposed into smaller and more tangible subsystems. Semantic equivalence, on the other hand, allows us to move across different levels of abstraction, from high-level abstract specifications to low-level detailed implementations. A successful combination of parallel composition and process semantics provides great flexibility in model construction and correctness analysis, thus increasing the appeal of the particular modeling framework.

In this paper, we focus on systems that exhibit both nondeterministic behavior and stochastic behavior, while the latter is restricted to discrete probability distributions. Our goal is to develop a compositional framework for modeling these systems. In particular, we are interested in compositionality of trace-style semantics, which has proven to be a surprisingly difficult problem. Resorting to distributed scheduling among parallel components, we offer a solution quite unlike most existing proposals.

[⋆] Preliminary versions of this paper appeared as [CLSV04a] and [CLSV04b].

[∗] Corresponding author.
Email address: `lcheung@cs.ru.nl` (Ling Cheung).

[1] Supported by DFG/NWO bilateral cooperation project 600.050.011.01 Validation of Stochastic Systems.

[2] Supported by DARPA/AFOSR MURI Award #F49620-02-1-0325, NSF Award #CCR-0326277, and USAF, AFRL Award #FA9550-04-1-0121.

[3] Supported by MURST project Constraint-based Verification of Reactive Systems.

A good part of this introduction is devoted to discussions of related literature (Sections 1.1 and 1.2). A disinterested reader may wish to begin with Section 1.3, where we illustrate via a simple example the difficulty with trace-style semantics.

1.1 Process Semantics

We focus on systems with both nondeterministic and probabilistic choices, as opposed to purely probabilistic systems. This is because nondeterminism is essential in modeling lack of information in either the object system or the external environment. Moreover, as we shall discuss shortly, the interleaving interpretation of parallel composition relies on the presence of nondeterministic choices. Finally, nondeterministic choices can be used to model implementation freedom, making our framework more widely applicable.

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,CSV06] and many others.

Overall, semantics of a branching character, such as bisimulation and simulation, have been more common than their linear counterparts, such as trace distribution preorder [Seg95]. One likely reason is that a trace-style semantics requires one to resolve all nondeterministic choices, usually by means of the so-called *adversaries*[1]. 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.

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 probability distributions on states) 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 consensus algorithm, we may wish to calculate a lower bound for the probability of reaching agreement during the

[1] These are called *policies* in the setting of *Markov decision processes*, as studied in planning and optimization.

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.

1.2 Parallel Composition

A fundamental idea in concurrency theory is the *interleaving* interpretation of parallel composition:

(i) every atomic step of a composite system is an atomic step of one of its components (more in case of synchronization);
(ii) the scheduling among components is arbitrary, up to some appropriate fairness constraints.

In particular, the parallel composition of two independent actions is interpreted as a nondeterministic choice between the two possible interleavings of these actions. This interpretation is generally regarded as a simplifying assumption, reducing the complexity of single-step evolution.

Most existing proposals of parallel composition for stochastic processes adopt the interleaving assumption, thereby necessitating the use of (some form of) adversaries to resolve nondeterministic choices among parallel components. 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 *static* adversary, resolving the choice between two processes in the same manner at every step.

- *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. Given delay parameters of all components, one can use specific properties of exponential distributions to calculate the bias towards each component. Therefore, this approach essentially uses *state-dependent* adversaries to resolve nondeterministic choices arising from parallel composition.
- *Compose-and-schedule* [DHK98,Seg95]. 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* ones (i.e., those between parallel components).

Clearly, the last approach is the most robust, in that scheduling decisions may depend on dynamic behaviors of the entire system. Here we pay a hefty price for such expressivity: trace-style semantics is not compositional [Seg95]. Put simply, trace-style semantics abstracts away from internal branching structures of processes. Yet a powerful adversary can observe differences in internal branching and is therefore capable of exposing these differences when equivalent processes are composed in parallel with the same probabilistic context. We shall return to this point in Section 1.3 and give a concrete example (Figures 1 and 2).

Moving to the less robust approach of real-time delay, one can in fact achieve compositionality for trace-style semantics [WSS94]. However, this approach relies on the assumption that delay patterns of processes can be universally characterized by exponential distributions. In the end, it is unclear whether the theory is applicable outside specific areas such as hybrid systems and queuing networks.

Finally, we mention an approach that takes us away from the realm of interleaving semantics. 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 by simply multiplying the probabilities of all atomic moves involved. In this setting, it is also possible to obtain a compositional trace-style semantics [dAHJ01]. Nonetheless, synchronous models are not suitable for a large class of problems in which simultaneous executions are not possible. Obvious examples include mutual exclusion and distributed consensus algorithms. This takes us back to the challenge of scheduling via adversaries.

1.3 Observational Powers of Adversaries

As promised, we give a simple example in which a trace-style semantics fails to be compositional. In particular, we take the trace distribution semantics of [Seg95], where each possible behavior is a probability space on the set of traces and is induced by a history-dependent adversary.

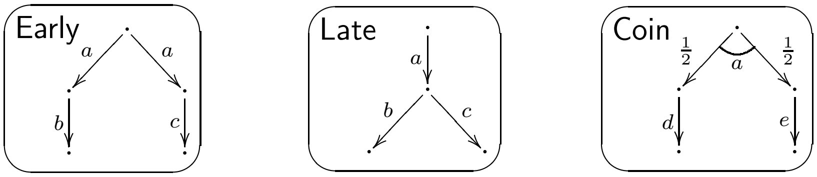

Fig. 1. Probabilistic automata Early, Late and Coin

Consider the three automata shown in Figure 1. As its name suggests, automaton Early forces the adversary to choose between b and c at the beginning of each execution, as it chooses one of the two available a-transitions. On the other hand, in automaton Late, the adversary may postpone this decision until after the a-transition. Clearly, these two automata have the same set of trace distributions, but they can be distinguished by composing with the context Coin. This context 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.

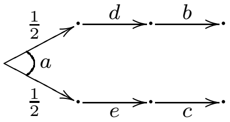

Fig. 2. Non-substitutivity of trace distribution equivalence

The composed system Late $\|$ Coin has a trace distribution that assigns probability $\frac{1}{2}$ to each of these traces: adb and aec (Figure 2). This 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 the composite Early $\|$ Coin. This is because the adversary must resolve the choice between b and c (equivalently, the choice between the two distinct a-transitions) at the beginning of each execution, before the random outcome in Coin is available. Therefore, the two composites Early || Coin and Late || Coin are not trace distribution equivalent.

Inspired by this example, we showed that the coarsest pre-congruence refining trace distribution preorder coincides with the probabilistic simulation preorder [LSV03]. In other words, the observational power given to adversaries forces us into the realm of branching-style semantics, where internal branching structures can be used to distinguish processes.

In the present paper, we follow a different direction: rather than taking the largest pre-congruence induced by trace distribution preorder, we attempt to weaken the observational power of adversaries. Notice, in the composition mechanism of probabilistic automata, nondeterministic choices are resolved after the two automata are composed, allowing the adversary to make decisions in one component using state information of the other. This sort of "information leakage" is precisely the source of difficulty in compositionality. We therefore aim at a framework in which global nondeterminism is clearly separated from local nondeterminism. (Recall that the former arises from uncertainty in a distributed environment, while the latter from uncertainty within components.) The challenge is then to achieve this separation without sacrificing the flexibility to treat a composite of multiple components as a single component.

In fact, adversary models of various strengths have been studied in the setting of randomized distributed algorithms [Asp03,AB04], where correctness and complexity of algorithms depend crucially upon the particular choice of adversary model. In the formal methods community, however, this issue has not received much attention. Some initial steps along these lines can be found in [CH05].

1.4 Distributed Scheduling

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 perform actively 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 PIOA* for short). It augments the *probabilistic input/output automata (PIOA)* model [WSS94,PSL00,BPW04] 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 nondeterministic 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 2 and allows us to give a compositional trace-style semantics (Definition 16 and Theorem 33).

Distributed scheduling (as opposed to centralized scheduling) has been a mainstream approach in the area of security analysis [BPW04,Can01], where information flow is a sensitive issue. Compared with the *interactive Turing machines* of [Can01] and *asynchronous reactive systems* of [BPW04], our framework provides much better modeling flexibility, as we allow local nondeterministic choices to account for lack of information and to allow implementation freedom. However, we must admit this is an unfair comparison, because the two frameworks mentioned above are highly specialized for delicate reasoning in computational cryptography. Distributed scheduling also arises in practical settings, for example, token ring or token bus networks. In such a network, a token is passed among network nodes and the unique node possessing the token may transmit data.

For those who may still 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 way of action synchronization and it makes scheduling decisions accordingly. Since the input signature of such an arbiter is completely flexible, we have a convenient means to specify what information is available for inter-component scheduling. This will be further discussed in Section 7.

1.5 Overview

This introduction is followed by Section 2, which contains basic mathematical preliminaries. Section 3 presents a new formulation of the PIOA framework, combining *reactive* and *generative* system types [DHK98,vGSS95,SdV04] in the presence of input/output (I/O) distinction. We also define *I/O schedulers* for PIOAs and the notion of *execution trees* induced by I/O schedulers.

Starting from Section 4, we focus on distributed scheduling. Switched PIOAs are defined as PIOAs satisfying a set of switch axioms, which formalize the idea of control exchange. A switched probabilistic system is then given by a switched PIOA and a set of I/O schedulers. Using a trace-style abstraction on execution trees, we define an external behavior semantics for switched probabilistic systems.

In Section 5, we define parallel composition for PIOAs, switched PIOAs and switched probabilistic systems. Section 6 then presents the main technical contribution of this paper: the external behavior semantics for switched probabilistic systems is compositional (Theorem 33).

Finally, Section 7 describes *controllable PIOAs* and arbiters, which can be used to implement various centralized scheduling schemes. Concluding discussions follow in Section 8.

2 Preliminaries

Let two sets X and Y be given. A function $\mu : X \to [0, 1]$ is called a *discrete probability distribution* on X if $\sum_{x \in X} \mu(x) = 1$. Similarly, μ is said to be a *discrete sub-probability distribution* if $\sum_{x \in X} \mu(x) \leq 1$. The *support* of μ, denoted $\mathsf{Supp}(\mu)$, is the set $\{x \in X \mid \mu(x) > 0\}$. We write $\mathsf{Disc}(X)$ for the set of all discrete distributions on X. Given $x \in X$, the *Dirac distribution* on x, denoted $\mathsf{Dirac}(x)$, assigns probability 1 to x. If μ is a discrete distribution on 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 $y \in Y \setminus X$.

We write $X \times Y$ for the *Cartesian product* of X and Y, where the projection maps are denoted π_L and π_R, 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 π_i. Throughout this paper, we assume that the index set $\mathcal{I}$ is non-empty and finite. Also, when $\mathcal{I}$ is clear from context, we write $\vec{x}$ for a typical member of $\prod_{i\in\mathcal{I}} X_i$.

Given a family $\{\mu_i \mid i \in \mathcal{I}\}$ 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 $\pi_i(\mu)$, by:

$$\pi_i(\mu)(s) := \sum_{\vec{t}:t_i=s} \mu(\vec{t}).$$

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

Finally, the set of all *partial functions* from X to Y is denoted $X \rightharpoonup Y$. For each $f \in X \rightharpoonup Y$, we write $\mathsf{dom}(f)$ for the *domain* of f and $f(x) = \perp$ whenever $x \notin \mathsf{dom}(f)$. The symbol $\emptyset$ denotes the empty function, as well as the empty set.

3 Probabilistic Input/Output Automata

In this section, we define the basic framework of probabilistic input/output automata, following the tradition of Input/Output Automata (IOA) of Lynch and Tuttle [LT89]. Variations of this framework have appeared in many places (e.g., [BPW04,PSL00,WSS94]), yet the actual definitions diverge significantly. Among other goals, this paper aims to provide a concise and unifying formulation.

We assume a fixed, countably infinite alphabet Act of action symbols. The set of finite (resp., infinite) sequences over Act is denoted by $\mathsf{Act}^{<\omega}$ (resp., Act^{ω}). The set of all sequences over Act is $\mathsf{Act}^{\leq\omega}$. A similar convention applies to other sets of sequences.

3.1 Reactive and Generative Transition Structures

Inspired by [vGSS95], we define reactive and generative transition structures as follows.

Definition 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 *action $a \in X$ if* $\mathbf{R}(s, a) = \emptyset$*. It 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, $\mathbf{R}(s, a)$ yields a set of discrete distributions on S. Thus we allow nondeterministic 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 3 below illustrates two such reactive systems.

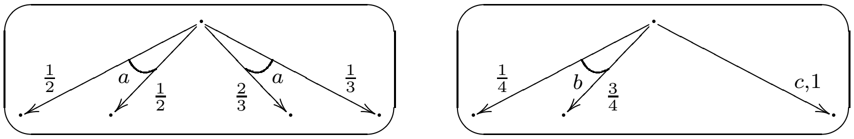

Fig. 3. Examples of Reactive Transition Systems

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 nondeterministic choices over bundles, while each bundle specifies a random choice over next transitions. We use variables f, g, etc., for these transition bundles. Figure 4 below illustrates two such generative systems.

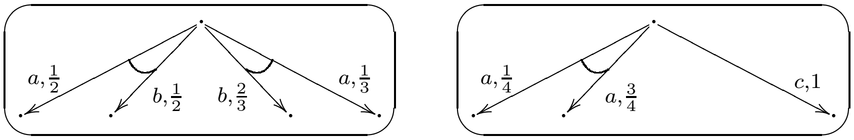

Fig. 4. Examples of Generative Transition Systems

3.2 PIOAs

Now we introduce the notion of probabilistic I/O automata as a combination of reactive and generative system types, in the presence of I/O distinction. Notice that, we impose I/O distinction not only on the action signature, but also on the transition structure.

Definition 2 *A probabilistic I/O automaton (PIOA) A is a tuple*

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

where:

(1) S_A is a set of states with initial state $s^0_A \in S_A$;
(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 Section 4.

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 $a \in I_A$.

In a typical automata-theoretic setting, an execution (or a path) is a sequence of states and actions in alternating fashion satisfying the obvious reachability condition. Our version, called an execution branch, is enriched with additional information from the reactive and generative transition structures.

Definition 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, where $\mathsf{last}(\underline{s}) := s$.*
- *For all $r \in \mathsf{Bran}(s)$, $a \in I_A$, $\mu \in \mathbf{R}_A(\mathsf{last}(r), a)$ 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, $\mathsf{last}(r.f.a.s') := s'$.*

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. Since we assume that S_A, $\mathbf{R}_A$ and $\mathbf{G}_A$ are all countable, we may infer that $\mathsf{Bran}(A)$ is also countable. An infinite branch from s is simply an infinite subset of $\mathsf{Bran}(s)$ that is linearly ordered by the prefix ordering on sequences, which is denoted $\sqsubseteq$. We write $\mathsf{Bran}^{\leq\omega}(s)$ for the set of finite and infinite branches from s. Similarly, $\mathsf{Bran}^{\leq\omega}(A) := \mathsf{Bran}^{\leq\omega}(s^0_A)$.

The *trace* of a branch $r \in \mathsf{Bran}(s)$ 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 $a \in H_A$.

Given a finite trace $\alpha \in (I_A \cup O_A)^{<\omega}$, we write $\mathsf{tr}^{\text{-}1}(\alpha)$ for the set of branches r in $\mathsf{Bran}(A)$ with $\mathsf{tr}(r) = \alpha$.

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 $\mathsf{last}(r) = s$.

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 4 below.

Definition 4 *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 $a \in I$, $\mathbf{R}_A(s, a) \subseteq \mathbf{R}_B(s, a)$ and $\mathbf{G}_A(s) \subseteq \mathbf{G}_B(s)$.*

3.3 I/O Schedulers and Execution Trees

As we saw in Section 1.3, the full observational power of history-dependent adversaries leads to unrealistic correlations in a parallel composition (Figure 2). To exclude these correlations from our semantics, we pair a PIOA with a set of acceptable schedulers, forming a probabilistic system (Definition 8 below).

We first make explicit the notion of schedulers in an I/O setting.

Definition 5 *Let A be a PIOA. An* input scheduler *σ for A is a partial function*

$$\sigma : \mathsf{Bran}(A) \times I \rightharpoonup \mathsf{Disc}(S_A)$$

such that: for all $\langle r, a\rangle \in \mathsf{Bran}(A) \times I$, if $\mathbf{R}_A(\mathsf{last}(r), a)$ is non-empty, then $\sigma(r, a)$ is defined and is in $\mathbf{R}_A(\mathsf{last}(r), a)$. An output scheduler *$\rho$ 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* $\rho(r) \in \mathbf{G}_A(\mathsf{last}(r))$*. 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 nondeterministic 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 by $\mathsf{last}(r)$, σ selects a distribution from $\mathbf{R}_A(last(r), a)$. 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 has slightly more freedom compared to its input counterpart: it may *halt* the execution by setting $\rho(r)$ to $\perp$, even if $\mathbf{G}_A(\mathsf{last}(r))$ is non-empty.

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 probabilistic automata framework [Seg95].

Definition 6 *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* $Q_{\sigma,\rho} : \mathsf{Bran}(A) \to [0, 1]$ *defined recursively by:*

- $Q_{\sigma,\rho}(\underline{s^0_A}) = 1$*;*
- *given* r' *of the form* $r.a.\mu.s'$*,*
  - $\cdot$ $Q_{\sigma,\rho}(r') := Q_{\sigma,\rho}(r) \cdot \mu(s')$*, if* $\mu = \sigma(r, a)$*;*
  - $\cdot$ $Q_{\sigma,\rho}(r') := 0$*, otherwise;*
- *given* r' *of the form* $r.f.a.s'$*,*
  - $\cdot$ $Q_{\sigma,\rho}(r') := Q_{\sigma,\rho}(r) \cdot f(\langle a,\, s'\rangle)$*, if* $f = \rho(r)$*;*
  - $\cdot$ $Q_{\sigma,\rho}(r') := 0$*, otherwise.*

If A is closed, then the input scheduler σ must be the empty function. In that case, we write Q_ρ for $Q_{\sigma,\rho}$. We claim that Q_ρ 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 [Seg95] 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 $\mathbf{m}_\rho[\mathbf{C}_r] = Q_\rho(r)$ for all r in $\mathsf{Bran}(A)$.

In this way, Q_ρ gives rise to the probability space $(\Omega_A, \mathcal{F}_A, \mathbf{m}_\rho)$, which is called a probabilistic execution in [Seg95]. Notice, for every $r \in \mathsf{Bran}(A)$, the singleton set $\{r\}$ is measurable (i.e., $\{r\} \in \mathcal{F}_A$). By countable additivity, we may conclude that Q_ρ also induces a discrete sub-probability distribution on $\mathsf{Bran}(A)$, namely, $\mathbf{P}_\rho(r) := \mathbf{m}_\rho(\{r\})$.

In case A is open, an execution tree does not always induce a probability measure. This is because I/O schedulers do not resolve nondeterministic choices that are external to A (i.e., those controlled by the computation environment). For example, given a branch r' of the form $r.a.\sigma(r,a).s'$, the value $Q_{\sigma,\rho}(r')$ is computed from $Q_{\sigma,\rho}(r)$ and $\sigma(r,a)(s')$, neither of which contains information about the probability of a being provided as an input by the environment.

As we introduce switched PIOAs in Section 4, the token structure eliminates nondeterministic choices between input events and locally controlled activities. In that setting, execution trees acquire better structural properties, because all remaining nondeterministic choices are those among different inputs. This allows us to construct conditional probability distributions as follows: given a finite trace α, an execution tree induces a sub-probability distribution on the set of branches with trace α. (Proposition 10 is a discrete version of this claim). This holds even for switched PIOAs that are open. As a result, our trace-style abstraction of execution trees is well-defined.

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 (unconditional) probability measure.

4 Switched PIOAs

We now augment the PIOA model of Section 3 with additional structures and axioms, yielding the notion of switched PIOAs. These changes are prompted by our proposal to use distributed scheduling (cf. Section 1.4). Namely, we use a token structure to eliminate global scheduling conflicts, 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.

In order to implement this token structure, we must distinguish between *active* and *inactive* states of an automaton. Moreover, we designate special *control actions* and impose five *switch axioms*, formalizing our intuitions about control passage among components. This leads to Definition 7 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 7 *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) $\mathsf{active}_A(s) = 0 \;\Rightarrow\; (\mathbf{G}_A(s) = \emptyset \wedge \forall a \in I_A.\ \mathbf{R}_A(s,a) \neq \emptyset)$
(S2) $\mathsf{active}_A(s) = 1 \;\Rightarrow\; \forall a \in I_A.\ \mathbf{R}_A(s,a) = \emptyset$
(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) $(s \xrightarrow{a} s' \;\wedge\; a \in \mathsf{BAct} \cup H_A \cup \mathsf{Sync}_A) \;\Rightarrow\; \mathsf{active}_A(s) = \mathsf{active}_A(s')$

Definition 8 *A* switched 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 in the sense of Definition 5. Such a system is* full *if $\mathcal{S}$ is the set of all I/O schedulers for A.*

For better 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 $\mathsf{active}_A(s^0_A) = 1$. 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 the various types of actions 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 9 below.

Lemma 9 *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 $\mu \in \mathbf{R}_A(s, a)$ 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 1. Their adaptations to the switched PIOA framework are illustrated in Figure 5 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.

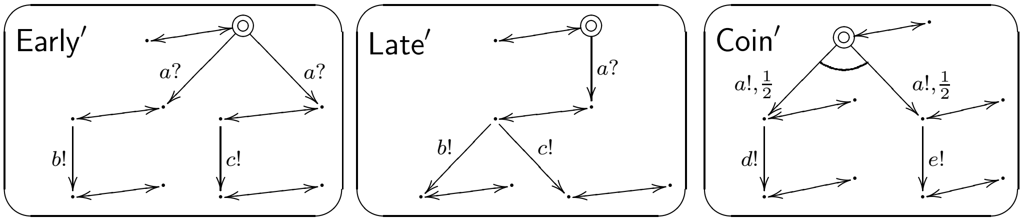

Fig. 5. Adaptations of Early, Late and Coin

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.

Recall from Section 3.3 that each I/O scheduler $\langle \sigma,\, \rho \rangle$ for a PIOA A induces an execution tree $Q_{\sigma,\rho} : \mathsf{Bran}(A) \to [0,1]$. Also, 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 demonstrate the fact 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 below.

First, we need the notion of minimal branches: 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. Notice, the empty branch is minimal. For non-empty r, it is minimal if and only if its last action label is visible. We write $\mathsf{Bran}_{\min}(s)$ for the set of minimal branches in $\mathsf{Bran}(s)$. For each $\alpha \in (I_A \cup O_A)^{<\omega}$, let $\mathsf{tr}^{-1}_{\min}(\alpha)$ denote the set of minimal branches of A with trace α.

Minimality is important because distinct minimal branches always represent mutually exclusive events, even if these branches have the same trace. In contrast, if we drop the minimality requirement, distinct branches with the same trace may be prefix-related; that is, one event may be strictly included in the other. Restricting our attention to minimal branches, we are able to define discrete probability distributions directly, bypassing the usual cone construction (cf. Section 3.3).

Proposition 10 *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 $\mathsf{tr}^{-1}(\alpha)$ is nonempty. Then the restriction of $Q_{\sigma,\rho}$ to $\mathsf{tr}^{-1}_{\min}(\alpha)$ is a discrete sub-probability distribution.*

The proof of Proposition 10 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 9.

Lemma 11 *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 9, we know that $\mathsf{active}_A(\mathsf{last}(r)) = \mathsf{active}_A(t) = 1$. □

Extending Lemma 11, we show that inputs transitions are never preceded by hidden transitions.

Lemma 12 *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 9 guarantees that $\mathsf{active}_A(\mathsf{last}(r)) = 0$. Thus, by Lemma 11, r must be minimal. □

Next we consider a situation in which an output action a takes place after a trace α. Observe that Lemma 13 applies to PIOAs in general (i.e., it does not require switch axioms).

Lemma 13 *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 $\mathsf{tr}^{\text{-1}}(\alpha a)$ in A is nonempty. The following hold for every $r \in \mathsf{tr}^{\text{-1}}_{\min}(\alpha)$.*

(i) Let $\mathbf{C}$ denote the set of branches $r' \in \mathsf{tr}^{\text{-1}}(\alpha)$ such that $r \sqsubseteq r' \sqsubseteq r''$ for some $r'' \in \mathsf{tr}^{\text{-1}}_{\min}(\alpha a)$. 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 $\sum_{r' \in \mathbf{C}_k} Q_{\sigma,\rho}(r') \leq Q_{\sigma,\rho}(r)$.

(ii) $\sum_{r'' \in \mathsf{tr}^{\text{-1}}_{\min}(\alpha a),\ r \sqsubseteq r''} Q_{\sigma,\rho}(r'') \leq Q_{\sigma,\rho}(r)$.

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 $Q_{\sigma,\rho}$, we know that $Q_{\sigma,\rho}(r'') \neq 0$ implies $f = \rho(r')$ and $\langle b,\, s''\rangle \in \mathsf{Supp}(f)$. Therefore we have the following.

$$
\begin{aligned}
&\sum_{r'' \in \mathbf{C}_{k+1}} Q_{\sigma,\rho}(r'') \\
&= \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \bot\}} \ \sum_{\{\langle b,\, t\rangle \in \mathsf{Supp}(\rho(r')) \ \mid\ r'.\rho(r').b.t \in \mathbf{C}_{k+1}\}} Q_{\sigma,\rho}(r') \cdot \rho(r')(\langle b,\, t\rangle) \\
&= \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \bot\}} (Q_{\sigma,\rho}(r') \cdot \sum_{\{\langle b,\, t\rangle \in \mathsf{Supp}(\rho(r')) \ \mid\ r'.\rho(r').b.t \in \mathbf{C}_{k+1}\}} \rho(r')(\langle b,\, t\rangle)) \\
&\leq \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \bot\}} Q_{\sigma,\rho}(r') \cdot 1 \\
&\leq \sum_{r' \in \mathbf{C}_k} Q_{\sigma,\rho}(r')
\end{aligned}
$$

By the induction hypothesis, this is at most $Q_{\sigma,\rho}(r)$.

We move on to Item (ii). By the definition of minimality, every $r'' \in \mathsf{tr}^{\text{-1}}_{\min}(\alpha a)$ is of the from $r'.f.a.s''$ with $r' \in \mathbf{C}_k$ for some k. Again, by the definition of

$Q_{\sigma,\rho}$, we have $Q_{\sigma,\rho}(r'') \neq 0$ implies $f = \rho(r)$ and $\langle a, s'' \rangle \in \mathsf{Supp}(f)$. This implies:

$$\begin{aligned}
&\sum_{r'' \in \mathsf{tr}^{-1}_{\min}(\alpha a),\ r \sqsubseteq r''} Q_{\sigma,\rho}(r'') \\
&= \sum_{k=0}^{\infty} \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \perp\}} \sum_{\{s'' \ \mid\ \langle a, s'' \rangle \in \mathsf{Supp}(\rho(r'))\}} Q_{\sigma,\rho}(r') \cdot \rho(r')(\langle a, s'' \rangle).
\end{aligned}$$

Therefore, it suffices to show that all partial sums are less than or equal to $Q_{\sigma,\rho}(r)$. 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

$$\begin{aligned}
&\sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \perp\}} \sum_{\{s'' \ \mid\ \langle a, s'' \rangle \in \mathsf{Supp}(\rho(r'))\}} Q_{\sigma,\rho}(r') \cdot \rho(r')(\langle a, s'' \rangle) \\
&= \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \perp\}} Q_{\sigma,\rho}(r') \cdot L_{r'} \\
&\leq \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \perp\}} Q_{\sigma,\rho}(r') \cdot (1 - M_{r'}) \\
&= \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \perp\}} Q_{\sigma,\rho}(r') - \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \perp\}} Q_{\sigma,\rho}(r') \cdot M_{r'} \\
&\leq \sum_{r' \in \mathbf{C}_k} Q_{\sigma,\rho}(r') \ - \sum_{r' \in \mathbf{C}_{k+1}} Q_{\sigma,\rho}(r'),
\end{aligned}$$

where the last inequality follows from the proof of Item (i). Now, for all $K \in \mathbb{N}$,

$$\begin{aligned}
&\sum_{k=0}^{K} \sum_{\{r' \in \mathbf{C}_k \ \mid\ \rho(r') \neq \perp\}} \sum_{\{s'' \ \mid\ \langle a, s'' \rangle \in \mathsf{Supp}(\rho(r'))\}} Q_{\sigma,\rho}(r') \cdot \rho(r')(\langle a, s'' \rangle) \\
&\leq \sum_{k=0}^{K} (\sum_{r' \in \mathbf{C}_k} Q_{\sigma,\rho}(r') \ - \sum_{r' \in \mathbf{C}_{k+1}} Q_{\sigma,\rho}(r')) \\
&= \sum_{r' \in \mathbf{C}_0} Q_{\sigma,\rho}(r') \ - \sum_{r' \in \mathbf{C}_{K+1}} Q_{\sigma,\rho}(r')) \\
&\leq \sum_{r' \in \mathbf{C}_0} Q_{\sigma,\rho}(r') = Q_{\sigma,\rho}(r). \qquad \square
\end{aligned}$$

We are now ready to prove Proposition 10.

PROOF. [Proposition 10] 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 $Q_{\sigma,\rho}(\underline{s^0_A}) = 1$.

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 12, r is minimal and hence in $\mathsf{tr}^{-1}_{\min}(\alpha)$. Moreover, by the definition of $Q_{\sigma,\rho}$, we know that $Q_{\sigma,\rho}(r') \neq 0$ implies $\mu = \sigma(r, a)$. Therefore,

$$\begin{aligned}
&\sum_{r' \in \mathsf{tr}^{-1}_{\min}(\alpha')} Q_{\sigma,\rho}(r') \\
&= \sum_{\{r \in \mathsf{tr}^{-1}_{\min}(\alpha) \;\mid\; \sigma(r,a) \neq \bot\}} \;\; \sum_{s' \in \mathsf{Supp}(\sigma(r,a))} Q_{\sigma,\rho}(r) \cdot \sigma(r,a)(s') \\
&= \sum_{\{r \in \mathsf{tr}^{-1}_{\min}(\alpha) \;\mid\; \sigma(r,a) \neq \bot\}} (Q_{\sigma,\rho}(r) \cdot \sum_{s' \in \mathsf{Supp}(\sigma(r,a))} \sigma(r,a)(s')) \\
&\leq \sum_{\{r \in \mathsf{tr}^{-1}_{\min}(\alpha) \;\mid\; \sigma(r,a) \neq \bot\}} Q_{\sigma,\rho}(r) \\
&\leq \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} Q_{\sigma,\rho}(r) \;\leq 1,
\end{aligned}$$

  where the last inequality follows from the induction hypothesis.
- $a \in O_A$. By Lemma 13, we have

$$\begin{aligned}
\sum_{r' \in \mathsf{tr}^{-1}_{\min}(\alpha')} Q_{\sigma,\rho}(r') &= \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} \;\; \sum_{r'' \in \mathsf{tr}^{-1}_{\min}(\alpha a),\; r \sqsubseteq r''} Q_{\sigma,\rho}(r'') \\
&\leq \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} Q_{\sigma,\rho}(r) \leq 1.
\end{aligned}$$

  Again, the last inequality follows from the induction hypothesis. □

4.2 Likelihood Assignments

We are now ready to define a 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. This is analogous to the notion of *trace distributions* in [Seg95], where a trace distribution is obtained from a probabilistic automaton (without I/O distinction) together with a randomized, history-dependent scheduler.

Just as trace distributions are behavioral abstractions of probabilistic executions, likelihood assignments are behavioral abstractions of execution trees. Roughly speaking, the probability of observing a certain trace $\alpha \in \mathsf{Act}^{<\omega}$ 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. Since execution trees of open PIOAs do not always induce probability measures, we opt for the term "likelihood", as opposed to "probability". Nonetheless, the method of abstraction is completely analogous; namely, it is done via a lifting of the trace operator $\mathsf{tr} : \mathsf{Bran}(A) \to (I_A \cup O_A)^{<\omega}$.

Definition 14 *Let A be a switched PIOA and let $\langle \sigma, \rho \rangle$ be an I/O scheduler for A. The* likelihood assignment *induced by $\langle A, \sigma, \rho \rangle$, denoted $\mathbf{L}_{\sigma,\rho}$, is the function $\mathsf{tr}(Q_{\sigma,\rho}) : (I_A \cup O_A)^{<\omega} \to [0, 1]$ given as follows.*

$$\mathsf{tr}(Q_{\sigma,\rho})(\alpha) := \sum_{r \in \mathsf{tr}^{-1}_{\min}(\alpha)} Q_{\sigma,\rho}(r).$$

This is well-defined by virtue of Proposition 10.

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 $\Omega := O_A^{\leq\omega}$. The σ-field $\mathcal{F}$ on Ω is generated by the collection $\{\mathbf{C}_\alpha \mid \alpha \in O_A^{<\omega}\}$, 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 $\alpha \in O_A^{<\omega}$. This yields a probability space $\langle \Omega_A, \mathcal{F}_A, \mathbf{m}_\rho \rangle$, which is called a trace distribution in [Seg95].

Thus, our notions of execution trees and likelihood assignments can be seen as generalizations of probabilistic executions and trace distributions, respectively. Since the latter 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.

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). This leads to a very simple and natural notion of external behavior.

Definition 15 *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 16 *Switched probabilistic systems* $\mathcal{A} = \langle A, \mathcal{S} \rangle$ *and* $\mathcal{B} = \langle B, \mathcal{T} \rangle$ *are said to be* comparable *if:*

- $\mathsf{active}_A(s_A^0) = \mathsf{active}_B(s_B^0)$ *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* $\mathsf{ExtBeh}(\mathcal{A}) \subseteq \mathsf{ExtBeh}(\mathcal{B})$*.*

5 Parallel Composition

In this section, we define parallel composition in an incremental fashion. First we do so for PIOAs (Section 5.1), specifying the composite transition structures. As usual, this definition is based on action synchronization and does not attempt to resolve nondeterministic choices among parallel components. In Section 5.2, we extend this composition operator to switched PIOAs, taking care that the composite still satisfies all switch axioms.

Then, departing from the "compose-and-schedule" approach (cf. Section 1.2), we describe how to compose I/O schedulers for compatible switched PIOAs to form a single I/O scheduler for their composite (Section 5.3). This extends easily to parallel composition for probabilistic systems. Thus, our approach can be described as "schedule-and-compose", where parallel composition is imposed *after* local schedules have been completely specified.

Unlike most composition operators in the literature, our definitions do *not* involve normalization mechanisms, which collect and redistribute deadlock probabilities. Instead, we take advantage of I/O distinction and use probabilistic input enabling to make sure that deadlocks never occurs.

5.1 Composing PIOAs

Let us begin with an example to illustrate how we intend to compose reactive and generative transition structures. Consider automata A, B and C in Figure 6 and assume that action a is in the signatures of all three automata, while b is in the signatures of B and C only.

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

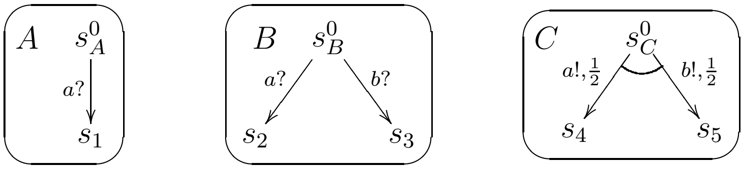

Fig. 6. Automata A, B and C

only B reacts and A simply stutters (i.e., no transition takes place). This is illustrated in Figure 7 on the left.

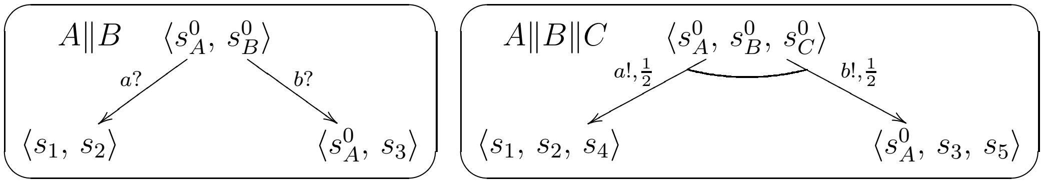
$A \| B$
$\langle s_A^0, s_B^0 \rangle$
$a?$
$b?$
$\langle s_1, s_2 \rangle$
$\langle s_A^0, s_3 \rangle$
$A \| B \| C$
$\langle s_A^0, s_B^0, s_C^0 \rangle$
$a!, \frac{1}{2}$
$b!, \frac{1}{2}$
$\langle s_1, s_2, s_4 \rangle$
$\langle s_A^0, s_3, s_5 \rangle$

Fig. 7. Parallel Composites $A \| B$ and $A \| B \| C$

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 7 on the right.

Despite its simplicity, Figure 7 demonstrates 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. In the rest of this section, we try to formalize this simple idea in the general setting of PIOAs, where components may exhibit nondeterministic behavior.

We start with the notion of compatibility: two PIOAs A and B are said to be *compatible* if $O_A \cap O_B = \mathsf{Act}_A \cap H_B = \mathsf{Act}_B \cap H_A = \emptyset$.

Let $\{A_i \mid 1 \leq i \leq n\}$ denote a set of pairwise compatible PIOAs and, for readability, we replace all subscripts A_i with i. (The same convention will be adopted throughout this paper.) The *parallel composite*, denoted $\upuparrows_{i=1}^{n} A_i$, is the PIOA D with the following state space and action signature:

(1) $S_D := \prod_{i=1}^{n} S_i$ with $s_D^0 := \langle s_1^0, \ldots, s_n^0 \rangle$;
(2) $I_D := \bigcup_{i=1}^{n} I_i \setminus \bigcup_{i=1}^{n} O_i$, $O_D := \bigcup_{i=1}^{n} O_i$, and $H_D := \bigcup_{i=1}^{n} H_i$;

The reactive transition structure $\mathbf{R}_D$ and the generative transition structure $\mathbf{G}_D$ are given in Definition 17 and Definition 18, respectively.

Definition 17 *Let $\vec{s} \in S_D$ and $a \in I_D$ be given. We define $\mathbf{R}_D(\vec{s}, a) \subseteq \mathsf{Disc}(S_D)$ 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, $\mu_i \in \mathbf{R}_i(s_i, a)$.*

In other words, each process A_i stutters if the given input a is not in the signature of A_i. Otherwise, A_i reacts to this input by

(i) first choosing *nondeterministically* a distribution μ_i from $\mathbf{R}_i(s_i, a)$;
(ii) then choosing *randomly* a state t_i according to μ_i.

We assume that processes evolve independently, therefore a product construction on state distributions μ_i yields a typical member of $\mathbf{R}_D(\vec{s}, a)$.

The definition of $\mathbf{G}_D$ for $D =\upuparrows_{i=1}^{n} A_i$ is slightly more complicated, where exactly one component D_j is generative and all others are reactive.

Definition 18 *Let $\vec{s} \in S_D$ and $1 \leq j \leq n$ be given. Let N_j denote the index set $(O_D \cup H_D) \times \{i \mid 1 \leq i \leq n, i \neq j\}$. 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, $\mu_{a,i} \in \mathbf{R}_i(s_i, a)$.*

Then g_j and $\vec{\mu}$ are said to generate *the following distribution f on $(O_D \cup H_D) \times S_D$: 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 $\mathbf{G}_D^j(\vec{s}) \subseteq \mathsf{Disc}((O_D \cup H_D) \times S_D)$ 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 $\mathbf{G}_D(\vec{s}) := \bigcup_{1 \leq j \leq n} \mathbf{G}_D^j(\vec{s})$.

Here the unique active component A_j chooses *nondeterministically* a transition bundle g_j enabled from s_j. 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 is 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 19 below shows that the new bundles f constructed in Definition 18 are in fact well-defined discrete distributions.

Lemma 19 *The bundle f in Definition 18 is well-defined.*

PROOF. We need to verify that f is a discrete distribution on $(O_D \cup H_D) \times S_D$. First consider fixed $a \in O_j \cup H_j$. By the definition of f, we have

$$\sum_{\vec{t} \in S_D} f(\langle a, \vec{t} \rangle) = \sum_{t_1 \in S_1} \cdots \sum_{t_n \in S_n} g_j(\langle a, t_j \rangle) \cdot \prod_{i \neq j} \mu_{a,i}(t_i).$$

We can rearrange the sums and factor out $g_j(\langle a, t_j \rangle)$ to obtain:

$$\sum_{t_j \in S_j} g_j(\langle a, t_j \rangle) \cdot (\sum_{t_1 \in S_1} \cdots \sum_{t_{j-1} \in S_{j-1}} \sum_{t_{j+1} \in S_{j+1}} \cdots \sum_{t_n \in S_n} \prod_{i \neq j} \mu_{a,i}(t_i)).$$

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 $\sum_{\vec{t} \in S_D} f(\langle a, \vec{t} \rangle) = \sum_{t_j \in S_j} g_j(\langle a, t_j \rangle)$.

Now notice that $f(\langle a, \vec{t} \rangle) = 0$ whenever $a \notin O_j \cup H_j$. Therefore,

$$\begin{aligned} \sum_{\langle a, \vec{t} \rangle \in (O_D \cup H_D) \times S_D} f(\langle a, \vec{t} \rangle) &= \sum_{a \in O_j \cup H_j} \sum_{\vec{t} \in S_D} f(\langle a, \vec{t} \rangle) \\ &= \sum_{a \in O_j \cup H_j} \sum_{t_j \in S_j} g_j(\langle a, t_j \rangle) && \text{calculation above} \\ &= 1 && g_j \text{ discrete distribution} \end{aligned}$$

Therefore f is a discrete distribution on $(O_D \cup H_D) \times S_D$. □

This completes the definition of parallel composition for PIOAs. We write $\upuparrows^n$ 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 $\upuparrows$ is commutative. We claim that $\upuparrows$ is also associative, because both $(A \upuparrows B) \upuparrows C$ and $A \upuparrows (B \upuparrows C)$ are isomorphic to $\upuparrows^3 \{A, B, C\}$. We omit the details.

5.2 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;
- $\mathsf{Act}_A \cap \mathsf{Sync}_B = \mathsf{Act}_B \cap \mathsf{Sync}_A = \mathsf{CI}_A \cap \mathsf{CI}_B = \emptyset$;
- at most one of them is initially active.

Since switched PIOAs are special instances of PIOAs, one may apply the operator $\upuparrows$ of Section 5.1 to compatible switched PIOAs. Unfortunately, the result does not always satisfy all switch axioms. We give a simple example.

Consider automata D and E in Figure 8 below and assume that all actions shown are control actions.

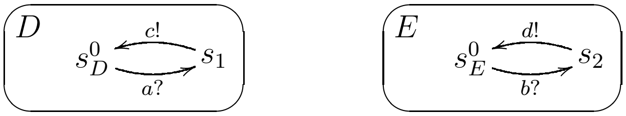

Fig. 8. Automata D and E

If from the initial state the composite $D \upuparrows E$ 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 9. 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 9).

$$\langle s_D^0, s_E^0 \rangle \xrightarrow{a?} \langle s_1, s_E^0 \rangle \xrightarrow{b?} \langle s_1, s_2 \rangle \xrightarrow{c!} \langle s_D^0, s_2 \rangle$$

Fig. 9. A Potential Execution of $D \upuparrows E$

This is a counterintuitive scenario: if the environment of $D \upuparrows E$ 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 $\upuparrows$, 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. This is done in Definition 20 below.

Definition 20 *Let* $\{A_i \mid i \in I\}$ *be a set of pairwise compatible switched PIOAs. The* parallel composite $\|_{i=1}^n A_i$ *is the sub-automaton* E *of* $\upuparrows_{i=1}^n A_i$

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, $\mathsf{Sync}_E := \bigcup_{1\leq i\leq n} \mathsf{Sync}_i \cup \bigcup_{1\leq i,j\leq n}(\mathsf{CI}_i \cap \mathsf{CO}_j)$, *and* $\mathsf{active}_E(\vec{s}) := 0$ *if and only if* $\mathsf{active}_i(s_i) = 0$ *for all* i.

Although the signature of $E =\|_{i=1}^{n} A_i$ is completely specified in Definition 20, it is instructive to provide a list of explicit identities.

Lemma 21 *The following equalities hold:*

- $\mathsf{BI}_E = \bigcup_{1\leq i\leq n} \mathsf{BI}_i \setminus \bigcup_{1\leq i\leq n} \mathsf{BO}_i$;
- $\mathsf{CI}_E = \bigcup_{1\leq i\leq n} \mathsf{CI}_i \setminus \bigcup_{1\leq i\leq n} \mathsf{CO}_i$;
- $\mathsf{BO}_E = \bigcup_{1\leq i\leq n} \mathsf{BO}_i$;
- $\mathsf{CO}_E = \bigcup_{1\leq i\leq n} \mathsf{CO}_i \setminus \bigcup_{1\leq i\leq n} \mathsf{CI}_i$.

PROOF. By definition, $I_E = \bigcup_{1\leq i\leq n} I_i \setminus \bigcup_{1\leq i\leq n} O_i$. Since BAct and CAct are disjoint, we have the desired properties about BI_E and CI_E.

Similarly, $O_E = \bigcup_{1\leq i\leq n} O_i$, therefore $\mathsf{BO}_E = \bigcup_{1\leq i\leq n} \mathsf{BO}_i$ and $O_E \cap \mathsf{CAct} = \bigcup_{1\leq i\leq n} \mathsf{CO}_i$. Applying the definitions of CO_E and Sync_E, we have
$\mathsf{CO}_E = \bigcup_{1\leq i\leq n} \mathsf{CO}_i \setminus \bigcup_{1\leq i\leq n} \mathsf{CI}_i$. □

To show that such E is a well-defined PIOA, we need to verify (i) $s_E^0 \in S_E$, and (ii) S_E is closed under the transition structures $\mathbf{R}_E$ and $\mathbf{G}_E$. Clearly, the first claim holds by the definition of compatibility. The second is confirmed by Lemmas 22 and 23 below.

For convenience, we partition S_E into two sets:

- $S_{E,0}$ is the set of all $\vec{s}$ such that $\mathsf{active}_i(s_i) = 0$ for all i;
- $S_{E,1}$ is the set of all $\vec{s}$ such that $\mathsf{active}_i(s_i) = 1$ for exactly one i.

Lemma 22 *Let* $\vec{s} \in S_E$ *and* $a \in I_E$ *be given. For all* $\mu \in \mathbf{R}_E(\vec{s}, a)$*:*

- $a \in \mathsf{BI}_E$ *implies* $\mathsf{Supp}(\mu) \subseteq S_{E,0}$;
- $a \in \mathsf{CI}_E$ *implies* $\mathsf{Supp}(\mu) \subseteq S_{E,1}$.

PROOF. By Definition 20, $\mathbf{R}_E(\vec{s}, a)$ is empty whenever $\vec{s} \in S_{E,1}$. Therefore we may assume that $\vec{s} \in S_{E,0}$. Let $\mu \in \mathbf{R}_E(\vec{s}, a)$ and $\vec{s}\,' \in \mathsf{Supp}(\mu)$ be given.

First assume $a \in \mathsf{BI}_E$. 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 9 to conclude that $\mathsf{active}_i(s'_i) = 0$. Therefore $\vec{s}\,' \in S_{E,0}$.

Now assume $a \in \mathsf{CI}_E$. By compatibility, $a \in \mathsf{Act}_j$ for exactly one j. Choose such j. By Lemma 9, 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 $\vec{s}\,' \in S_{E,1}$. □

Lemma 23 *Let $\vec{s} \in S_E$ and $f \in \mathbf{G}_E(\vec{s})$ be given. For every $\langle a, \vec{s}\,'\rangle \in \mathsf{Supp}(f)$;*

- *If $a \in \mathsf{BO}_E \cup \mathsf{Sync}_E \cup H_E$, then $\vec{s}\,' \in S_{E,1}$;*
- *If $a \in \mathsf{CO}_E$, then $\vec{s}\,' \in S_{E,0}$.*

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 $s \in S_{E,1}$, because otherwise $\mathbf{G}_E(\vec{s})$ 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 18, 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 9 to A_j and conclude that $\mathsf{active}_i(s'_j) = \mathsf{active}_i(s_j) = 1$. Therefore, $\vec{s}\,' \in S_{E,1}$.

(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 9 to conclude that $\mathsf{active}_i(s'_i) = 0$. As in the previous case, we know $\mathsf{active}_i(s'_j) = 1$. Therefore, $\vec{s}\,' \in S_{E,1}$.

(3) $a \in \mathsf{CO}_j \cap \mathsf{CI}_k$ for some $k \neq j$. By Lemma 9, 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 $\vec{s}\,' \in S_{E,1}$.

(4) $a \in \mathsf{CO}_E$. By the definition of CO_E, 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 9, we have $\mathsf{active}_j(s'_j) = 0$. Thus, $\vec{s}\,' \in S_{E,0}$. □

It remains to show that E satisfies all switch axioms.

Lemma 24 *The PIOA E, together with active_E and Sync_E, satisfies Axioms (S1) through (S5) in Definition 7.*

PROOF. Note that $\mathsf{active}_E(\vec{s}) = 0$ if and only if $\vec{s} \in S_{E,0}$. For Axiom (S1), let $\vec{s} \in S_{E,0}$ and $a \in I_E$ be given. Applying Axiom (S1) on each component, we know that $\mathbf{G}_i(s_i)$ is empty for every i and hence $\mathbf{G}_E(\vec{s}) = \emptyset$. On the other hand, for all i with $a \in I_i$, Axiom (S2) requires $\mathbf{R}_i(s_i, a)$ is non-empty. Hence $\mathbf{R}_E(\vec{s}, a)$ is non-empty. This proves that E satisfies Axiom (S1).

Axiom (S2) follows from the definition of $\mathbf{R}_E$. Axioms (S3) through (S5) follow from Lemmas 22 and 23. □

We adopt the same notational conventions as with $\upuparrows$. 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.

5.3 Composing I/O Schedulers

The goal of this section is to extend the parallel operator $\|$ to probabilistic systems, therefore we consider composition of I/O schedulers. For that end, we need some basic notions of projection. Notice, these projection operators apply to PIOAs in general, not just switched PIOAs.

In Section 2, we described projection operators for discrete distributions on a product space. Extending the same idea, we define projection on composite transition bundles.

Definition 25 *Let $\{A_i \mid 1 \leq i \leq n\}$ be a set of pairwise compatible PIOAs and let D denote $\upuparrows_{i=1}^n A_i$. Let $\vec{s} \in S_D$ and $f \in \mathbf{G}_D(\vec{s})$ be given. Let j be the unique index such that $\pi_L(\mathsf{Supp}(f)) \subseteq O_j \cup H_j$ (equivalently, $f \in \mathbf{G}_D^j(\vec{s})$). The* jth-projection *of f, denoted $\pi_j(f)$, is the discrete distribution on $(O_j \cup H_j) \times S_j$ given by:*

$$\pi_j(f)(\langle a, t\rangle) := \sum_{\vec{t} \in S_D \ : \ t_j = t} f(\langle a, \vec{t}\rangle).$$

For every $a \in \pi_L(\mathsf{Supp}(f))$ and $i \neq j$, the $\langle a, i\rangle$th-projection *of f, denoted $\pi_{a,i}(f)$, is the discrete distribution on S_i given by:*

$$\pi_{a,i}(f)(t) := \frac{\sum_{\vec{t} \in S_D \ : \ t_i = t, t_j = u} f(\langle a, \vec{t}\rangle)}{\pi_j(f)(\langle a, u\rangle)},$$

where u is any state in S_j such that $\pi_j(f)(\langle a, u\rangle) \neq 0$.

Lemmas 26 and 27 below show that these projection operators are in fact well-defined.

Lemma 26 *The distribution $\pi_j(f)$ in Definition 25 is well-defined and is in $\mathbf{G}_j(s_j)$.*

PROOF. By the definition of $\mathbf{G}_D(\vec{s})$, we may choose $g_j \in \mathbf{G}_j(s_j)$ such that f is generated by g_j. It suffices to show $\pi_j(f) = g_j$. Let $\langle a,\, t\rangle \in (O_j \cup H_j) \times S_j$ be given. By definition,

$$\pi_j(f)(\langle a,\, t\rangle) = \sum_{\vec{t} \in S_D \;:\; t_j = t} f(\langle a,\, \vec{t}\rangle) = \sum_{\vec{t} \in S_D \;:\; t_j = t} g_j(\langle a,\, t_j\rangle) \cdot \prod_{i \neq j} \mu_{a,i}(t_i).$$

We can rearrange the sums and factor out $g_j(\langle a,\, t_j\rangle)$ to obtain:

$$\pi_j(f)(\langle a,\, t\rangle) = g_j(\langle a,\, t\rangle) \cdot (\sum_{t_1 \in S_1} \cdots \sum_{t_{j-1} \in S_{j-1}} \sum_{t_{j+1} \in S_{j+1}} \cdots \sum_{t_n \in S_n} \prod_{i \neq j} \mu_{a,i}(t_i)).$$

Since every $\mu_{a,i}$ is a discrete distribution on S_i, the second factor equals 1. Hence $\pi_j(f)(\langle a,\, t\rangle) = g_j(\langle a,\, t\rangle)$. □

Lemma 27 *The distribution $\pi_{a,i}(f)$ in Definition 25 is well-defined. Moreover, if $a \in I_i$, then $\pi_{a,i}(f) \in \mathbf{R}_i(s_i, a)$; otherwise, $\pi_{a_i}(f) = \mathsf{Dirac}(s_i)$.*

PROOF. By the definition of $\mathbf{G}_D(\vec{s})$, 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 (in part) by $\mu_{a,i}$ and g_j. It suffices to show $\pi_{a,i}(f) = \mu_{a,i}$. Let $t \in S_i$ be given. By definition, $\pi_{a,i}(f)(t)$ equals

$$\frac{\sum_{\vec{t} \in S_D \,:\, t_i = t,\, t_j = u} f(\langle a,\, \vec{t}\rangle)}{\pi_j(f)(\langle a,\, u\rangle)} = \frac{\sum_{\vec{t} \in S_D \,:\, t_i = t,\, t_j = u} (g_j(\langle a,\, t_j\rangle) \cdot \prod_{k \neq j} \mu_{a,k}(t_k))}{\pi_j(f)(\langle a,\, u\rangle)}.$$

Factoring out $g_j(\langle a,\, u\rangle)$ and $\mu_{a,i}(t)$, the numerator becomes

$$g_j(\langle a,\, u\rangle) \cdot \mu_{a,i}(t) \cdot \sum_{\vec{t} \in S_D \,:\, t_i = t,\, t_j = u} \prod_{k \neq i,j} \mu_{a,k}(t_k)).$$

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 26 that $\pi_j(f) = g_j$, therefore the denominator equals $g_j(\langle a,\, u\rangle)$. Now we have

$$\pi_{a,i}(f)(t) = \frac{g_j(\langle a,\, u\rangle) \cdot \mu_{a,i}(t)}{g_j(\langle a,\, u\rangle)} = \mu_{a,i}(t).$$

Notice we have not 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 28 *Let $\{A_i \mid 1 \leq i \leq n\}$ be a set of pairwise compatible PIOAs and let D denote $\upuparrows_{i=1}^{n} A_i$. Let $\vec{s} \in S_D$ and $1 \leq i \leq n$ be given. We define, recursively, the i-*th projection *operator on* $\mathsf{Bran}(\vec{s})$ *as follows:*

- $\pi_i(\underline{\langle s_1^0, \, \ldots, \, s_n^0 \rangle}) := \underline{s_i^0};$
- $\pi_i(r.a.\mu.\vec{t})$ *equals*
  - $\cdot$ $\pi_i(r).a.\pi_i(\mu).t_i$, *if* $a \in I_i$;
  - $\cdot$ $\pi_i(r)$, *otherwise;*
- $\pi_i(r.f.a.\vec{t})$ *equals*
  - $\cdot$ $\pi_i(r).\pi_i(f).a.t_i$, *if i is the unique index with* $\pi_L(\mathsf{Supp}(f)) \subseteq O_i \cup H_i$;
  - $\cdot$ $\pi_i(r).a.\pi_{a,i}(f).t_i$, *if* $a \in I_i$;
  - $\cdot$ $\pi_i(r)$, *otherwise.*

These projected branches are well-defined by virtue of Lemma 29 below.

Lemma 29 *Let $1 \leq i \leq n$ be given. For all $q \in \mathsf{Bran}(\vec{s})$, we have*

(1) $\pi_i(\mathsf{last}(q)) = \mathsf{last}(\pi_i(q))$;
(2) if q is of the form $r.a.\mu.\vec{t}$ and $a \in I_i$, then $\pi_i(\mu) \in \mathbf{R}_i(\mathsf{last}(\pi_i(r)), a)$ and $t_i \in \mathsf{Supp}(\pi_i(\mu))$;
(3) if q is of the form $r.f.a.\vec{t}$ and $a \in O_i \cup H_i$, then $\pi_i(f) \in \mathbf{G}_i(\mathsf{last}(\pi_i(r)))$ and $\langle a, \, t_i \rangle \in \mathsf{Supp}(\pi_i(f))$;
(4) if q is of the form $r.f.a.\vec{t}$ and $a \in I_i$, then $\pi_{a,i}(f) \in \mathbf{R}_i(\mathsf{last}(\pi_i(r)), a)$ and $t_i \in \mathsf{Supp}(\pi_{a,i}(f))$;

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 $\pi_i(\mathsf{last}(r)) = u_i = \mathsf{last}(\pi_i(r))$. Recall that μ is of the form $\prod_{i=1}^{n} \pi_i(\mu_i)$. We have two cases.

- $a \in I_i$. Then by Definition 17 we have $\pi_i(\mu) \in \mathbf{R}_i(u_i, a)$. Since $\vec{t} \in \mathsf{Supp}(\mu)$, it must be that $t_i \in \mathsf{Supp}(\pi_i(\mu))$. Moreover, $\pi_i(\mathsf{last}(q)) = t_i = \mathsf{last}(\pi_i(q))$.
- $a \notin I_i$. Then by Definition 17 we have $\pi_i(\mu) = \mathsf{Dirac}(u_i)$. Since $\vec{t} \in \mathsf{Supp}(\mu)$, it must be that $t_i = u_i$. Therefore $\pi_i(\mathsf{last}(q)) = t_i = u_i = \mathsf{last}(\pi_i(r)) = \mathsf{last}(\pi_i(q))$.

Now we consider a branch of the form $r.f.a.\vec{t}$. Again, let $\vec{u}$ denote $\mathsf{last}(r)$ and we have $\pi_i(\mathsf{last}(r)) = u_i = \mathsf{last}(\pi_i(r))$ by the induction hypothesis. By Definition 18 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 $\pi_i(f) = g_i \in \mathbf{G}_i(u_i)$. Since $\langle a, \, \vec{t} \rangle \in \mathsf{Supp}(f)$, it must be that $\langle a, t_i \rangle \in \mathsf{Supp}(g_i) = \mathsf{Supp}(\pi_i(f))$. Moreover, $\pi_i(\mathsf{last}(q)) = t_i = \mathsf{last}(\pi_i(q))$.

- $i \neq j$ and $a \in I_i$. Then by Definition 18 we have $\pi_{a,i}(f) \in \mathbf{R}_i(u_i, a)$. Since $\langle a, \vec{t} \rangle \in \mathsf{Supp}(f)$, it must be that $t_i \in \mathsf{Supp}(\pi_{a,i}(f))$. Moreover, $\pi_i(\mathsf{last}(q)) = t_i = \mathsf{last}(\pi_i(q))$.
- $i \neq j$ and $a \notin I_i$. Then by Definition 18 we have $\pi_{a,i}(f) = \mathsf{Dirac}(u_i)$. Since $\langle a, \vec{t} \rangle \in \mathsf{Supp}(\mu)$, it must be that $t_i = u_i$. Then $\pi_i(\mathsf{last}(q)) = t_i = u_i = \mathsf{last}(\pi_i(r)) = \mathsf{last}(\pi_i(q))$. □

We are now ready to consider composition of I/O schedulers for switched PIOAs.

Definition 30 *Let $\{A_i \mid 1 \leq i \leq n\}$ be a set of pairwise compatible switched PIOAs and let E 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 E. Let $r \in \mathsf{Bran}(E)$ be given and let $\vec{s}$ denote* $\mathsf{last}(r)$.

- *If* $\mathsf{active}_E(\vec{s}) = 1$*, then* $\sigma(r, a) := \perp$ *for all* $a \in I_E$.
- *If* $\mathsf{active}_E(\vec{s}) = 0$*, then for all* $a \in I_E$*,* $\sigma(r, a) := \prod_{i=1}^{n} \mu_i$*, where* μ_i *equals* $\mathsf{Dirac}(s_i)$ *whenever* $a \notin I_i$ *and* $\sigma_i(\pi_i(r), a)$ *otherwise.*
- *If* $\mathsf{active}_E(\vec{s}) = 0$*, then* $\rho(r) := \perp$.
- *If* $\mathsf{active}_E(\vec{s}) = 1$*, then* $\rho(r) \neq \perp$ *if and only* $\rho_j(\pi_j(r)) \neq \perp$*, where* j *is the unique index with* $\mathsf{active}_j(s_j) = 1$*. In that case,* $\rho(r)$ *is the bundle* $f = \rho_j(\pi_j(r)) \times \prod_{\langle a,\, i \rangle \in N_j} \mu_{a,i}$*, where* $\mu_{a,i}$ *equals* $\mathsf{Dirac}(s_i)$ *whenever* $a \notin I_i$ *and* $\sigma_i(\pi_i(r), a)$ *otherwise.*

Lemma 31 *The I/O scheduler $\langle \sigma, \rho \rangle$ in Definition 30 is well-defined.*

PROOF. Let $r \in \mathsf{Bran}(E)$ and $a \in I_E$ be given. Let $\vec{s}$ denote $\mathsf{last}(r)$.

First we consider the case where $\mathbf{R}_E(\mathsf{last}(r), a)$ is non-empty. Since E satisfies Axiom (S2), it must be the case that $\mathsf{active}_E(\vec{s}) = 0$ and hence $\mathsf{active}_i(s_i) = 0$ for all i.

By Axiom (S1), $\mathbf{R}_i(s_i, a)$ is non-empty for all $a \in I_i$. By the definition of input schedulers, this implies $\sigma_i(\pi_i(r), a)$ is defined and is in $\mathbf{R}_i(s_i, a)$. By the definition of $\mathbf{R}_E$, we have that $\prod_{i=1}^{n} \mu_i$ is in $\mathbf{R}_E(\vec{s}, a)$. This proves that $\sigma(r, a)$ is in $\mathbf{R}_E(\mathsf{last}(r), a)$ whenever $\mathbf{R}_E(\mathsf{last}(r), a)$ is non-empty.

Now assume that $\mathbf{R}_E(\mathsf{last}(r), a)$ is empty. By Axiom (S1), we may conclude that $\mathsf{active}_E(\vec{s}) = 1$, in which case $\sigma(r, a)$ is by definition undefined for all $a \in I_E$. This completes the proof that σ is a well-defined input scheduler for E.

For the output scheduler ρ, we need to show that $\rho(r) \in \mathbf{G}_E(\mathsf{last}(r))$ whenever $\rho(r)$ is defined. Therefore, we may focus on the case in which $\mathsf{active}_E(\vec{s}) = 1$. By the definition of S_E, there is unique j with $\mathsf{active}_j(s_j) = 1$. Assume without loss that $\rho_j(\pi_j(r))$ is defined. By the definition of output schedulers, $\rho_j(\pi_j(r)) \in \mathbf{G}_j(s_j)$.

Moreover, we know that $\mathsf{active}_i(s_i) = 0$ for all $i \neq j$. Fix $a \in O_E \cup H_E$ and $i \neq j$. By Axiom (S1), $\mathbf{R}_i(s_i, a)$ is non-empty whenever $a \in I_i$. This implies that $\sigma_i(\pi_i(r), a)$ is defined and is in $\mathbf{R}_i(s_i, a)$. Therefore, the family $\{\mu_{a,i}\}_{\langle a,\, i\rangle \in N_j}$ satisfies the conditions in Definition 18 and thus the bundle f generated by $\rho_j(\pi_j(r))$ and $\{\mu_{a,i}\}_{\langle a,\, i\rangle \in N_j}$ is in $\mathbf{G}_E(\mathsf{last}(r))$. This completes the proof that ρ is a well-defined input scheduler for E. □

Notice that Definition 30 and the proof of Lemma 31 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 nondeterminism, 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 nondeterminism.

Extending Definition 30, we have a very natural notion of composition for switched probabilistic systems.

Definition 32 *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 $\mathcal{E} = \langle E,\, \mathcal{T} \rangle$ defined as follows:*

- *the underlying switched PIOA is $E = \|_{i=1}^n A_i$;*
- *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 5. 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 32, it is no longer possible to obtain the schedule depicted in Figure 2. This is because the local output 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.

6 Compositionality

We proceed to state and prove our main theorem: the external behavior semantics for switched probabilistic systems (Definition 16) is compositional with respect to the composition operator introduced in Definition 32.

Theorem 33 *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 14 that likelihood assignments are defined in terms of minimal execution branches. We will start with a pasting result on minimal branches in a parallel composition of switched PIOAs (Section 6.1, Lemma 36). Then, in Section 6.2, we consider pasting results for execution trees and likelihood assignments. That lays sufficient ground for the proof of Theorem 33 in Section 6.3.

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 30).

6.1 Minimal Execution Branches

Lemma 34 below says, when we project a minimal branch in B onto one of its components, the result is always minimal.

Lemma 34 *For every minimal branch r in $\mathsf{Bran}(B)$, both $\pi_1(r)$ and $\pi_2(r)$ are minimal.*

PROOF. Without loss of generality, we consider only $\pi_1(r)$. 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 $\pi_1(r.a.\mu.\vec{t}) = \pi_1(r).a.\pi_1(\mu).t_1$, which is minimal because a is visible.
- $a \notin I_1$. Then $\pi_1(r.a.\mu.\vec{t}) = \pi_1(r)$. Moreover, note that $\mu \in \mathbf{R}_B(\vec{s}, a)$. Therefore, by Axiom (2), we know that $\mathsf{active}_B(\vec{s}) = 0$. This implies

$\mathsf{active}_1(\mathsf{last}(\pi_1(r))) = \mathsf{active}_1(s_1) = 0$. Therefore by Lemma 11 we know $\pi_1(r)$ 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 $\pi_1(r.f.a.\vec{t}) = \pi_1(r).\pi_1(f).a.t_1$, which is minimal because a is visible.
- $a \in I_1$. Then $\pi_1(r.f.a.\vec{t}) = \pi_1(r).a.\pi_{a,1}(f).t_1$, which is minimal because a is visible.
- $a \notin I_1$. Then $\pi_1(r.f.a.\vec{t}) = \pi_1(r)$. 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 $\mathsf{active}_1(\mathsf{last}(\pi_1(r))) = \mathsf{active}_1(s_1) = 0$. Again, by Lemma 11, we know $\pi_1(r)$ is minimal. □

Lemma 35 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.

Lemma 35 *Let $\alpha \in (I_B \cup O_B)^{<\omega}$ be given. Let p be a minimal branch of A_1 such that $\mathsf{tr}(p) = \pi_1(\alpha)$. Similarly for q in A_2. There is a unique minimal branch r of B such that $\pi_1(r) = p$, $\pi_2(r) = q$, 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 $\pi_1(\alpha a)$ and let p denote the unique minimal prefix of p' with trace $\pi_1(\alpha)$. Similarly for $q \sqsubseteq q'$ in A_2. By induction hypothesis, choose a unique minimal branch r such that $\pi_1(r) = p$, $\pi_2(r) = q$, 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 $\pi_2(\alpha) = \pi_2(\alpha a)$. 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 9, we know that $\mathsf{active}_2(\mathsf{last}(q_0)) = 0$. By Lemma 11, 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 9 and Lemma 11 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. □

Finally, Lemma 36 says, given a fixed trace α, there is a bijective correspondence between $\mathsf{tr}^{-1}_{\min}(\alpha)$ in B and the Cartesian product of $\mathsf{tr}^{-1}_{\min}(\pi_1(\alpha))$ in A_1 and $\mathsf{tr}^{-1}_{\min}(\pi_2(\alpha))$ in A_2.

Lemma 36 *Let X denote $\mathsf{tr}^{-1}_{\min}(\alpha)$ in B. Let Y and Z denote $\mathsf{tr}^{-1}_{\min}(\pi_1(\alpha))$ in A_1 and $\mathsf{tr}^{-1}_{\min}(\pi_2(\alpha))$ in A_2, respectively. There exists an isomorphism $\mathsf{zip} : Y \times Z \to X$ whose inverse is $\langle \pi_1, \pi_2 \rangle$.*

PROOF. By Lemma 34 and Lemma 35. □

6.2 Execution Trees and Likelihood Assignments

For the rest of this section, let Q, Q_1 and Q_2 be abbreviations for the execution trees $Q_{\sigma,\rho}, Q_{\sigma_1,\rho_1}$ and Q_{σ_2,ρ_2}, respectively. Similarly, let $\mathbf{L}, \mathbf{L}_1$ and $\mathbf{L}_2$ denote the likelihood assignments $\mathbf{L}_{\sigma,\rho}, \mathbf{L}_{\sigma_1,\rho_1}$ and $\mathbf{L}_{\sigma_2,\rho_2}$, respectively. Lemma 37 below says an execution tree of the parallel composite can be obtained as a pointwise product of the execution trees of the components. Lemma 38 then combines Lemma 36 and Lemma 37 to show the analogous result for likelihood assignments.

Lemma 37 *For every r in $\mathsf{Bran}(B)$, we have $Q(r) = Q_1(\pi_1(r)) \cdot Q_2(\pi_2(r))$.*

PROOF. If r is empty, $Q(r) = 1 = Q_1(\pi_1(r)) \cdot Q_2(\pi_2(r))$.

Consider $r' = r.a.\mu.\vec{t}$ and let $\vec{s}$ denote $\mathsf{last}(r)$. By Definition 17, 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 $\mu_i \neq \sigma_i(\pi_i(r), a)$. Otherwise, c_i is 1. Then we have

$$\begin{aligned}
Q(r') &= Q(r) \cdot \mu(\vec{t}) \cdot c_1 \cdot c_2 && \text{definitions } \sigma, Q\\
&= Q_1(\pi_1(r)) \cdot Q_2(\pi_2(r)) \cdot c_1 \cdot \mu_1(t_1) \cdot c_2 \cdot \mu_2(t_2) && \text{I.H.}\\
&= Q_1(\pi_1(r')) \cdot Q_2(\pi_2(r')) && \text{definitions } Q_1, Q_2
\end{aligned}$$

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 $g_1 \neq \rho_1(\pi_1(r))$ and 1 otherwise. Similarly, define c_2 to be 0 if $a \in I_2$ but $\mu_{a,2} \neq \sigma_2(\pi_2(r), a)$. Otherwise, c_2 is 1. Similar to the previous case, we have

$$\begin{aligned}
&Q(r') \\
&= Q(r) \cdot f(\langle a,\, \vec{t}\rangle) \cdot c_1 \cdot c_2 && \text{definitions } \rho, Q \\
&= Q_1(\pi_1(r)) \cdot Q_2(\pi_2(r)) \cdot c_1 \cdot g_1(\langle a,\, t_1\rangle) \cdot c_2 \cdot \mu_{a,2}(t_2) && \text{definition } f \text{ and I.H.} \\
&= Q_1(\pi_1(r')) \cdot Q_2(\pi_2(r')) && \text{definitions } Q_1, Q_2
\end{aligned}$$

□

Lemma 38 *Let $\alpha \in (I_B \cup O_B)^{<\omega}$ be given. We have $\mathbf{L}(\alpha) = \mathbf{L}_1(\pi_1(\alpha)) \cdot \mathbf{L}_2(\pi_2(\alpha))$.*

PROOF. Let X denote $\mathsf{tr}^{-1}_{\min}(\alpha)$ in B. Let Y and Z denote $\mathsf{tr}^{-1}_{\min}(\pi_1(\alpha))$ in A_1 and $\mathsf{tr}^{-1}_{\min}(\pi_2(\alpha))$ in A_2, respectively. We have

$$\begin{aligned}
\mathbf{L}(\alpha) &= \sum_{r \in X} Q(r) && \text{definition of } \mathbf{L} \\
&= \sum_{r \in X} Q_1(\pi_1(r)) \cdot Q_2(\pi_2(r)) && \text{Lemma 37} \\
&= \sum_{p \in Y, q \in Z} Q_1(p) \cdot Q_2(q) && \text{Lemma 36} \\
&= (\sum_{p \in Y} Q_1(p)) \cdot (\sum_{q \in Z} Q_2(q)) && \text{factorization} \\
&= \mathbf{L}_1(\pi_1(\alpha)) \cdot \mathbf{L}_2(\pi_2(\alpha)). && \text{definition of } \mathbf{L}_1 \text{ and } \mathbf{L}_2
\end{aligned}$$

□

6.3 Main Proof

PROOF. [Theorem 33] 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 $\mathbf{L} = \mathsf{tr}(Q_{\sigma,\rho})$. 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 $\mathsf{tr}(Q_{\sigma_A,\rho_A})$ and $\mathsf{tr}(Q_{\sigma_C,\rho_C})$, 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 $\mathbf{L}_D := \mathsf{tr}(Q_{\sigma_D,\rho_D}) = \mathbf{L}_A$. 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 $\mathsf{tr}(Q_{\sigma',\rho'})$.

Now, let I denote $I_{A\|C} = I_{D\|C}$ and O denote $O_{A\|C} = O_{D\|C}$. Applying Lemma 38, we have for all $\alpha \in (I \cup O)^{<\omega}$, $\mathbf{L}(\alpha) = \mathbf{L}_A(\pi_A(\alpha)) \cdot \mathbf{L}_C(\pi_C(\alpha))$. Since $\mathcal{A}$ and $\mathcal{D}$ have the same external signature, we know that $\pi_A(\alpha) = \pi_D(\alpha)$. Moreover, by the choice of $\langle \sigma_D, \rho_D \rangle$, we have $\mathbf{L}_A = \mathbf{L}_D$. Hence $\mathbf{L}_A(\pi_A(\alpha)) = \mathbf{L}_D(\pi_D(\alpha))$.

Applying Lemma 38 again, we have

$$\mathbf{L}(\alpha) = \mathbf{L}_A(\pi_A(\alpha)) \cdot \mathbf{L}_C(\pi_C(\alpha)) = \mathbf{L}_D(\pi_D(\alpha)) \cdot \mathbf{L}_C(\pi_C(\alpha)) = \mathbf{L}'(\alpha).$$

This proves that $\mathbf{L} = \mathbf{L}' \in \mathsf{ExtBeh}(\mathcal{D} \parallel \mathcal{C})$. □

7 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 $\bigcup_{i \in \mathcal{I}} \{\mathsf{go}_i, \mathsf{done}_i\}$. We restrict our attention to controllable automata, defined as follows.

Definition 39 *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, and then it must return the activity token by executing a control output action. These can be seen as *fairness* conditions, so that none of the components are allowed to retain the activity token indefinitely.

To compose a set of (pairwise compatible) controllable automata, we use an arbiter automaton, which models either uncertainties in the parallel environment or low-level protocols that specify the exact ordering of events.

Definition 40 *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 below.

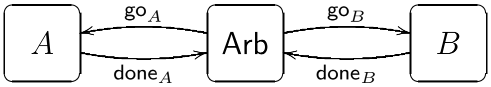

Fig. 10. Arbitrated Composition

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 1.2), 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 40. This determines the observational power of the arbiter, that is, 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 arbiters that ignore parameters of write-related actions.

8 Conclusions and Future Work

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 nondeterministic choices are eliminated. Since the choices between parallel components are often considered nondeterministic, scheduling directly affects the semantics 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 paper 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,CSV06] 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.

Acknowledgment

We wish to thank three anonymous referees for the current issue of *Theoretical Computer Science* who read our paper very carefully and gave valuable

comments.

References

[AB04] Y. Aumann and M.A. Bender. Efficient low-contention asynchronous consensus with the value-oblivious adversary scheduler. *Distributed Computing*, 2004. Accepted in 2004.

[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.

[AM99] J.H. Anderson and M. Moir. Wait-free synchronization in multiprogrammed systems: integrating priority-based and quantum-based scheduling. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 123–132, 1999.

[Asp00] J. Aspnes. Fast deterministic consensus in a noisy environment. In *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 299–309, 2000.

[Asp03] J. Aspnes. Randomized protocols for asynchronous consensus. *Distributed Computing*, 16(2-3):165–175, 2003.

[BPW04] M. Backes, B. Pfitzmann, and M. Waidner. Secure asynchronous reactive systems. Cryptology ePrint Archive Report 2004/082, 2004.

[Can01] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computing*, pages 136–145, 2001.

[CH05] L. Cheung and M. Hendriks. Causal dependencies in parallel composition of stochastic processes. Technical Report ICIS-R05020, Institute for Computing and Information Sciences, University of Nijmegen, 2005.

[Cha96] T.D. Chandra. Polylog randomized wait-free consensus. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, pages 166–175, 1996.

[CLSV04a] L. Cheung, N.A. Lynch, R. Segala, and F.W. Vaandrager. Switched probabilistic i/o automata. In *Proceedings First International Colloquium on Theoretical Aspects of Computing (ICTAC2004)*, Lecture Notes in Computer Science. Springer-Verlag, 2004.

[CLSV04b] L. Cheung, N.A. Lynch, R. Segala, and F.W. Vaandrager. Switched probabilistic i/o automata. Technical Report NIII-R0437, University of Nijmegen, September 2004.

[CSV06] L. Cheung, M.I.A. Stoelinga, and F.W. Vaandrager. A testing scenario for probabilistic processes. Technical Report ICIS-R06002, ICIS, Radboud University Nijmegen, January 2006.

[dAHJ01] L. de Alfaro, T.A. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In K.G. Larsen and M. Nielsen, editors, *Proceedings CONCUR 01*, volume 2154 of *Lecture Notes in Computer Science*, pages 351–365. Springer-Verlag, 2001.

[DEP02] J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labeled markov processes. *Information and Computaton*, 179(2):163–193, 2002.

[DGRV00] M.C.A. Devillers, W.O.D. Griffioen, J.M.T. Romijn, and F.W. Vaandrager. Verification of a leader election protocol - formal methods applied to IEEE 1394. *Formal Methods in System Design*, 16(3), 2000.

[DHK98] P. D'Argenio, H. Hermanns, and J.-P. Katoen. On generative parallel composition. In *Proceedings PROBMIV'98*, volume 22 of *ENTCS*, pages 105–122, 1998.

[Hav98] B.R. Haverkort. *Performance of Computer Communication Systems.* John Wiley & Sons, Ltd, 1998.

[JLY01] B. Jonsson, K.G. Larsen, and W. Yi. *Handbook of Process Algebras*, chapter Probabilistic extensions of process algebras. Elsevier, 2001.

[KS76] J.G. Kennedy and J.L. Snell. *Finite Markov Chains.* Springer-Verlag, New York, 1976.

[LS91] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 91:1–28, 1991.

[LSS94] N.A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proceedings of the 13th Annual ACM Symposium on the Principles of Distributed Computing*, pages 314–323, 1994.

[LSV03] N.A. Lynch, R. Segala, and F.W. Vaandrager. Compositionality for probabilistic automata. In R. Amadio and D. Lugiez, editors, *Proceedings 14th International Conference on Concurrency Theory (CONCUR 2003)*, volume 2761 of *Lecture Notes in Computer Science*, pages 208–221. Springer-Verlag, 2003.

[LT89] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[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(3):155–186, 2000.

[SAGG$^+$93] J. Søgaard-Andersen, S. Garland, J. Guttag, N. Lynch, and A. Pogosyants. Computer-assisted simulation proofs. In *Proceedings of the 4th Conference on Computer Aided Verification, CAV'93*, pages 305–319, 1993.

[SdV04] A. Sokolova and E.P. de Vink. Probabilistic automata: system types, parallel composition and comparison. In *Validation of Stochastic Systems*, volume 2925 of *Lecture Notes in Computer Science*, pages 1–43. Springer-Verlag, 2004.

[Seg95] R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, June 1995. Available as Technical Report MIT/LCS/TR-676.

[SL95] R. Segala and N.A. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.

[Ste94] W.J. Stewart. *Introduction to the Numerical Solutions of Markov Chains*. Princeton University Press, 1994.

[SV99] M.I.A. Stoelinga and F.W. Vaandrager. Root contention in IEEE 1394. In J.-P. Katoen, editor, *Proceedings 5th International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems*, volume 1601 of *Lecture Notes in Computer Science*, pages 53–74. Springer-Verlag, 1999.

[SV03] M.I.A. Stoelinga and F.W. Vaandrager. A testing scenario for probabilistic automata. In *Proceedings 30 ICALP*, volume 2719 of *Lecture Notes in Computer Science*, pages 407–418. Springer-Verlag, 2003.

[vGSS95] R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121:59–80, 1995.

[WSS94] S.-H. Wu, S.A. Smolka, and E.W. Stark. Composition and behaviors of probabilistic I/O automata. In B. Jonsson and J. Parrow, editors, *Proceedings CONCUR 94*, volume 836 of *Lecture Notes in Computer Science*, pages 513–528. Springer-Verlag, 1994.

[YL92] W. Yi and K.G. Larsen. Testing probabilistic and nondeterministic processes. *Testing and Verification*, 12:47–61, 1992. Protocol Specification.