Computer Science and Artificial Intelligence Laboratory
Technical Report

MIT-CSAIL-TR-2006-060 September 5, 2006

Task-Structured Probabilistic I/O Automata

Ran Canetti,, Ling Cheung,, Dilsun Kaynar,, Moses Liskov,, Nancy Lynch,, Olivier Pereira,, and Roberto Segala

massachusetts institute of technology, cambridge, ma 02139 usa — www.csail.mit.edu

Task-Structured Probabilistic I/O Automata *

Ran Canetti
MIT and IBM TJ Watson Research Center

Ling Cheung
Radboud University of Nijmegen

Dilsun Kaynar
MIT

Moses Liskov
College of William and Mary

Nancy Lynch
MIT

Olivier Pereira
Université catholique de Louvain

Roberto Segala
University of Verona

August 31, 2006

Abstract

Modeling frameworks such as Probabilistic I/O Automata (PIOA) and Markov Decision Processes permit both probabilistic and nondeterministic choices. In order to use such frameworks to express claims about probabilities of events, one needs mechanisms for resolving nondeterministic choices. For PIOAs, nondeterministic choices have traditionally been resolved by schedulers that have perfect information about the past execution. However, such schedulers are too powerful for certain settings, such as cryptographic protocol analysis, where information must sometimes be hidden.

Here, we propose a new, less powerful nondeterminism-resolution mechanism for PIOAs, consisting of *tasks* and *local schedulers*. Tasks are equivalence classes of system actions that are scheduled by oblivious, global task sequences. Local schedulers resolve nondeterminism within system components, based on local information only. The resulting task-PIOA framework yields simple notions of external behavior and implementation, and supports simple compositionality results. We also define a new kind of simulation relation, and show it to be sound for proving implementation. We illustrate the potential of the task-PIOA framework by outlining its use in verifying an Oblivious Transfer protocol.

*This report presents an extension of the task-PIOA theory first introduced in [CCK+05, CCK+06d]. This extension is used in [CCK+06e, CCK+06c] to carry out a computational analysis of an Oblivious Transfer protocol. An earlier version of the current report appears as [CCK+06a] and an extended abstract appears as [CCK+06b].

1 Introduction

The *Probabilistic I/O Automata (PIOA)* modeling framework [Seg95, SL95] is a simple combination of I/O Automata [LT89] and Markov Decision Processes (MDP) [Put94]. As demonstrated in [LSS94, SV99, PSL00], PIOAs are well suited for modeling and analyzing distributed algorithms that use randomness as a computational primitive. In this setting, distributed processes use random choices to break symmetry, in solving problems such as choice coordination [Rab82] and consensus [BO83, AH90]. Each process is modeled as an automaton with random transitions, and an entire protocol is modeled as the parallel composition of process automata and automata representing communication channels.

This modeling paradigm combines nondeterministic and probabilistic choices in a natural way. Nondeterminism is used here for modeling uncertainties in the timing of events in highly unpredictable distributed environments. It is also used for modeling distributed algorithms at high levels of abstraction, leaving many details unspecified. This in turn facilitates algorithm verification, because results proved about nondeterministic algorithms apply automatically to an entire family of algorithms, obtained by resolving the nondeterministic choices in particular ways.

In order to formulate and prove probabilistic properties of distributed algorithms, one needs mechanisms for resolving nondeterministic choices. In the randomized distributed setting, the most common mechanism is a *perfect-information* event scheduler, which has access to local state and history of all system components and has unlimited computation power. Thus, probabilistic properties of distributed algorithms are typically asserted with respect to worst-case, adversarial schedulers who can choose the next event based on complete knowledge of the past (e.g., [Seg95, SL95, PSL00]).

One would expect that a similar modeling paradigm, including both probabilistic and nondeterministic choices, would also be useful for modeling *cryptographic protocols*. These are special kinds of distributed algorithms, designed to protect sensitive data when they are transmitted over unreliable channels. Their correctness typically relies on computational assumptions, which say that certain problems cannot be solved by an adversarial entity with bounded computation resources [Gol01]. However, a major problem with this extension is that the perfect-information scheduler mechanism used for distributed algorithms is too powerful for use in the cryptographic setting. A scheduler that could see all information about the past would, in particular, see "secret" information hidden in the states of non-corrupted protocol participants, and be able to "divulge" this information to corrupted participants, e.g., by encoding it in the order in which it schedules events.

In this paper, we present *task-PIOAs*, an adaptation of PIOAs, that has new, less powerful mechanisms for resolving nondeterminism. Task-PIOAs are suitable for modeling and analyzing cryptographic protocols; they may also be useful for other kinds of distributed systems in which the perfect information assumption is unrealistically strong.

Task-PIOAs: A *task-PIOA* is simply a PIOA augmented with a partition of non-input actions into equivalence classes called *tasks*, as in the original I/O automata framework of Lynch and Tuttle [LT89]. A task is typically a set of related actions, for example, all the actions of a cryptographic protocol that send a round 1 message. Tasks are units of scheduling, as for I/O automata; they are scheduled by simple oblivious, global *task schedule* sequences. We define notions of *external behavior* and *implementation* for task-PIOAs, based on the trace distribution semantics proposed by Segala [Seg95]. We define parallel composition in the obvious way and show that our implementation relation is compositional.

We also define a new type of *simulation relation*, which incorporates tasks, and prove that it is sound for proving implementation relationships between task-PIOAs. This new relation differs from simulation relations studied earlier [SL95, LSV03], in that it relates probability measures rather than states. In many cases, including our work on cryptographic protocols (see below), tasks alone suffice for resolving nondeterminism. However, for extra expressive power, we define a second mechanism, *local schedulers*, which can be used to resolve nondeterminism within system components, based on local information only. This mechanism is based on earlier work in [CLSV].

Cryptographic protocols: In [CCK$^+$06e], we apply the task-PIOA framework to analyze an Oblivious

Transfer (OT) protocol of Goldreich, et al. [GMW87]. That analysis requires defining extra structure for task-PIOAs, in order to express issues involving computational limitations. Thus, we define notions such as *time-bounded task-PIOAs*, and *approximate implementation with respect to time-bounded environments*. We use these, for example, to express computational hardness assumptions for cryptographic primitives. Details are beyond the scope of this paper, but we outline our approach in Section 5.

Adversarial scheduling: The standard scheduling mechanism in the cryptographic community is an *adversarial scheduler*, namely, a resource-bounded algorithmic entity that determines the next move adaptively, based on its own view of the computation so far. This is weaker than the *perfect-information scheduler* used for distributed algorithms, which have access to local state and history of all components and have unlimited computation power. It is however stronger than our notion of global task schedule sequences, which are essentially *oblivious schedulers* that fix the entire schedule of tasks nondeterministically in advance.

In order to capture the adaptivity of adversarial schedulers within our framework, we separate scheduling concerns into two parts. The adaptive adversarial scheduler is modeled as a system component, for example, a message delivery service that can eavesdrop on the communications and control the order of message delivery. Such a system component has access to partial information about the execution: it sees information that other components communicate to it during execution, but not "secret information" that these components hide. On the other hand, basic scheduling choices are resolved by a task schedule sequence, chosen nondeterministically in advance. These tasks are equivalence classes of actions, independent of actual choices that are determined during the execution. We believe this separation is conceptually meaningful: The high-level adversarial scheduler is responsible for choices that are essential in security analysis, such as the ordering of message deliveries. The low-level schedule of tasks resolves inessential choices. For example, in the OT protocol, both the transmitter and receiver make random choices, but it is inconsequential which does so first.

Related work: The literature contains numerous models that combine nondeterministic and probabilistic choices (see [SdV04] for a survey). However, few tackle the issue of partial-information scheduling, as we do. Exceptions include [CH05], which models local-oblivious scheduling, and [dA99], which uses partitions on the state space to obtain partial-information schedules. The latter is essentially within the framework of *partially observable MDPs (POMDPs)*, originally studied in the context of reinforcement learning [KLC98]. None of these accounts deal with partial information aspects of (parameterized) actions, therefore they are not suitable in a cryptographic setting.

Our general approach to cryptographic protocol verification was directly inspired by the Interactive Turing Machine (ITM) framework used in [Can01]. There, participants in a protocol are modeled as ITMs and messages as bit strings written on input and output tapes. ITMs are purely probabilistic, and scheduling nondeterminism is resolved using predefined rules. In principle, this framework could be used to analyze cryptographic protocols rigorously, including computational complexity issues; typical correctness arguments reduce the correctness of a protocol to assumptions about its underlying cryptographic primitives. However, complete analysis of protocols in terms of Turing machines is impractical, because it involves too many low-level machine details. Indeed, in the computational cryptography community, protocols are typically described using an informal high-level language, and proof sketches are given in terms of the informal protocol descriptions. We aim to provide a framework in which proofs in the ITM style can be carried out formally, at a high level of abstraction. Also, we aim to exploit the benefits of nondeterminism to a greater extent than the ITM approach.

Several other research groups have added features for computational cryptographic analysis to conventional abstract concurrency modeling frameworks such as process algebras and variants of PIOAs [LMMS98, PW00, PW01, BPW04, MMS03, MRST06]. However, the semantic foundations of concurrent computation used in these papers differ from our task-PIOA framework in some fundamental ways.

Backes et al. [PW01, BPW04] use a network of interrupt-driven probabilistic state machines, with special "buffer" machines to capture message delays, and special "clock ports" to control the scheduling of message delivery. Each individual machine is purely probabilistic; that is, it is fully-specified up to inputs and/or random choices during execution. Given a closed system of such machines with no further inputs, a sequential activation scheme is used to define a unique probabilistic run for each possible initial state of the system. This scheme relies on the presence of a "master scheduler", which is activated by default if no other machine is active.

Thus, in order to capture nondeterministic choices using the framework of Backes et al., one must associate explicit inputs to each schedulable event and then quantify over different machines that provide these scheduling inputs. This deeply contrasts our treatment of nondeterminism, where nondeterministic choices may be present even in closed task-PIOAs and we quantify over task schedules to capture the possible ways of resolving these choices. As it turns out, such a technical difference in the underlying frameworks has some important consequences for security definitions. Namely, in the reactive simulatability definitions of Backes et al., the user and adversary are fixed only *after* all other machines are determined. In essence, this allows the worst possible adversary for every schedule of the system. On the other hand, in our security definitions [CCK$^+$06c, CCK$^+$06e], the environment and adversary are fixed *before* the task schedules. Therefore, we consider instead the worst possible schedule for each given adversary.

On this issue of concurrency and nondeterminism, our task-PIOA framework is more closely related to *PPC*, the process algebraic framework of Mitchell et al[1]. In particular, processes with nondeterministic choices are definable in PPC using the parallel operator and, in the semantics given in [MRST06], a scheduler function selects probabilistically an action label from a set of available actions. Typically, action labels in PPC correspond to the types of protocol messages, as opposed to the messages themselves. This is similar to our distinction between tasks and actions. However, our task schedules are oblivious sequences of tasks, whereas the scheduling functions of [MRST06] are (partially) state-dependent.

The PPC framework differs from our task-PIOA framework in another respect, namely, the use of observational equivalence and probabilistic bisimulation as the main semantic relations. Both of these are symmetric relations, whereas our implementation and simulation relations are asymmetric, expressing the idea that a system P can emulate another system Q but the converse is not necessarily true. The asymmetry of our definitions arises from our quantification over schedules: we assert that "for every schedule of P, there is a schedule of Q that yields equivalent behavior". This is analogous to the traditional formulation for non-probabilistic systems, where implementation means that "every behavior of P is a behavior of Q", but not necessarily vice versa. Experience in the concurrency community shows that such asymmetry can be used to make specifications more simple, by keeping irrelevant details unspecified. At the same time, it produces correctness guarantees that are more general, because correctness is preserved no matter how an implementer chooses to fill in the unspecified details.

Roadmap: Section 2 presents required basic mathematical notions, including definitions and basic results for PIOAs. Some detailed constructions appear in Appendix A. Section 3 defines task-PIOAs, task schedules, composition, and implementation, and presents a simple, fundamental compositionality result. Section 4 presents our simulation relation and its soundness theorem. Section 5 summarizes our OT protocol case study. Section 6 discusses local schedulers, and concluding discussions follow in Section 7.

2 Mathematical Preliminaries

2.1 Sets, functions etc.

We write $\mathbb{R}^{\geq 0}$ and $\mathbb{R}^{+}$ for the sets of nonnegative real numbers and positive real numbers, respectively.

Let X be a set. We denote the set of finite sequences and infinite sequences of elements from X by X^* and X^ω, respectively. If ρ is a sequence then we use $|\rho|$ to denote the length of ρ. We use λ to denote the empty sequence (over any set).

If $\rho \in X^*$ and $\rho' \in X^* \cup X^\omega$, then we write $\rho \frown \rho'$ for the concatentation of the sequences ρ and ρ'. Sometimes, when no confusion seems likely, we omit the $\frown$ symbol, writing just $\rho\rho'$.

[1]Although the authors have also developed a sequential version of PPC [DKMR05], with a semantics akin to the framework of Backes et al.

2.2 Probability measures

In this section, we first present basic definitions for probability measures. Then, we define three operations involving probability measures: *flattening*, *lifting*, and *expansion*; we will use these in Section 4 to define our new kind of simulation relation. These three operations have been previously defined in, for example, [LSV03].

2.2.1 Basic definitions

A σ-field over a set X is a set $\mathcal{F} \subseteq 2^X$ that contains the empty set and is closed under complement and countable union. A pair $(X, \mathcal{F})$ where $\mathcal{F}$ is a σ-field over X, is called a *measurable space*. A measure on a measurable space $(X, \mathcal{F})$ is a function $\mu : \mathcal{F} \rightarrow [0, \infty]$ that is countably additive: for each countable family $\{X_i\}_i$ of pairwise disjoint elements of $\mathcal{F}$, $\mu(\cup_i X_i) = \sum_i \mu(X_i)$. A *probability measure* on $(X, \mathcal{F})$ is a measure on $(X, \mathcal{F})$ such that $\mu(X) = 1$. A *sub-probability measure* on $(X, \mathcal{F})$ is a measure on $(X, \mathcal{F})$ such that $\mu(X) \leq 1$.

A *discrete probability measure* on a set X is a probability measure μ on $(X, 2^X)$, such that, for each $C \subseteq X$, $\mu(C) = \sum_{c \in C} \mu(\{c\})$. A *discrete sub-probability measure* on a set X, is a sub-probability measure μ on $(X, 2^X)$, such that for each $C \subseteq X$, $\mu(C) = \sum_{c \in C} \mu(\{c\})$. We define $\mathsf{Disc}(X)$ and $\mathsf{SubDisc}(X)$ to be, respectively, the set of discrete probability measures and discrete sub-probability measures on X. In the sequel, we often omit the set notation when we refer to the measure of a singleton set.

A *support* of a probability measure μ is a measurable set C such that $\mu(C) = 1$. If μ is a discrete probability measure, then we denote by $\mathsf{supp}(\mu)$ the set of elements that have non-zero measure (thus $\mathsf{supp}(\mu)$ is a support of μ). We let $\delta(x)$ denote the *Dirac measure* for x, the discrete probability measure that assigns probability 1 to $\{x\}$.

Given two discrete measures μ_1, μ_2 on $(X, 2^X)$ and $(Y, 2^Y)$, respectively, we denote by $\mu_1 \times \mu_2$ the *product measure*, that is, the measure on $(X \times Y, 2^{X \times Y})$ such that $\mu_1 \times \mu_2(x, y) = \mu_1(x) \cdot \mu_2(y)$ for each $x \in X, y \in Y$.

If $\{\rho_i\}_{i \in I}$ is a countable family of measures on $(X, \mathcal{F}_X)$ and $\{p_i\}_{i \in I}$ is a family of non-negative values, then the expression $\sum_{i \in I} p_i \rho_i$ denotes a measure ρ on $(X, \mathcal{F}_X)$ such that, for each $C \in \mathcal{F}_X$, $\rho(C) = \sum_{i \in I} p_i \cdot \rho_i(C)$.

A function $f : X \rightarrow Y$ is said to be measurable from $(X, \mathcal{F}_X) \rightarrow (Y, \mathcal{F}_Y)$ if the inverse image of each element of $\mathcal{F}_Y$ is an element of $\mathcal{F}_X$; that is, for each $C \in \mathcal{F}_Y$, $f^{-1}(C) \in \mathcal{F}_X$. Note that, if $\mathcal{F}_X$ is 2^X, then any function $f : X \rightarrow Y$ is measurable from $(X, \mathcal{F}_X) \rightarrow (Y, \mathcal{F}_Y)$ for any $\mathcal{F}_Y$.

Given measurable f from $(X, \mathcal{F}_X) \rightarrow (Y, \mathcal{F}_Y)$ and a measure μ on $(X, \mathcal{F}_X)$, the function $f(\mu)$ defined on $\mathcal{F}_Y$ by $f(\mu)(C) = \mu(f^{-1}(C))$ for each $C \in \mathcal{Y}$ is a measure on $(Y, \mathcal{F}_Y)$ and is called the *image measure* of μ under f. If $\mathcal{F}_X = 2^X$, $\mathcal{F}_Y = 2^Y$, and μ is a sub-probability measure, then the image measure $f(\mu)$ is a sub-probability satisfying $f(\mu)(Y) = \mu(X)$.

2.2.2 Flattening

In this and the following two subsections, we define our three operations involving probability measures. The first operation, which we call *flattening*, takes a discrete probability measure over probability measures and "flattens" it into a single probability measure.

Definition 2.1 *Let η be a discrete probability measure on $\mathsf{Disc}(X)$. Then the* flattening *of η, denoted by* $\mathsf{flatten}(\eta)$, *is the discrete probability measure on X defined by* $\mathsf{flatten}(\eta) = \sum_{\mu \in \mathsf{Disc}(X)} \eta(\mu)\mu$.

Lemma 2.2 *Let η be a discrete probability measure on $\mathsf{Disc}(X)$ and let f be a function from X to Y. Then* $f(\mathsf{flatten}(\eta)) = \mathsf{flatten}(f(\eta))$.

Proof. Recall that $\mathsf{flatten}(\eta)$ is defined to be $\sum_{\mu \in \mathsf{Disc}(X)} \eta(\mu)\mu$. Using the definition of image measures, it is easy to check that f distributes through the summation, so we have

$$f(\mathsf{flatten}(\eta)) = f(\sum_{\mu \in \mathsf{Disc}(X)} \eta(\mu)\mu) = \sum_{\mu \in \mathsf{Disc}(X)} \eta(\mu) f(\mu) = \sum_{\sigma \in \mathsf{Disc}(Y)} \sum_{\mu \in f^{-1}(\sigma)} \eta(\mu)\sigma.$$

Again by the definition of image measures, we have $f(\eta)(\sigma) = \eta(f^{-1}(\sigma)) = \sum_{\mu \in f^{-1}(\sigma)} \eta(\mu)$. This implies that $f(\mathsf{flatten}(\eta))$ equals $\sum_{\sigma \in \mathsf{Disc}(Y)} f(\eta)(\sigma)\sigma$, which is precisely $\mathsf{flatten}(f(\eta))$. □

Lemma 2.3 *Let $\{\eta_i\}_{i \in I}$ be a countable family of measures on $\mathsf{Disc}(X)$, and let $\{p_i\}_{i \in I}$ be a family of probabilities such that $\sum_{i \in I} p_i = 1$. Then we have $\mathsf{flatten}(\sum_{i \in I} p_i \eta_i) = \sum_{i \in I} p_i \mathsf{flatten}(\eta_i)$.*

Proof. By the definition of flatten and by rearranging sums. □

2.2.3 Lifting

The second operation, which we call *lifting*, takes a relation R between two domains X and Y and "lifts" it to a relation between discrete measures over X and Y. Informally speaking, a measure μ_1 on X is related to a measure μ_2 on Y if μ_2 can be obtained by "redistributing" the probability masses assigned by μ_1, in such a way that relation R is respected.

Definition 2.4 *The* lifting *of R, denoted by $\mathcal{L}(R)$, is the relation from $\mathsf{Disc}(X)$ to $\mathsf{Disc}(Y)$ defined by: $\mu_1\ \mathcal{L}(R)\ \mu_2$ iff there exists a* weighting function $w : X \times Y \to \mathbf{R}^{\geq 0}$ *such that the following hold:*

1. *For each $x \in X$ and $y \in Y$, $w(x, y) > 0$ implies $x\ R\ y$.*
2. *For each $x \in X$, $\sum_{y \in Y} w(x, y) = \mu_1(x)$.*
3. *For each $y \in Y$, $\sum_{x \in X} w(x, y) = \mu_2(y)$.*

2.2.4 Expansion

Finally, we define our third operation, called *expansion*. Expansion is defined in terms of flattening and lifting, and is used directly in our new definition of simulation relations. The *expansion* operation takes a relation between discrete measures on two domains X and Y, and returns a relation of the same kind that relates two measures whenever they can be decomposed into two $\mathcal{L}(R)$-related measures.

Definition 2.5 *Let R be a relation from $\mathsf{Disc}(X)$ to $\mathsf{Disc}(Y)$. The* expansion *of R, denoted by $\mathcal{E}(R)$, is a relation from $\mathsf{Disc}(X)$ to $\mathsf{Disc}(Y)$. It is defined by: $\mu_1\ \mathcal{E}(R)\ \mu_2$ iff there exist two discrete measures η_1 and η_2 on $\mathsf{Disc}(X)$ and $\mathsf{Disc}(Y)$, respectively, such that the following hold:*

1. $\mu_1 = \mathsf{flatten}(\eta_1)$.
2. $\mu_2 = \mathsf{flatten}(\eta_2)$.
3. $\eta_1\ \mathcal{L}(R)\ \eta_2$.

Informally speaking, we enlarge R by adding pairs of measures that can be "decomposed" into weighted sums of measures, in such a way that the weights can be "redistributed" in an R-respecting manner. Taking this intuition one step further, the following lemma provides a useful characterization of the expansion relation.

Lemma 2.6 *Let R be a relation on $\mathsf{Disc}(X) \times \mathsf{Disc}(Y)$. Then $\mu_1\ \mathcal{E}(R)\ \mu_2$ iff there exists a countable index set I, a discrete probability measure p on I, and two collections of probability measures, $\{\mu_{1,i}\}_I$ and $\{\mu_{2,i}\}_I$, such that*

1. $\mu_1 = \sum_{i \in I} p(i)\mu_{1,i}$.

2. $\mu_2 = \sum_{i \in I} p(i)\mu_{2,i}$.

3. For each $i \in I$, $\mu_{1,i}\ R\ \mu_{2,i}$.

Proof. Suppose that $\mu_1\ \mathcal{E}(R)\ \mu_2$, and let η_1, η_2 and w be the measures and weighting function used in the definition of $\mathcal{E}(R)$. Let $\{(\mu_{1,i}, \mu_{2,i})\}_{i \in I}$ be an enumeration of the pairs for which $w(\mu_{1,i}, \mu_{2,i}) > 0$, and let $p(i)$ be $w(\mu_{1,i}, \mu_{2,i})$. Then p, $\{(\mu_{1,i})\}_{i \in I}$, and $\{(\mu_{2,i})\}_{i \in I}$ satisfy Items 1, 2, and 3.

Conversely, given p, $\{(\mu_{1,i})\}_{i \in I}$, and $\{(\mu_{2,i})\}_{i \in I}$, we define $\eta_1(\mu)$ to be the sum $\sum_{i|\mu=\mu_{1,i}} p(i)$ and $\eta_2(\mu)$ to be $\sum_{i|\mu=\mu_{2,i}} p(i)$. Moreover, define $w(\mu'_1, \mu'_2)$ to be $\sum_{i|\mu'_1=\mu_{1,i}, \mu'_2=\mu_{2,i}} p(i)$. Then, η_1, η_2 and w satisfy the properties required in the definition of $\mathcal{E}(R)$. □

The next, rather technical lemma gives us a sufficient condition for showing that a pair of functions f and g preserve the relation $\mathcal{E}(R)$; that is, if $\mu_1\ \mathcal{E}(R)\ \mu_2$, then $f(\mu_1)\ \mathcal{E}(R)\ f(\mu_2)$. The required condition is that, when μ_1 and μ_2 are decomposed into weighted sums of measures as in the definition of $\mu_1\ \mathcal{E}(R)\ \mu_2$, f and g convert each pair (ρ_1, ρ_2) of R-related probability measures to $\mathcal{E}(R)$-related probability measures. We will use this lemma in the soundness proof for our new kind of simulation relation (Lemma 4.5), where the two functions f and g apply corresponding sequences of tasks to corresponding measures on executions.

Lemma 2.7 *Let* R *be a relation from* $\mathsf{Disc}(X)$ *to* $\mathsf{Disc}(Y)$*, and let* f, g *be two endo-functions on* $\mathsf{Disc}(X)$ *and* $\mathsf{Disc}(Y)$*, respectively. Suppose that* f *distributes over convex combinations of measures; that is, for each countable family* $\{\rho_i\}_i$ *of discrete measures on* X *and each countable family of probabilities* $\{p_i\}_i$ *such that* $\sum_i p_i = 1$, $f(\sum_i p_i\rho_i) = \sum_i p_i f(\rho_i)$*. Similarly for* g*. Let* μ_1 *and* μ_2 *be measures on* X *and* Y*, respectively, such that* $\mu_1\ \mathcal{E}(R)\ \mu_2$*. Let* η_1, η_2*, and* w *be a pair of measures and a weighting function witnessing the fact that* $\mu_1\ \mathcal{E}(R)\ \mu_2$*. Suppose further that, for any two distributions* $\rho_1 \in \mathsf{supp}(\eta_1)$ *and* $\rho_2 \in \mathsf{supp}(\eta_2)$ *with* $w(\rho_1, \rho_2) > 0$*, we have* $f(\rho_1)\ \mathcal{E}(R)\ g(\rho_2)$*.*
Then $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$*.*

Proof. Let W denote the set of pairs (ρ_1, ρ_2) such that $w(\rho_1, \rho_2) > 0$. Note that, by the definition of lifting, $(\rho_1, \rho_2) \in W$ implies $\rho_1 \in \mathsf{supp}(\eta_1)$ and $\rho_2 \in \mathsf{supp}(\eta_2)$. Therefore, by assumption, we have $f(\rho_1)\ \mathcal{E}(R)\ g(\rho_2)$ whenever $(\rho_1, \rho_2) \in W$.

Now, for each $(\rho_1, \rho_2) \in W$, choose a pair of measures $(\eta_1)_{\rho_1,\rho_2}$, $(\eta_2)_{\rho_1,\rho_2}$ and a weighting function $w_{\rho_1\rho_2}$ as guaranteed by the definition of $f(\rho_1)\ \mathcal{E}(R)\ g(\rho_2)$. Let $\eta'_1 = \sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)(\eta_1)_{\rho_1,\rho_2}$ and let $\eta'_2 = \sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)(\eta_2)_{\rho_1,\rho_2}$. Let $w' = \sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)w_{\rho_1,\rho_2}$.

We show that η'_1, η'_2, and w' satisfy the conditions for $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$.

1. $f(\mu_1) = \mathsf{flatten}(\eta'_1)$.

   By the definition of η'_1, $\mathsf{flatten}(\eta'_1) = \mathsf{flatten}(\sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)(\eta_1)_{\rho_1,\rho_2})$. By Lemma 2.3, this is in turn equal to $\sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)\mathsf{flatten}((\eta_1)_{(\rho_1,\rho_2)})$. By the choice of $(\eta_1)_{(\rho_1,\rho_2)}$, we know that $\mathsf{flatten}((\eta_1)_{(\rho_1,\rho_2)}) = f(\rho_1)$, so we obtain that $\mathsf{flatten}(\eta'_1) = \sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)f(\rho_1)$.

   We claim that the right side is equal to $f(\mu_1)$: Since $\mu_1 = \mathsf{flatten}(\eta_1)$, by the definition of flattening, $\mu_1 = \sum_{\rho_1\in\mathsf{Disc}(X)} \eta_1(\rho_1)\rho_1$. Then, by distributivity of f, $f(\mu_1) = \sum_{\rho_1\in\mathsf{Disc}(X)} \eta_1(\rho_1)f(\rho_1)$. By definition of lifting, $\eta_1(\rho_1) = \sum_{\rho_2\in\mathsf{Disc}(Y)} w(\rho_1, \rho_2)$.
   Therefore, $f(\mu_1) = \sum_{\rho_1\in\mathsf{Disc}(X)} \sum_{\rho_2\in\mathsf{Disc}(Y)} w(\rho_1, \rho_2)f(\rho_1)$, and this last expression is equal to $\sum_{(\rho_1,\rho_2)\in W} w(\rho_1, \rho_2)f(\rho_1)$, as needed.

2. $g(\mu_2) = \mathsf{flatten}(\eta'_2)$.

   Analogous to the previous case.

3. $\eta'_1\ \mathcal{L}(R)\ \eta'_2$ using w' as a weighting function.

   We verify that w' satisfies the three conditions in the definition of a weighting function:

(a) Let ρ'_1, ρ'_2 be such that $w'(\rho'_1, \rho'_2) > 0$. Then, by definition of w', there exists at least one pair $(\rho_1, \rho_2) \in R$ such that $w_{\rho_1,\rho_2}(\rho'_1, \rho'_2) > 0$. Since w_{ρ_1,ρ_2} is a weighting function, $\rho'_1 \; R \; \rho'_2$ as needed.

(b) By the definition of w', we have

$$\begin{aligned}\sum_{\rho'_2 \in \mathsf{Disc}(Y)} w'(\rho'_1, \rho'_2) &= \sum_{\rho'_2 \in \mathsf{Disc}(Y)} \sum_{(\rho_1,\rho_2) \in W} w(\rho_1, \rho_2) w_{\rho_1,\rho_2}(\rho'_1, \rho'_2) \\ &= \sum_{(\rho_1,\rho_2) \in W} \sum_{\rho'_2 \in \mathsf{Disc}(Y)} w(\rho_1, \rho_2) w_{\rho_1,\rho_2}(\rho'_1, \rho'_2) \\ &= \sum_{(\rho_1,\rho_2) \in W} (w(\rho_1, \rho_2) \cdot \sum_{\rho'_2 \in \mathsf{Disc}(Y)} w_{\rho_1,\rho_2}(\rho'_1, \rho'_2)).\end{aligned}$$

Since w_{ρ_1,ρ_2} is a weighting function, we also have $\sum_{\rho'_2 \in \mathsf{Disc}(Y)} w_{\rho_1,\rho_2}(\rho'_1, \rho'_2) = (\eta_1)_{\rho_1,\rho_2}(\rho'_1)$. This implies $\sum_{\rho'_2 \in \mathsf{Disc}(Y)} w'(\rho'_1, \rho'_2)$ equals $\sum_{(\rho_1,\rho_2)} w(\rho_1, \rho_2)(\eta_1)_{\rho_1,\rho_2}(\rho'_1)$, which is precisely $\eta'_1(\rho'_1)$.

(c) Symmetric to the previous case.

□

2.3 Probabilistic I/O Automata

In this subsection, we review basic definitions for Probabilistic I/O Automata.

2.3.1 PIOAs and their executions

A *probabilistic I/O automaton (PIOA)*, $\mathcal{P}$, is a tuple $(Q, \bar{q}, I, O, H, D)$ where:

- Q is a countable set of *states*, with *start state* $\bar{q} \in Q$;
- I, O and H are countable and pairwise disjoint sets of actions, referred to as *input, output and internal (hidden) actions*, respectively; and
- $D \subseteq (Q \times (I \cup O \cup H) \times \mathsf{Disc}(Q))$ is a *transition relation*, where $\mathsf{Disc}(Q)$ is the set of discrete probability measures on Q.

An action a is *enabled* in a state q if $(q, a, \mu) \in D$ for some μ. The set $A := I \cup O \cup H$ is called the *action alphabet* of $\mathcal{P}$. If $I = \emptyset$, then $\mathcal{P}$ is *closed*. The set of *external* actions of $\mathcal{P}$ is $E := I \cup O$, and the set of *locally controlled* actions is $L := O \cup H$.

We assume that $\mathcal{P}$ satisfies the following conditions:

- *Input enabling:* For every state $q \in Q$ and input action $a \in I$, a is enabled in q.
- *Transition determinism:* For every $q \in Q$ and $a \in A$, there is at most one $\mu \in \mathsf{Disc}(Q)$ such that $(q, a, \mu) \in D$. If there is exactly one such μ, it is denoted by $\mu_{q,a}$, and we write $\mathtt{tran}_{q,a}$ for the transition $(q, a, \mu_{q,a})$.

A (non-probabilistic) *execution fragment* of $\mathcal{P}$ is a finite or infinite sequence $\alpha = q_0 \; a_1 \; q_1 \; a_2 \; \ldots$ of alternating states and actions, such that:

- If α is finite, then it ends with a state.
- For every non-final i, there is a transition $(q_i, a_{i+1}, \mu) \in D$ with $q_{i+1} \in \mathsf{supp}(\mu)$.

We write $\mathsf{fstate}(\alpha)$ for q_0, and, if α is finite, we write $\mathsf{lstate}(\alpha)$ for the last state of α. We use $\mathsf{Frags}(\mathcal{P})$ (resp., $\mathsf{Frags}^*(\mathcal{P})$) to denote the set of all (resp., all finite) execution fragments of $\mathcal{P}$. An *execution* of $\mathcal{P}$ is an execution fragment beginning from the start state $\bar{q}$. $\mathsf{Execs}(\mathcal{P})$ (resp., $\mathsf{Execs}^*(\mathcal{P})$) denotes the set of all (resp., finite) executions of $\mathcal{P}$.

The *trace* of an execution fragment α, written $\mathsf{trace}(\alpha)$, is the restriction of α to the set of external actions of $\mathcal{P}$. We say that β is a *trace* of $\mathcal{P}$ if there is an execution α of $\mathcal{P}$ with $\mathsf{trace}(\alpha) = \beta$. The symbol $\leq$ denotes the prefix relation on sequences, which applies in particular to execution fragments and traces.

2.3.2 Schedulers and probabilistic executions

Nondeterministic choices in $\mathcal{P}$ are resolved using a *scheduler*:

Definition 2.8 *A* scheduler *for $\mathcal{P}$ is a function $\sigma : \mathsf{Frags}^*(\mathcal{P}) \longrightarrow \mathsf{SubDisc}(D)$ such that $(q, a, \mu) \in \mathsf{supp}(\sigma(\alpha))$ implies $q = \mathsf{lstate}(\alpha)$.*

Thus, σ decides (probabilistically) which transition (if any) to take after each finite execution fragment α. Since this decision is a discrete sub-probability measure, it may be the case that σ chooses to *halt* after α with non-zero probability: $1 - \sigma(\alpha)(D) > 0$.

A scheduler σ and a finite execution fragment α *generate* a measure $\epsilon_{\sigma,\alpha}$ on the σ-field $\mathcal{F}_{\mathcal{P}}$ generated by cones of execution fragments, where the cone $C_{\alpha'}$ of a finite execution fragment α' is the set of execution fragments that have α' as a prefix. The construction of the σ-field is standard and is presented in Appendix A.

Definition 2.9 *The measure of a cone, $\epsilon_{\sigma,\alpha}(C_{\alpha'})$, is defined recursively, as:*

1. *0, if $\alpha' \not\leq \alpha$ and $\alpha \not\leq \alpha'$;*
2. *1, if $\alpha' \leq \alpha$; and*
3. *$\epsilon_{\sigma,\alpha}(C_{\alpha''})\mu_{\sigma(\alpha'')}(a, q)$, if α' is of the form $\alpha''\ a\ q$ and $\alpha \leq \alpha''$. Here, $\mu_{\sigma(\alpha'')}(a, q)$ is defined to be $\sigma(\alpha'')(\mathsf{tran}_{\mathsf{lstate}(\alpha''),a})\mu_{\mathsf{lstate}(\alpha''),a}(q)$, that is, the probability that $\sigma(\alpha'')$ chooses a transition labeled by a and that the new state is q.*

Standard measure theoretic arguments ensure that $\epsilon_{\sigma,\alpha}$ is well-defined. We call the state $\mathsf{fstate}(\alpha)$ the *first state* of $\epsilon_{\sigma,\alpha}$ and denote it by $\mathsf{fstate}(\epsilon_{\sigma,\alpha})$. If α consists of the start state $\bar{q}$ only, we call $\epsilon_{\sigma,\alpha}$ a *probabilistic execution* of $\mathcal{P}$.

Let μ be a discrete probability measure over $\mathsf{Frags}^*(\mathcal{P})$. We denote by $\epsilon_{\sigma,\mu}$ the measure $\sum_\alpha \mu(\alpha)\epsilon_{\sigma,\alpha}$ and we say that $\epsilon_{\sigma,\mu}$ is *generated by* σ and μ. We call the measure $\epsilon_{\sigma,\mu}$ a *generalized probabilistic execution fragment* of $\mathcal{P}$. If every execution fragment in $\mathsf{supp}(\mu)$ consists of a single state, then we call $\epsilon_{\sigma,\mu}$ a *probabilistic execution fragment* of $\mathcal{P}$.

We note that the trace function is a measurable function from $\mathcal{F}_{\mathcal{P}}$ to the σ-field generated by cones of traces. Thus, given a probability measure ϵ on $\mathcal{F}_{\mathcal{P}}$, we define the *trace distribution* of ϵ, denoted $\mathsf{tdist}(\epsilon)$, to be the image measure of ϵ under trace. We extend the $\mathsf{tdist}()$ notation to arbitrary measures on execution fragments of $\mathcal{P}$. We denote by $\mathsf{tdists}(\mathcal{P})$ the set of trace distributions of (probabilistic executions of) $\mathcal{P}$.

Next we present some basic results about probabilistic executions and trace distributions of PIOAs. In particular, Lemmas 2.10-2.14 give some useful equations involving the probabilities of various sets of execution fragments.

Lemma 2.10 *Let σ be a scheduler for PIOA $\mathcal{P}$, μ be a discrete probability measure on finite execution fragments of $\mathcal{P}$, and α be a finite execution fragment of $\mathcal{P}$. Then*

$$\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + \sum_{\alpha' < \alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha).$$

Proof. By definition of $\epsilon_{\sigma,\mu}$, $\epsilon_{\sigma,\mu}(C_\alpha) = \sum_{\alpha'} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha)$. Since, by definition, $\epsilon_{\sigma,\alpha'}(C_\alpha) = 1$ whenever $\alpha \leq \alpha'$, this can be rewritten as

$$\epsilon_{\sigma,\mu}(C_\alpha) = \sum_{\alpha':\alpha\leq\alpha'} \mu(\alpha') + \sum_{\alpha'<\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha).$$

Observe that $\sum_{\alpha':\alpha\leq\alpha'} \mu(\alpha') = \mu(C_\alpha)$. Thus, by substitution, we get the statement of the lemma. □

Lemma 2.11 *Let σ be a scheduler for PIOA $\mathcal{P}$, μ be a discrete probability measure on finite execution fragments of $\mathcal{P}$, and α be a finite execution fragment of $\mathcal{P}$. Then*

$$\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha - \{\alpha\}) + \sum_{\alpha'\leq\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha).$$

Proof. Follows directly from Lemma 2.10 after observing that $\epsilon_{\sigma,\alpha}(C_\alpha) = 1$. □

Lemma 2.12 *Let σ be a scheduler for PIOA $\mathcal{P}$, and μ be a discrete measure on finite execution fragments of $\mathcal{P}$. Let $\alpha = \tilde{\alpha}aq$ be a finite execution fragment of $\mathcal{P}$. Then*

$$\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + (\epsilon_{\sigma,\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\, \sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q).$$

Proof. By Lemma 2.10 and the definitions of $\epsilon_{\sigma,\alpha'}(C_\alpha)$ and $\mu_{\sigma(\tilde{\alpha})}(a, q)$, we have

$$\begin{aligned}\epsilon_{\sigma,\mu}(C_\alpha) &= \mu(C_\alpha) + \sum_{\alpha'<\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_{\tilde{\alpha}})\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q) \\ &= \mu(C_\alpha) + (\sum_{\alpha'<\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_{\tilde{\alpha}}))(\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q)).\end{aligned}$$

Since $\alpha' \leq \tilde{\alpha}$ if and only if $\alpha' < \alpha$, this yields

$$\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + (\sum_{\alpha'\leq\tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_{\tilde{\alpha}}))(\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q)).$$

It suffices to show that $\sum_{\alpha'\leq\tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_{\tilde{\alpha}}) = \epsilon_{\sigma,\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$. But this follows immediately from Lemma 2.11 (with α instantiated as $\tilde{\alpha}$). □

As a notational convention we introduce a new symbol $\perp$ to denote termination. Given scheduler σ and finite execution fragment α, we write $\sigma(\alpha)(\perp)$ for the probability of terminating after α (namely, $1 - \sigma(\alpha)(D)$).

Lemma 2.13 *Let σ be a scheduler for PIOA $\mathcal{P}$, μ be a discrete probability measure on finite execution fragments of $\mathcal{P}$, and α be a finite execution fragment of $\mathcal{P}$. Then*

$$\epsilon_{\sigma,\mu}(\alpha) = (\epsilon_{\sigma,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\}))(\sigma(\alpha)(\perp)).$$

Proof. By definition of $\epsilon_{\sigma,\mu}$, $\epsilon_{\sigma,\mu}(\alpha) = \sum_{\alpha'} \mu(\alpha')\epsilon_{\sigma,\alpha'}(\alpha)$. The sum can be restricted to $\alpha' \leq \alpha$ since for all other α', $\epsilon_{\sigma,\alpha'}(\alpha) = 0$. Then, since for each $\alpha' \leq \alpha$, $\epsilon_{\sigma,\alpha'}(\alpha) = \epsilon_{\sigma,\alpha'}(C_\alpha)\sigma(\alpha)(\perp)$, we derive $\epsilon_{\sigma,\mu}(\alpha) = \sum_{\alpha'\leq\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha)\sigma(\alpha)(\perp)$. Observe that $\sigma(\alpha)(\perp)$ is a constant with respect to α', and thus can be moved out of the sum, yielding $\epsilon_{\sigma,\mu}(\alpha) = (\sum_{\alpha'\leq\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha))(\sigma(\alpha)(\perp))$.

It suffices to show that $\sum_{\alpha'\leq\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_\alpha) = \epsilon_{\sigma,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\})$. But this follows immediately from Lemma 2.11. □

Lemma 2.14 *Let σ be a scheduler for PIOA $\mathcal{P}$, and μ be a discrete probability measure on finite execution fragments of $\mathcal{P}$. Let α be a finite execution fragment of $\mathcal{P}$ and a be an action of $\mathcal{P}$ that is enabled in* $\mathsf{lstate}(\alpha)$. *Then*

$$\epsilon_{\sigma,\mu}(C_{\alpha a}) = \mu(C_{\alpha a}) + (\epsilon_{\sigma,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\}))\, \sigma(\alpha)(\mathsf{tran}_{\alpha,a}).$$

Proof. Observe that $C_{\alpha a} = \cup_q C_{\alpha a q}$, where the cones $C - \alpha a q$ are pairwise . Thus, $\epsilon_{\sigma,\mu}(C_{\alpha a}) = \sum_q \epsilon_{\sigma,\mu}(C_{\alpha a q})$. By Lemma 2.12, the right-hand side is equal to

$$\sum_q \left(\mu(C_{\alpha a q}) + (\epsilon_{\sigma,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\}))\, \sigma(\alpha)(\mathsf{tran}_{\alpha,a})\mu_{\alpha,a}(q)\right).$$

Since $\sum_q \mu(C_{\alpha a q}) = \mu(C_{\alpha a})$ and $\sum_q \mu_{\alpha,a}(q) = 1$, this is in turn equal to

$$\mu(C_{\alpha a}) + (\epsilon_{\sigma,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\}))\, \sigma(\alpha)(\mathsf{tran}_{\alpha,a}).$$

Combining the equations yields the result. □

Finally, we present a lemma about limits of generalized probabilistic execution fragments.

Proposition 2.15 *Let $\epsilon_1, \epsilon_2, \ldots$ be a chain of generalized probabilistic execution fragments of a PIOA $\mathcal{P}$, all generated from the same discrete probability measure μ on finite execution fragments. Then $\lim_{i\to\infty} \epsilon_i$ is a generalized probabilistic execution fragment of $\mathcal{P}$ generated from μ.*

Proof. Let ϵ denote $\lim_{i\to\infty} \epsilon_i$. For each $i \geq 1$, let σ_i be a scheduler such that $\epsilon_i = \epsilon_{\sigma_i,\mu}$, and for each finite execution fragment α, let $p^i_\alpha = \epsilon_{\sigma_i,\mu}(C_\alpha) - \mu(C_\alpha - \{\alpha\})$. For each finite execution α and each action a, let $p^i_{\alpha a} = \epsilon_{\sigma_i,\mu}(C_{\alpha a}) - \mu(C_{\alpha a})$.

By Lemma 2.14, if a is enabled in $\mathsf{lstate}(\alpha)$ then $p^i_\alpha \sigma_i(\alpha)(\mathsf{tran}_{\alpha,a}) = p^i_{\alpha a}$. Moreover, if $p^i_{\alpha a} \neq 0$, then $\sigma_i(\alpha)(\mathsf{tran}_{\alpha,a}) = p^i_{\alpha a}/p^i_\alpha$.

For each finite execution fragment α, let $p_\alpha = \epsilon(C_\alpha) - \mu(C_\alpha - \{\alpha\})$. For each finite execution fragment α and each action a, let $p_{\alpha a} = \epsilon(C_{\alpha a}) - \mu(C_{\alpha a})$. Define $\sigma(\alpha)(\mathsf{tran}_{\alpha,a})$ to be $p_{\alpha a}/p_\alpha$ if $p_\alpha > 0$; otherwise define $\sigma(\alpha)(\mathsf{tran}_{\alpha,a}) = 0$. By definition of ϵ and simple manipulations, $\lim_{i\to\infty} p^i_\alpha = p_\alpha$ and $\lim_{i\to\infty} p^i_{\alpha a} = p_{\alpha a}$. It follows that, if $p_\alpha > 0$, then $\sigma(\alpha)(\mathsf{tran}_{\alpha,a}) = \lim_{i\to\infty} \sigma_i(\alpha)(\mathsf{tran}_{\alpha,a})$.

It remains to show that σ is a scheduler and that $\epsilon_{\sigma,\mu} = \epsilon$. To show that σ is a scheduler, we must show that, for each finite execution fragment α, $\sigma(\alpha)$ is a sub-probability measure. Observe that, for each $i \geq 1$, $\sum_{\mathsf{tran}} \sigma_i(\alpha)(\mathsf{tran}) = \sum_a \sigma_i(\alpha)(\mathsf{tran}_{\alpha a})$. Similarly, $\sum_{\mathsf{tran}} \sigma(\alpha)(\mathsf{tran}) = \sum_a \sigma(\alpha)(\mathsf{tran}_{\alpha a})$. Since each σ_i is a scheduler, it follows that, for each $i \geq 0$, $\sum_a \sigma_i(\alpha)(\mathsf{tran}_{\alpha a}) \leq 1$. Thus,

$$\lim_{i\to\infty} \sum_a \sigma_i(\alpha)(\mathsf{tran}_{\alpha a}) \leq \sum_a \lim_{i\to\infty} \sigma_i(\alpha)(\mathsf{tran}_{\alpha a}) \leq 1.$$

We claim that $\sigma(\alpha)(\mathsf{tran}_{\alpha,a}) \leq \lim_{i\to\infty} \sigma_i(\alpha)(\mathsf{tran}_{\alpha,a})$, which implies that $\sigma(\alpha)(\mathsf{tran}_{\alpha a}) \leq 1$, as needed. To see this claim, we consider two cases: If $p_\alpha > 0$, then as shown earlier, $\sigma(\alpha)(\mathsf{tran}_{\alpha,a}) = \lim_{i\to\infty} \sigma_i(\alpha)(\mathsf{tran}_{\alpha,a})$. On the other hand, if $p_\alpha = 0$, then $\sigma(\alpha)(\mathsf{tran}_{\alpha,a})$ is defined to be zero, so that $\sigma(\alpha)(\mathsf{tran}_{\alpha,a}) = 0 \leq \lim_{i\to\infty} \sigma_i(\alpha)(\mathsf{tran}_{\alpha,a})$.

To show that $\epsilon_{\sigma,\mu} = \epsilon$, we show by induction on the length of a finite execution fragment α that $\epsilon_{\sigma,\mu}(C_\alpha) = \epsilon(C_\alpha)$. For the base case, let α consist of a single state q. By Lemma 2.10, $\epsilon_{\sigma,\mu}(C_q) = \mu(C_q)$, and for each $i \geq 1$, $\epsilon_{\sigma_i,\mu}(C_q) = \mu(C_q)$. Thus, $\epsilon(C_q) = \lim_{i\to\infty} \epsilon_{\sigma_i,\mu}(C_q) = \mu(C_q)$, as needed.

For the inductive step, let $\alpha = \tilde{\alpha} a q$. By Lemma 2.12,

$$\lim_{i\to\infty} \epsilon_{\sigma_i,\mu}(C_\alpha) = \lim_{i\to\infty} \left(\mu(C_\alpha) + (\epsilon_{\sigma_i,\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\, \sigma_i(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q)\right).$$

Observe that the left-hand side is $\epsilon(C_\alpha)$. By algebraic manipulation, the right-hand side becomes

$$\mu(C_\alpha) + \left(\left(\lim_{i\to\infty} \epsilon_{\sigma_i,\mu}(C_{\tilde{\alpha}})\right) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})\right) \left(\lim_{i\to\infty} \sigma_i(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\right) \mu_{\tilde{\alpha},a}(q).$$

By definition of ϵ, $\lim_{i\to\infty} \epsilon_{\sigma_i,\mu}(C_{\tilde{\alpha}}) = \epsilon(C_{\tilde{\alpha}})$, and by inductive hypothesis, $\epsilon(C_{\tilde{\alpha}}) = \epsilon_{\sigma,\mu}(C_{\tilde{\alpha}})$. Therefore,

$$\epsilon(C_\alpha) = \mu(C_\alpha) + (\epsilon_{\sigma,\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})) \left(\lim_{i\to\infty} \sigma_i(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\right) \mu_{\tilde{\alpha},a}(q).$$

Also by Lemma 2.12, we obtain that

$$\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + (\epsilon_{\sigma,\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\,\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q).$$

We claim that the right-hand sides of the last two equations are equal. To see this, consider two cases. First, if $p_{\tilde{\alpha}} > 0$, then we have already shown that $\lim_{i\to\infty}\sigma_i(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a}) = \sigma(\tilde{\alpha}(\mathsf{tran}_{\tilde{\alpha},a}))$. Since these two terms are the only difference between the two expressions, the expressions are equal.

On the other hand, if $p_{\tilde{\alpha}} = 0$, then by definition of $p_{\tilde{\alpha}}$, we get that $\epsilon(C_{\tilde{\alpha}}) = \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$. Then by the induction hypothesis the second terms of the two right-hand sides are both equal to zero, which implies that both expressions are equal to the first term $\mu(C_\alpha)$. Again, the two right-hand sides are equal.

Since the right-hand sides are equal, so are the left-hand sides, that is, $\epsilon_{\sigma,\mu}(C_\alpha) = \epsilon(C_\alpha)$, as needed to complete the inductive step. □

2.3.3 Composition

We define composition of PIOAs as follows.

Definition 2.16 *Two PIOAs $\mathcal{P}_i = (Q_i, \bar{q}_i, I_i, O_i, H_i, D_i)$, $i \in \{1,2\}$, are said to be* compatible *if $A_i \cap H_j = O_i \cap O_j = \emptyset$ whenever $i \neq j$. In that case, we define their* composition *$\mathcal{P}_1 \| \mathcal{P}_2$ to be the PIOA $(Q_1 \times Q_2,\ (\bar{q}_1, \bar{q}_2),\ (I_1 \cup I_2) \setminus (O_1 \cup O_2),\ O_1 \cup O_2,\ H_1 \cup H_2, D)$, where D is the set of triples $((q_1, q_2), a, \mu_1 \times \mu_2)$ such that*

1. *a is enabled in some q_i.*
2. *For every i, if $a \in A_i$ then $(q_i, a, \mu_i) \in D_i$, otherwise $\mu_i = \delta(q_i)$.*

Given a state $q = (q_1, q_2)$ in the composition and $i \in \{1,2\}$, we use $q\lceil\mathcal{P}_i$ to denote q_i. Note that these definitions can be extended to any finite number of PIOAs rather than just two.

2.3.4 Hiding

We define a hiding operation for PIOAs, which hides output actions.

Definition 2.17 *Let $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ be a PIOA and let $S \subseteq O$. Then $\mathsf{hide}(\mathcal{P}, S)$ is the PIOA $\mathcal{P}'$ that is the same as $\mathcal{P}$ except that $O_{\mathcal{P}'} = O_{\mathcal{P}} - S$ and $H_{\mathcal{P}'} = H_{\mathcal{P}} \cup S$.*

3 Task-PIOAs

In this section, we present our definition for task-PIOAs. We introduce task schedules, which are used to generate probabilistic executions. We define composition and hiding operations. We define an implementation relation, which we call $\leq_0$. And finally, we state and prove a simple compositionality result. In the next section, Section 4, we define our new simulation relation for task-PIOAs and prove that it is sound.

3.1 Task-PIOA definition

We now augment the PIOA framework with task partitions, our main mechanism for resolving nondeterminism.

Definition 3.1 *A* task-PIOA *is a pair $\mathcal{T} = (\mathcal{P}, R)$ where*

- *$\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ is a PIOA (satisfying transition determinism).*
- *R is an equivalence relation on the locally-controlled actions ($O \cup H$).*

For clarity, we sometimes write $R_{\mathcal{T}}$ for R.

The equivalence classes of R are called tasks. *A task T is* enabled *in a state q if some $a \in T$ is enabled in q. It is* enabled *in a set S of states provided it is enabled in every $q \in S$.*

Unless otherwise stated, technical notions for task-PIOAs are inherited from those for PIOAs. Exceptions include the notions of probabilistic executions and trace distributions.

For now, we impose the following action-determinism assumption, which implies that tasks alone are enough to resolve all nondeterministic choices. We will remove this assumption when we introduce local schedulers, in Section 6. To make it easier to remove the action-determinism hypothesis later, we will indicate explicitly, before Section 6, where we are using the action-determinism hypothesis.

- *Action determinism:* For every state $q \in Q$ and task $T \in R$, at most one action $a \in T$ is enabled in q.

3.2 Task schedules and the apply function

Definition 3.2 *If $\mathcal{T} = (\mathcal{P}, R)$ is a task-PIOA, then a* task schedule *for $\mathcal{T}$ is any finite or infinite sequence $\rho = T_1 T_2 \ldots$ of tasks in R.*

Thus, a task schedule is *static* (or *oblivious*), in the sense that it does not depend on dynamic information generated during execution. Under the action-determinism assumption, a task schedule can be used to generate a unique probabilistic execution, and hence, a unique trace distribution, of the underlying PIOA $\mathcal{P}$. One can do this by repeatedly scheduling tasks, each of which determines at most one transition of $\mathcal{P}$.

In general, one could define various classes of task schedules by specifying what dynamic information may be used in choosing the next task. Here, however, we opt for the oblivious version because we intend to model system dynamics separately, via high-level nondeterministic choices (cf. Section 1).

Formally, we define an operation that "applies" a task schedule to a task-PIOA:

Definition 3.3 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA where $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$. Given $\mu \in \mathsf{Disc}(\mathsf{Frags}^*(\mathcal{P}))$ and a task schedule ρ, $\mathsf{apply}(\mu, \rho)$ is the probability measure on $\mathsf{Frags}(\mathcal{P})$ defined recursively by:*

1. $\mathsf{apply}(\mu, \lambda) := \mu$. *($\lambda$ denotes the empty sequence.)*
2. *For $T \in R$, $\mathsf{apply}(\mu, T)$ is defined as follows. For every $\alpha \in \mathsf{Frags}^*(\mathcal{P})$, $\mathsf{apply}(\mu, T)(\alpha) := p_1(\alpha) + p_2(\alpha)$, where:*
   - *$p_1(\alpha) = \mu(\alpha')\eta(q)$ if α is of the form $\alpha'\, a\, q$, where $a \in T$ and $(\mathsf{lstate}(\alpha'), a, \eta) \in D$; $p_1(\alpha) = 0$ otherwise.*
   - *$p_2(\alpha) = \mu(\alpha)$ if T is not enabled in $\mathsf{lstate}(\alpha)$; $p_2(\alpha) = 0$ otherwise.*
3. *For ρ of the form $\rho'\, T$, $T \in R$, $\mathsf{apply}(\mu, \rho) := \mathsf{apply}(\mathsf{apply}(\mu, \rho'), T)$.*
4. *For ρ infinite, $\mathsf{apply}(\mu, \rho) := \lim_{i \to \infty}(\mathsf{apply}(\mu, \rho_i))$, where ρ_i denotes the length-i prefix of ρ.*

In Case (2) above, p_1 represents the probability that α is executed when applying task T at the end of α'. Because of transition-determinism and action-determinism, the transition $(\mathsf{lstate}(\alpha'), a, \eta)$ is unique, and so p_1 is well-defined. The term p_2 represents the original probability $\mu(\alpha)$, which is relevant if T is not enabled after α. It is routine to check that the limit in Case (4) is well-defined. The other two cases are straightforward.

3.3 Properties of the apply function

In this subsection, we give some basic properties of the probabilities that arise from the $\mathsf{apply}(,)$ function.

Lemma 3.4 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let μ be a discrete probability measure over finite execution fragments of $\mathcal{P}$ and let T be a task. Let p_1 and p_2 be the functions used in the definition of* $\mathsf{apply}(\mu, T)$. *Then:*

1. *for each state q, $p_1(q) = 0$;*
2. *for each finite execution fragment α,*

$$\mu(\alpha) = p_2(\alpha) + \sum_{(a,q):\alpha a q \in \mathsf{Frags}^*(\mathcal{P})} p_1(\alpha a q).$$

Proof. Item (1) follows trivially from the definition of $p_1(q)$.
For Item (2), we observe the following facts.

- If T is not enabled from $\mathsf{lstate}(\alpha)$, then, by definition of p_2, $\mu(\alpha) = p_2(\alpha)$. Furthermore, for each action a and each state q such that $\alpha a q$ is an execution fragment, we claim that $p_1(\alpha a q) = 0$. Indeed, if $a \notin T$, then the first case of the definition of $p_1(\alpha)$ trivially does not apply; if $a \in T$, then, since T is not enabled from $\mathsf{lstate}(\alpha)$, there is no ρ such that $(\mathsf{lstate}(\alpha), a, \rho) \in D_{\mathcal{P}}$, and thus, again, the first case of the definition of $p_1(\alpha)$ does not apply.
- If T is enabled from $\mathsf{lstate}(\alpha)$, then trivially $p_2(\alpha) = 0$. Furthermore, we claim that $\mu(\alpha) = \sum_{(a,q)} p_1(\alpha a q)$. By action determinism, only one action $b \in T$ is enabled from $\mathsf{lstate}(\alpha)$. By definition of p_1, $p_1(\alpha a q) = 0$ if $a \neq b$ (either $a \notin T$ or a is not enabled from $\mathsf{lstate}(\alpha)$). Thus,

$$\sum_{(a,q)} p_1(\alpha a q) = \sum_{q} p_1(\alpha b q) = \sum_{q} \mu(\alpha)\mu_{\alpha,b}(q).$$

  This in turn is equal to $\mu(\alpha)$ since $\sum_q \mu_{\alpha,b}(q) = 1$.

In each case, we get $\mu(\alpha) = p_2(\alpha) + \sum_{(a,q)} p_1(\alpha a q)$, as needed. □

Lemma 3.5 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let μ be a discrete probability measure over finite execution fragments and ρ be a finite sequence of tasks. Then* $\mathsf{apply}(\mu, \rho)$ *is a discrete probability measure over finite execution fragments.*

Proof. By a simple inductive argument on the length of ρ. The base case is trivial. For the inductive step, it suffices to show that, for each measure ϵ on finite executions fragments and each task T, $\mathsf{apply}(\epsilon, T)$ is a probability measure over finite execution fragments.

Let ϵ' be $\mathsf{apply}(\epsilon, T)$. The fact that ϵ' is a measure on finite execution fragments follows directly by Item (2) of Definition 3.3. To show that ϵ' is in fact a probability measure, we show that $\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = 1$. By Item (2) of Definition 3.3,

$$\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = \sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} (p_1(\alpha) + p_2(\alpha)).$$

Rearranging terms, we obtain

$$\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = \sum_{q} p_1(q) + \sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} (p_2(\alpha) + \sum_{(a,q):\alpha a q \in \mathsf{Frags}^*(\mathcal{P})} p_1(\alpha a q)).$$

By Lemma 3.4, the right side becomes $\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon(\alpha)$, which equals 1 since ϵ is by assumption a probability measure. Therefore $\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = 1$, as needed. □

Lemma 3.6 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA and let T be a task in R. Define $\mu' =$ $\mathsf{apply}(\mu, T)$. Then, for each finite execution fragment α:*

1. *If α consists of a single state q, then $\mu'(C_\alpha) = \mu(C_\alpha)$.*
2. *If $\alpha = \tilde{\alpha}aq$ and $a \notin T$, then $\mu'(C_\alpha) = \mu(C_\alpha)$.*
3. *If $\alpha = \tilde{\alpha}aq$ and $a \in T$, then $\mu'(C_\alpha) = \mu(C_\alpha) + \mu(\tilde{\alpha})\mu_{\tilde{\alpha},a}(q)$.*

Proof. Let p_1 and p_2 be the functions used in the definition of $\mathsf{apply}(\mu, T)$, and let α be a finite execution fragment. By definition of a cone and of μ', $\mu'(C_\alpha) = \sum_{\alpha' | \alpha \leq \alpha'} (p_1(\alpha') + p_2(\alpha'))$. By definition of a cone and Lemma 3.4,

$$\mu(C_\alpha) = \sum_{\alpha' | \alpha \leq \alpha'} (p_2(\alpha') + \sum_{(a,q): \alpha' a q \in \mathsf{Frags}^*(\mathcal{P})} p_1(\alpha' a q)) = \sum_{\alpha' | \alpha \leq \alpha'} (p_1(\alpha') + p_2(\alpha')) - p_1(\alpha).$$

Thus, $\mu'(C_\alpha) = \mu(C_\alpha) + p_1(\alpha)$. We distinguish three cases. If α consists of a single state, then $p_1(\alpha) = 0$ by Lemma 3.4, yielding $\mu'(C_\alpha) = \mu(C_\alpha)$. If $\alpha = \tilde{\alpha}aq$ and $a \notin T$, then $p_1(\alpha) = 0$ by definition, yielding $\mu'(C_\alpha) = \mu(C_\alpha)$. Finally, if $\alpha = \tilde{\alpha}aq$ and $a \in T$, then $p_1(\alpha) = \mu(\tilde{\alpha})\mu_{\tilde{\alpha},a}(q)$ by definition, yielding $\mu'(C_\alpha) = \mu(C_\alpha) + \mu(\tilde{\alpha})\mu_{\tilde{\alpha},a}(q)$. □

Lemma 3.7 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let μ be a discrete probability measure over finite execution fragments, T a task, and $\mu' = \mathsf{apply}(\mu, T)$. Then $\mu \leq \mu'$.*

Proof. Follows directly by Lemma 3.6. □

Lemma 3.8 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let μ be a discrete measure over finite execution fragments and let ρ_1 and ρ_2 be two finite sequences of tasks such that ρ_1 is a prefix of ρ_2. Then $\mathsf{apply}(\mu, \rho_1) \leq \mathsf{apply}(\mu, \rho_2)$.*

Proof. Simple inductive argument using Lemma 3.7 for the inductive step. □

The next lemma relates the probability measures on execution fragments that arise as a result when applying a sequence of tasks to a given probability measure μ on execution fragments.

Lemma 3.9 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let $\rho_1, \rho_2, \cdots$ be a finite or infinite sequence of finite task schedules, and let $\rho = \rho_1 \rho_2 \cdots$ (where juxtaposition denotes concatenation of finite sequences).*
Let μ be a discrete probability measure on finite execution fragments. For each integer i, $1 \leq i \leq |\rho|$, let $\epsilon_i = \mathsf{apply}(\mu, \rho_1 \rho_2 \cdots \rho_i)$, where $\rho_1 \cdots \rho_i$ denotes the concatenation of the sequences ρ_1 through ρ_i. Let $\epsilon = \mathsf{apply}(\mu, \rho)$. Then the ϵ_i's form a chain and $\epsilon = \lim_{i \to \infty} \epsilon_i$.

Proof. The fact that the ϵ_i's form a chain follows from Lemma 3.7. For the limit property, if the sequence $\rho_1, \rho_2, \ldots$ is finite, then the result is immediate. Otherwise, simply observe that the sequence $\epsilon_1, \epsilon_2, \ldots$ is a subsequence of the sequence used in the definition of $\mathsf{apply}(\mu, \rho_1 \rho_2 \ldots)$, and therefore, they have the same limit. □

Lemma 3.10 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let μ be a discrete probability measure over finite execution fragments of $\mathcal{P}$, ρ a task scheduler for $\mathcal{T}$, and q a state of $\mathcal{T}$. Then $\mathsf{apply}(\mu, \rho)(C_q) =$ $\mu(C_q)$.*

Proof. We prove the result for finite ρ's by induction on the length of ρ. The infinite case then follows immediately. The base case is trivial since, by definition, $\mathsf{apply}(\mu, \rho) = \mu$. For the inductive step, let $\rho = \rho' T$, and let ϵ be $\mathsf{apply}(\mu, \rho')$. By Definition 3.3, $\mathsf{apply}(\mu, \rho) = \mathsf{apply}(\epsilon, T)$. By induction, $\epsilon(C_q) = \mu(C_q)$. Therefore it suffices to show $\mathsf{apply}(\epsilon, T)(C_q) = \epsilon(C_q)$.

Let ϵ' be $\mathsf{apply}(\epsilon, T)$. By definition of cone, $\epsilon'(C_q) = \sum_{\alpha:q\leq\alpha} \epsilon'(\alpha)$. By Lemma 3.5, both ϵ and ϵ' are measures over finite execution fragments; therefore we can restrict the sum to finite execution fragments. Let p_1 and p_2 be the two functions used for the computation of $\epsilon'(\alpha)$ according to Item (2) in Definition 3.3. Then $\epsilon'(C_q) = \sum_{\alpha\in\mathsf{Execs}^*(\mathcal{P}):q\leq\alpha}(p_1(\alpha) + p_2(\alpha))$. By rearranging terms, we get $\epsilon'(C_q) = p_1(q) + \sum_{\alpha\in\mathsf{Execs}^*(\mathcal{P}):q\leq\alpha}(p_2(\alpha) + \sum_{(a,s)} p_1(C_{\alpha as}))$. By Lemma 3.4, the right side of the equation above is $\sum_{\alpha:q\leq\alpha} \epsilon(\alpha)$, which is precisely $\epsilon(C_q)$. □

The next proposition states that $\mathsf{apply}(\cdot, \rho)$ distributes over convex combinations of probability measures. This requires a preliminary lemma.

Lemma 3.11 *Let $\{\mu_i\}_i$ be a countable family of discrete probability measures on finite execution fragments and let $\{p_i\}_i$ be a countable family of probabilities such that $\sum_i p_i = 1$. Let T be a task. Then* $\mathsf{apply}(\sum_i p_i\mu_i, T) = \sum_i p_i\, \mathsf{apply}(\mu_i, T)$.

Proof. Let p_1 and p_2 be the functions used in the definition of $\mathsf{apply}(\sum_i p_i\mu_i, T)$, and let, for each i, p_1^i and p_2^i be the functions used in the definition of $\mathsf{apply}(\mu_i, T)$. Let α be a finite execution fragment. We show that $p_1(\alpha) = \sum_i p_i p_1^i(\alpha)$ and $p_2(\alpha) = \sum_i p_i p_2^i(\alpha)$. Then

$$
\begin{aligned}
\mathsf{apply}(\sum_i p_i\mu_i, T)(\alpha) &= p_1(\alpha) + p_2(\alpha) && \text{definition of } \mathsf{apply}(\sum_i p_i\mu_i, T) \\
&= \sum_i p_i p_1^i(\alpha) + \sum_i p_i p_2^i(\alpha) && \text{claims proven below} \\
&= \sum_i p_i (p_1^i(\alpha) + p_2^i(\alpha)) \\
&= \sum_i p_i\, \mathsf{apply}(\mu_i, T)(\alpha) && \text{definition of } \mathsf{apply}(\mu_i, T)
\end{aligned}
$$

To prove our claim about p_1 we distinguish two cases. If α can be written as $\alpha'\ a\ q$, where $\alpha' \in \mathsf{supp}(\mu)$, $a \in T$, and $(\mathsf{lstate}(\alpha'), a, \rho) \in D_{\mathcal{P}}$, then, by Definition 3.3, $p_1(\alpha) = (\sum_i p_i\mu_i)(\alpha')\rho(q)$, and, for each i, $p_1^i(\alpha) = \mu_i(\alpha')\rho(q)$. Thus, $p_1(\alpha) = \sum_i p_i p_1^i(\alpha)$ trivially. Otherwise, again by Definition 3.3, $p_1(\alpha) = 0$, and, for each i, $p_1^i(\alpha) = 0$. Thus, $p_1(\alpha) = \sum_i p_i p_1^i(\alpha)$ trivially.

To prove our claim about p_2 we also distinguish two cases. If T is not enabled in $\mathsf{lstate}(\alpha)$, then, by Definition 3.3, $p_2(\alpha) = (\sum_i p_i\mu_i)(\alpha)$, and, for each i, $p_2^i(\alpha) = \mu_i(\alpha)$. Thus, $p_2(\alpha) = \sum_i p_i p_2^i(\alpha)$ trivially. Otherwise, again by Definition 3.3, $p_2(\alpha) = 0$, and, for each i, $p_2^i(\alpha) = 0$. Thus, $p_2(\alpha) = \sum_i p_i p_2^i(\alpha)$ trivially. □

Proposition 3.12 *Let $\{\mu_i\}_i$ be a countable family of discrete probability measures on finite execution fragments and let $\{p_i\}_i$ be a countable family of probabilities such that $\sum_i p_i = 1$. Let ρ be a finite sequence of tasks. Then,* $\mathsf{apply}(\sum_i p_i\mu_i, \rho) = \sum_i p_i\, \mathsf{apply}(\mu_i, \rho)$.

Proof. We proceed by induction on the length of ρ. If $\rho = \lambda$, then the result is trivial since $\mathsf{apply}(\cdot, \lambda)$ is defined to be the identity function, which distributes over convex combinations of probability measures. For the inductive step, let ρ be $\rho' T$. By Definition 3.3 and the induction hypothesis,

$$\mathsf{apply}(\sum_i p_i\mu_i, \rho' T) = \mathsf{apply}(\mathsf{apply}(\sum_i p_i\mu_i, \rho'), T) = \mathsf{apply}(\sum_i p_i\, \mathsf{apply}(\mu_i, \rho'), T).$$

By Lemma 3.5, each $\mathsf{apply}(\mu_i, \rho')$ is a discrete probability measure over finite execution fragments. By Lemma 3.11, $\mathsf{apply}(\sum_i p_i\, \mathsf{apply}(\mu_i, \rho'), T) = \sum_i p_i\, \mathsf{apply}(\mathsf{apply}(\mu_i, \rho'), T)$, and by Definition 3.3, for each i, $\mathsf{apply}(\mathsf{apply}(\mu_i, \rho'), T) = \mathsf{apply}(\mu_i, \rho' T)$. Thus, $\mathsf{apply}(\sum_i p_i\mu_i, \rho' T) = \sum_i p_i\, \mathsf{apply}(\mu_i, \rho' T)$ as needed. □

3.4 Task schedules vs. standard PIOA schedulers

Here, we show that $\mathsf{apply}(\mu, \rho)$ is a generalized probabilistic execution fragment generated by μ and a scheduler for $\mathcal{P}$, in the usual sense. Thus, a task schedule for a task-PIOA is a special case of a scheduler for the underlying PIOA.

Theorem 3.13 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. For each probability measure μ on $\mathsf{Frags}^*(\mathcal{P})$ and task schedule ρ, there is scheduler σ for $\mathcal{P}$ such that $\mathsf{apply}(\mu, \rho)$ is the generalized probabilistic execution fragment $\epsilon_{\sigma,\mu}$.*

The proof of Theorem 3.13 uses several auxiliary lemmas. The first talks about applying λ, the empty sequence of tasks. It is used in the base case of the inductive proof for Lemma 3.16, which involves applying any finite sequence of tasks.

Lemma 3.14 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let μ be a discrete probability measure over finite execution fragments. Then $\mathsf{apply}(\mu, \lambda)$ is a generalized probabilistic execution fragment generated by μ.*

Proof. Follows directly from the definitions, by defining a scheduler σ such that $\sigma(\alpha)(\mathsf{tran}) = 0$ for each finite execution fragment α and each transition tran. □

The next lemma provides the inductive step needed for Lemma 3.16.

Lemma 3.15 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. If ϵ is a generalized probabilistic execution fragment generated by a measure μ, then, for each task T, $\mathsf{apply}(\epsilon, T)$ is a generalized probabilistic execution fragment generated by μ.*

Proof. Suppose ϵ is generated by μ together with a scheduler σ (that is, $\epsilon_{\sigma,\mu} = \epsilon$). Let ϵ' be $\mathsf{apply}(\epsilon, T)$. For each finite execution fragment α, let $D(\mathsf{lstate}(\alpha))$ denote the set of transitions of D with source state $\mathsf{lstate}(\alpha)$. For each $\mathsf{tran} \in D$, let $\mathsf{act}(\mathsf{tran})$ denote the action that occurs in tran. Now we define a new scheduler σ' as follows: given finite execution fragment α and $\mathsf{tran} \in D$,

- if $\epsilon'(C_\alpha) - \mu(C_\alpha - \{\alpha\}) = 0$, then $\sigma'(\alpha)(\mathsf{tran}) = 0$;
- otherwise, if $\mathsf{tran} \in D(\mathsf{lstate}(\alpha))$ and $\mathsf{act}(\mathsf{tran}) \in T$, then

$$\sigma'(\alpha)(\mathsf{tran}) = \frac{\epsilon(C_\alpha) - \mu(C_\alpha - \{\alpha\})}{\epsilon'(C_\alpha) - \mu(C_\alpha - \{\alpha\})}(\sigma(\alpha)(\mathsf{tran}) + \sigma(\alpha)(\perp));$$

- otherwise,

$$\sigma'(\alpha)(\mathsf{tran}) = \frac{\epsilon(C_\alpha) - \mu(C_\alpha - \{\alpha\})}{\epsilon'(C_\alpha) - \mu(C_\alpha - \{\alpha\})}\sigma(\alpha)(\mathsf{tran}).$$

We first argue that σ', thus defined, is a scheduler. Let a finite execution fragment α be given. If the first clause applies, then $\sigma'(\alpha)$ is 0 everywhere, hence is a sub-probability measure. Assume otherwise. By action- and transition-determinism, there is at most one tran with $\mathsf{tran} \in D(\mathsf{lstate}(\alpha))$ and $\mathsf{act}(\mathsf{tran}) \in T$. Let Y denote $\{\mathsf{tran}\}$ if such tran exists and $\emptyset$ otherwise. Then we have the following.

$$\begin{aligned}
&\sum_{\mathsf{tran} \notin Y} \sigma(\alpha)(\mathsf{tran}) + \sum_{\mathsf{tran} \in Y} (\sigma(\alpha)(\mathsf{tran}) + \sigma(\alpha)(\perp)) \\
&= (\sum_{\mathsf{tran} \in D} \sigma(\alpha)(\mathsf{tran})) + \sigma(\alpha)(\perp) && Y \text{ is either empty or a singleton} \\
&= 1 && \sigma \text{ is a scheduler}
\end{aligned}$$

Furthermore, by Lemma 3.7, we know that $\epsilon(C_\alpha) \leq \epsilon'(C_\alpha)$, thus the fraction $\frac{\epsilon(C_\alpha)-\mu(C_\alpha-\{\alpha\})}{\epsilon'(C_\alpha)-\mu(C_\alpha-\{\alpha\})}$ is at most 1. Putting the pieces together, we have

$$\sum_{\mathsf{tran}\in D} \sigma'(\alpha)(\mathsf{tran}) = \frac{\epsilon(C_\alpha) - \mu(C_\alpha - \{\alpha\})}{\epsilon'(C_\alpha) - \mu(C_\alpha - \{\alpha\})} \cdot \Big(\sum_{\mathsf{tran}\notin Y} \sigma(\alpha)(\mathsf{tran}) + \sum_{\mathsf{tran}\in Y} (\sigma(\alpha)(\mathsf{tran}) + \sigma(\alpha)(\bot))\Big) \leq 1.$$

Next, we prove by induction on the length of a finite execution fragment α that $\epsilon_{\sigma',\mu}(C_\alpha) = \epsilon'(C_\alpha)$.

For the base case, let $\alpha = q$. By Lemma 2.10,

$$\epsilon_{\sigma',\mu}(C_q) = \mu(C_q) = \epsilon_{\sigma,\mu}(C_q).$$

By the choice of σ, the last expression equals $\epsilon(C_q)$, which in turn is equal to to $\epsilon'(C_q)$ by virtue of Lemma 3.10. Thus, $\epsilon_{\sigma',\mu}(C_q) = \epsilon'(C_q)$, as needed.

For the inductive step, let $\alpha = \tilde{\alpha} a q$. By Lemma 2.10 and the definition of the measure of a cone, we get

$$\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha) + \sum_{\alpha'<\alpha} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_\alpha) = \mu(C_\alpha) + \sum_{\alpha'\leq\tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})\mu_{\sigma'(\tilde{\alpha})}(a, q).$$

We know that a is enabled from $\mathsf{lstate}(\tilde{\alpha})$, because α is an execution fragment of $\mathcal{P}$. Thus, $\mathsf{tran}_{\tilde{\alpha},a}$ and $\mu_{\tilde{\alpha},a}$ are defined. By expanding $\mu_{\sigma'(\tilde{\alpha})}(a, q)$ in the equation above, we get

$$\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha) + \sum_{\alpha'\leq\tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})\sigma'(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q). \tag{1}$$

We distinguish three cases.

1. $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$.

   By inductive hypothesis, $\epsilon_{\sigma',\mu}(C_{\tilde{\alpha}}) = \epsilon'(C_{\tilde{\alpha}})$. Then by Lemma 2.12, $\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha)$. It is therefore sufficient to show that $\epsilon'(C_\alpha) = \mu(C_\alpha)$.

   By Lemma 3.7, $\epsilon(C_{\tilde{\alpha}}) \leq \epsilon'(C_{\tilde{\alpha}})$. Thus, using $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$, we get $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) \leq 0$. On the other hand, from Lemma 2.11 and the fact that $\epsilon = \epsilon_{\sigma,\mu}$, we have $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) \geq 0$. Thus, $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$. Now, using Lemma 2.12 and the fact that $\epsilon_{\sigma,\mu} = \epsilon$ and $\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) = 0$, we get $\epsilon(C_\alpha) = \mu(C_\alpha)$.

   Since $C_{\tilde{\alpha}} - \{\tilde{\alpha}\}$ is a union of cones, we may use Lemma 3.7 to obtain $\mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) \leq \epsilon(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$. Adding $\epsilon(\{\tilde{\alpha}\})$ on both sides, we get $\mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) + \epsilon(\{\tilde{\alpha}\}) \leq \epsilon(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) + \epsilon(\{\tilde{\alpha}\}) = \epsilon(C_{\tilde{\alpha}})$. Since $\epsilon(C_{\tilde{\alpha}}) = \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$, the previous inequalities imply $\epsilon(C_{\tilde{\alpha}}) + \epsilon(\{\tilde{\alpha}\}) \leq \epsilon(C_{\tilde{\alpha}})$, therefore $\epsilon(\{\tilde{\alpha}\}) = 0$. By Lemma 3.6 (Items (2) and (3)), we have $\epsilon'(C_\alpha) = \epsilon(C_\alpha) = \mu(C_\alpha)$, as needed.

2. $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) > 0$ and $a \notin T$.

   By Equation (1) and the definition of σ', we know that $\epsilon_{\sigma',\mu}(C_\alpha)$ equals

   $$\mu(C_\alpha) + \sum_{\alpha'\leq\tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})\frac{\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})}{\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})}\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q).$$

   Observe that in the sum above only the factors $\mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})$ are not constant with respect to the choice of α'. By Lemma 2.11, $\sum_{\alpha'\leq\tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}}) = \epsilon_{\sigma',\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$. By the inductive hypothesis, $\epsilon_{\sigma',\mu}(C_{\tilde{\alpha}}) = \epsilon'(C_{\tilde{\alpha}})$. Thus, replacing $\sum_{\alpha'\leq\tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})$ with $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$ and simplifying the resulting expression, we obtain

   $$\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha) + (\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\,\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q).$$

   By definition, $\epsilon = \epsilon_{\sigma,\mu}$. Therefore, by Lemma 2.12, the right side of the equation above is $\epsilon(C_\alpha)$. Moreover, $\epsilon(C_\alpha) = \epsilon'(C_\alpha)$ by Lemma 3.6, Item (2). Thus, $\epsilon_{\sigma',\mu}(C_\alpha) = \epsilon'(C_\alpha)$, as needed.

3. $\epsilon'(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}) > 0$ and $a \in T$.

   As in the previous case, $\epsilon_{\sigma',\mu}(C_\alpha)$ equals

$$\mu(C_\alpha) + (\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))(\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a}) + \sigma(\tilde{\alpha})(\perp))\mu_{\tilde{\alpha},a}(q).$$

   Also shown in the previous case, we have

$$\epsilon(C_\alpha) = \mu(C_\alpha) + (\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\,\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q).$$

   Therefore,

$$\epsilon_{\sigma',\mu}(C_\alpha) = \epsilon(C_\alpha) + (\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\sigma(\tilde{\alpha})(\perp)\mu_{\tilde{\alpha},a}(q).$$

   By definition, $\epsilon = \epsilon_{\sigma,\mu}$. Applying Lemma 2.13, we substitute $\epsilon(\tilde{\alpha})$ for $(\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\sigma(\tilde{\alpha})(\perp)$. Now we have

$$\epsilon_{\sigma',\mu}(C_\alpha) = \epsilon(C_\alpha) + \epsilon(\tilde{\alpha})\mu_{\tilde{\alpha},a}(q).$$

   The desired result now follows from Lemma 3.6, Item (3).

□

Now we can show that applying any finite sequences of tasks to a probability measure on finite execution fragments leads to a generalized probabilistic execution fragment.

Lemma 3.16 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. For each probability measure μ on finite execution fragments and each finite sequence of tasks ρ, $\mathsf{apply}(\mu, \rho)$ is a generalized probabilistic execution fragment generated by μ.*

Proof. Simple inductive argument using Lemma 3.14 for the base case and Lemma 3.15 for the inductive step. □

And now we consider infinite sequences of tasks.

Lemma 3.17 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. For each measure μ on finite execution fragments and each infinite sequence of tasks ρ, $\mathsf{apply}(\mu, \rho)$ is a generalized probabilistic execution fragment generated by μ.*

Proof. For each $i \geq 0$, let ρ_i denote the length-i prefix of ρ and let ϵ_i be $\mathsf{apply}(\mu, \rho_i)$. By Lemmas 3.16 and 3.8, the sequence $\epsilon_0, \epsilon_1, \ldots$ is a chain of generalized probabilistic execution fragments generated by μ. By Proposition 2.15, $\lim_{i\to\infty} \epsilon_i$ is a generalized probabilistic execution fragment generated by μ. This suffices, since $\mathsf{apply}(\mu, \rho)$ is $\lim_{i\to\infty} \epsilon_i$ by definition. □

This completes the proof of Theorem 3.13.

Proof (Theorem 3.13). Follows directly from Lemmas 3.16 and 3.17. □

The idea here is, for any measure μ and task sequence ρ, the probability measure on execution fragments generated by $\mathsf{apply}(\mu, \rho)$ is "standard", in the sense that it can be obtained from μ and a scheduler as defined in Section 3 for basic PIOAs. Any such $\mathsf{apply}(\mu, \rho)$ is said to be a *generalized probabilistic execution fragment* of the task-PIOA $\mathcal{T}$. *Probabilistic execution fragments* and *probabilistic executions* are then defined by making the same restrictions as for basic PIOAs. We write $\mathsf{tdist}(\mu, \rho)$ as shorthand for $\mathsf{tdist}(\mathsf{apply}(\mu, \rho))$, the trace distribution obtained by applying task schedule ρ starting from the measure μ on execution fragments. We write $\mathsf{tdist}(\rho)$ for $\mathsf{tdist}(\mathsf{apply}(\delta(\bar{q}), \rho))$ the trace distribution obtained by applying ρ from the unique start state. (Recall that $\delta(\bar{q})$ denotes the Dirac measure on $\bar{q}$.) A *trace distribution* of $\mathcal{T}$ is any $\mathsf{tdist}(\rho)$. We use $\mathsf{tdists}(\mathcal{T})$ to denote the set $\{\mathsf{tdist}(\rho) : \rho$ is a task schedule for $\mathcal{T}\}$.

3.5 Composition

We define composition of task-PIOAs:

Definition 3.18 *Two task-PIOAs $\mathcal{T}_i = (\mathcal{P}_i, R_i)$, $i \in \{1, 2\}$, are said to be* compatible *provided the underlying PIOAs are compatible. Then we define their* composition *$\mathcal{T}_1 \| \mathcal{T}_2$ to be the task-PIOA $(\mathcal{P}_1 \| \mathcal{P}_2, R_1 \cup R_2)$.*

It is easy to see that $\mathcal{T}_1 \| \mathcal{T}_2$ is in fact a task-PIOA. In particular, since compatibility ensures disjoint sets of locally-controlled actions, $R_1 \cup R_2$ is an equivalence relation on the locally-controlled actions of $\mathcal{P}_1 \| \mathcal{P}_2$. It is also easy to see that action determinism is preserved under composition. Note that, when two task-PIOAs are composed, no new mechanisms are required to schedule actions of the two components—the tasks alone are enough.

3.6 Hiding

We also define a hiding operator for task-PIOAs. It simply hides output actions:

Definition 3.19 *Let $\mathcal{T} = (\mathcal{P}, R)$ be any task-PIOA, where $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$, and let $S \subseteq O$. Then $\mathsf{hide}(\mathcal{T}, S)$ is the task-PIOA $(\mathsf{hide}(\mathcal{P}, S), R)$, that is, the task-PIOA obtained by hiding S in the underlying PIOA $\mathcal{P}$, without any change to the task equivalence relation.*

Note that, in the special case where tasks respect the output vs. internal action classification, one can also define a hiding operation that hides all output actions in a set of tasks. We omit the details here.

3.7 Implementation

We now define the notion of external behavior for a task-PIOA and the induced implementation relation between task-PIOAs. Unlike previous definitions of external behavior, the one we use here is not simply a set of trace distributions. Rather, it is a mapping that specifies, for every possible "environment" $\mathcal{E}$ for the given task-PIOA $\mathcal{T}$, the set of trace distributions that can arise when $\mathcal{T}$ is composed with $\mathcal{E}$.

Definition 3.20 *Let $\mathcal{T}$ be any task-PIOA and $\mathcal{E}$ be an action-deterministic task-PIOA. We say that $\mathcal{E}$ is an* environment *for $\mathcal{T}$ if the following hold:*

1. *$\mathcal{E}$ is compatible with $\mathcal{T}$.*
2. *The composition $\mathcal{T} \| \mathcal{E}$ is closed.*

Note that $\mathcal{E}$ is allowed to have output actions that are not inputs of $\mathcal{T}$.

Definition 3.21 *The* external behavior *of $\mathcal{T}$, denoted by $\mathsf{extbeh}(\mathcal{T})$, is the total function that maps each environment $\mathcal{E}$ to the set of trace distributions $\mathsf{tdists}(\mathcal{T} \| \mathcal{E})$.*

Thus, for each environment, we consider the set of trace distributions that arise from all task schedules. Note that these traces may include new output actions of $\mathcal{E}$, in addition to the external actions already present in $\mathcal{T}$.

Our definition of *implementation* says that the lower-level system must "look like" the higher-level system from the perspective of every possible environment. The style of this definition is influenced by common notions in the security protocol literature (e.g., [LMMS98, Can01, PW01]). An advantage of this style of definition is that it yields simple compositionality results (Theorem 3.24). In our case, "looks like" is formalized in terms of inclusion of sets of trace distributions, that is, of external behavior sets.

Definition 3.22 *Let $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ be task-PIOAs, and I_i and O_i the input and output actions sets for $\mathcal{P}_i$, $i \in \{1, 2\}$. Then $\mathcal{T}_1$ and $\mathcal{T}_2$ are* comparable *if $I_1 = I_2$ and $O_1 = O_2$.*

Definition 3.23 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be* comparable *action-deterministic task-PIOAs. Then we say that $\mathcal{T}_1$ implements $\mathcal{T}_2$, written $\mathcal{T}_1 \leq_0 \mathcal{T}_2$, if $\mathsf{extbeh}(\mathcal{T}_1)(\mathcal{E}) \subseteq \mathsf{extbeh}(\mathcal{T}_2)(\mathcal{E})$ for every environment $\mathcal{E}$ for both $\mathcal{T}_1$ and $\mathcal{T}_2$. In other words, we require $\mathsf{tdists}(\mathcal{T}_1||\mathcal{E}) \subseteq \mathsf{tdists}(\mathcal{T}_2||\mathcal{E})$ for every $\mathcal{E}$.*

The subscript 0 in the relation symbol $\leq_0$ refers to the requirement that every trace distribution in $\mathsf{tdists}(\mathcal{T}_1||\mathcal{E})$ must have an identical match in $\mathsf{tdists}(\mathcal{T}_2||\mathcal{E})$. For security analysis, we also define another relation $\leq_{neg,pt}$, which allows "negligible" discrepancies between matching trace distributions [CCK+06e].

3.8 Compositionality

Because external behavior and implementation are defined in terms of mappings from environments to sets of trace distributions, a compositionality result for $\leq_0$ follows easily:

Theorem 3.24 *Let $\mathcal{T}_1$, $\mathcal{T}_2$ be comparable action-deterministic task-PIOAs such that $\mathcal{T}_1 \leq_0 \mathcal{T}_2$, and let $\mathcal{T}_3$ be an action-deterministic task-PIOA compatible with each of $\mathcal{T}_1$ and $\mathcal{T}_2$. Then $\mathcal{T}_1\|\mathcal{T}_3 \leq_0 \mathcal{T}_2\|\mathcal{T}_3$.*

Proof. Let $\mathcal{T}_4 = (\mathcal{P}_4, R_4)$ be any environment (action-deterministic) task-PIOA for both $\mathcal{T}_1\|\mathcal{T}_3$ and $\mathcal{T}_2\|\mathcal{T}_3$. Fix any task schedule ρ_1 for $(\mathcal{T}_1\|\mathcal{T}_3)\|\mathcal{T}_4$. Let τ be the trace distribution of $(\mathcal{T}_1\|\mathcal{T}_3)\|\mathcal{T}_4$ generated by ρ_1. It suffices to show that τ is also generated by some task schedule ρ_2 for $(\mathcal{T}_2\|\mathcal{T}_3)\|\mathcal{T}_4$.

Note that ρ_1 is also a task schedule for $\mathcal{T}_1\|(\mathcal{T}_3\|\mathcal{T}_4)$, and that ρ_1 generates the same trace distribution τ in the composed task-PIOA $\mathcal{T}_1\|(\mathcal{T}_3\|\mathcal{T}_4)$.

Now, $\mathcal{T}_3\|\mathcal{T}_4$ is an (action-deterministic) environment task-PIOA for each of $\mathcal{T}_1$ and $\mathcal{T}_2$. Since, by assumption, $\mathcal{T}_1 \leq_0 \mathcal{T}_2$, we infer the existence of a task schedule ρ_2 for $\mathcal{T}_2\|(\mathcal{T}_3\|\mathcal{T}_4)$ such that ρ_2 generates trace distribution τ in the task-PIOA $\mathcal{T}_2\|(\mathcal{T}_3\|\mathcal{T}_4)$. Since ρ_2 is also a task schedule for $(\mathcal{T}_2\|\mathcal{T}_3)\|\mathcal{T}_4$ and ρ_2 generates τ, this suffices. □

4 Simulation Relations

Now we define a new notion of simulation relations for closed, action-deterministic task-PIOAs, and show that it is sound for proving $\leq_0$. Our definition is based on the three operations defined in Section 2.2: flattening, lifting, and expansion.

4.1 Simulation relation definition

We begin with two auxiliary definitions. The first expresses consistency between a probability measure over finite executions and a task schedule. Informally, a measure ϵ over finite executions is said to be consistent with a task schedule ρ if it assigns non-zero probability only to those executions that are possible under the task schedule ρ. We use this condition to avoid extraneous proof obligations in our definition of simulation relation.

Definition 4.1 *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed, action-deterministic task-PIOA and let ϵ be a discrete probability measure over finite executions of $\mathcal{P}$. Also, let a finite task schedule ρ for $\mathcal{T}$ be given. Then ϵ is* consistent with *ρ provided that $\mathsf{supp}(\epsilon) \subseteq \mathsf{supp}(\mathsf{apply}(\delta(\bar{q}), \rho))$, where $\bar{q}$ is the start state of $\mathcal{P}$.*

For the second definition, suppose we have two task-PIOAs $\mathcal{T}_1$ and $\mathcal{T}_2$, and a mapping c that takes a finite task schedule ρ and a task T of $\mathcal{T}_1$ to a task schedule of $\mathcal{T}_2$. The idea is that $\mathsf{c}(\rho, T)$ describes how $\mathcal{T}_2$ matches task T, given that it has already matched the task schedule ρ. Using c, we define a new function $\mathsf{full}(\mathsf{c})$ that, given a task schedule ρ, iterates c on all the elements of ρ, thus producing a "full" task schsedule of $\mathcal{T}_2$ that matches all of ρ.

Definition 4.2 *Let $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ be two task-PIOAs, and let $\mathsf{c} : ({R_1}^* \times R_1) \to {R_2}^*$ be given. Define $\mathsf{full}(\mathsf{c}) : {R_1}^* \to {R_2}^*$ recursively as follows: $\mathsf{full}(\mathsf{c})(\lambda) := \lambda$, and $\mathsf{full}(\mathsf{c})(\rho T) := \mathsf{full}(\mathsf{c})(\rho) \frown \mathsf{c}(\rho, T)$ (that is, the concatenation of $\mathsf{full}(\mathsf{c})(\rho)$ and $\mathsf{c}(\rho, T)$).*

Next, we define our new notion of simulation for task-PIOAs. Note that our simulation relations are relations between probability measures on executions, as opposed to relations between states. Here the use of measures on executions is motivated by certain cases that arise in our OT protocol proof. For example, we wish to match random choices that are made at different points in the low-level and high-level models (see Section 4.3).

Definition 4.3 *Let $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ be two comparable task-PIOAs that are closed and action-deterministic. Let R be a relation from $\mathsf{Disc}(\mathsf{Execs}^*(\mathcal{P}_1))$ to $\mathsf{Disc}(\mathsf{Execs}^*(\mathcal{P}_2))$, such that, if $\epsilon_1 \; R \; \epsilon_2$, then $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$. (That is, the two measures on finite executions yield the same measure on traces.) Then R is a* simulation *from $\mathcal{T}_1$ to $\mathcal{T}_2$ if there exists $\mathsf{c} : ({R_1}^* \times R_1) \to {R_2}^*$ such that the following properties hold:*

1. Start condition: $\delta(\bar{q}_1) \; R \; \delta(\bar{q}_2)$.
2. Step condition: *If $\epsilon_1 \; R \; \epsilon_2$, $\rho_1 \in {R_1}^*$, ϵ_1 is consistent with ρ_1, ϵ_2 is consistent with $\mathsf{full}(\mathsf{c})(\rho_1)$, and $T \in R_1$, then $\epsilon_1' \; \mathcal{E}(R) \; \epsilon_2'$ where $\epsilon_1' = \mathsf{apply}(\epsilon_1, T)$ and $\epsilon_2' = \mathsf{apply}(\epsilon_2, \mathsf{c}(\rho_1, T))$.*

Intuitively, $\epsilon_1 \; R \; \epsilon_2$ means that it is possible to simulate from ϵ_2 anything that can happen from ϵ_1. Furthermore, $\epsilon_1' \; \mathcal{E}(R) \; \epsilon_2'$ means that we can decompose ϵ_1' and ϵ_2' into pieces that can simulate each other, and so we can also say that it is possible to simulate from ϵ_2' anything that can happen from ϵ_1'. This rough intuition is at the base of the proof of our soundness result, Theorem 4.6.

The next three subsections establish the soundness of our simulation relations with respect to the $\leq_0$ relation.

4.2 Soundness

In this section, we state and prove two soundness results. The first result, Theorem 4.6, says that, for closed task-PIOAs, the existence of a simulation relation implies inclusion of sets of trace distributions.

The proof requires two lemmas. Recall that the definition of simulation relations requires that any two R-related execution distributions must have the same trace distribution. Lemma 4.4 extends this property to the claim that any pair of execution distributions that are related by the expansion of the relation R, $\mathcal{E}(R)$, must also have the same trace distribution. (For the proof, the only property of simulation relations we need is that related execution distributions have the same trace distribution.)

Lemma 4.4 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be comparable closed action-deterministic task-PIOAs and let R be a simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Let ϵ_1 and ϵ_2 be discrete probability measures over finite executions of $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, such that $\epsilon_1 \; \mathcal{E}(R) \; \epsilon_2$. Then $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$.*

Proof. Since $\epsilon_1 \; \mathcal{E}(R) \; \epsilon_2$, we may choose measures η_1, η_2 and a weighting functions w as in the definition of expansion. Then for all $\rho_1 \in \mathsf{supp}(\eta_1)$, we have $\eta_1(\rho_1) = \sum_{\rho_2 \in \mathsf{supp}(\eta_2)} w(\rho_1, \rho_2)$. Moreover, we have $\epsilon_1 = \mathsf{flatten}(\eta_1)$, therefore

$$\mathsf{tdist}(\epsilon_1) = \sum_{\rho_1 \in \mathsf{supp}(\eta_1)} \eta_1(\rho_1) \, \mathsf{tdist}(\rho_1) = \sum_{\rho_1 \in \mathsf{supp}(\eta_1)} \sum_{\rho_2 \in \mathsf{supp}(\eta_2)} w(\rho_1, \rho_2) \, \mathsf{tdist}(\rho_1).$$

Now consider any ρ_1 and ρ_2 with $w(\rho_1, \rho_2) > 0$. By the definition of a weighting function, we may conclude that $\rho_1 \; R \; \rho_2$. Since R is a simulation relation, we have $\mathsf{tdist}(\rho_1) = \mathsf{tdist}(\rho_2)$. Thus we may replace $\mathsf{tdist}(\rho_1)$ by $\mathsf{tdist}(\rho_2)$ in the summation above. This yields:

$$\mathsf{tdist}(\epsilon_1) = \sum_{\rho_1 \in \mathsf{supp}(\eta_1)} \sum_{\rho_2 \in \mathsf{supp}(\eta_2)} w(\rho_1, \rho_2) \, \mathsf{tdist}(\rho_2) = \sum_{\rho_2 \in \mathsf{supp}(\eta_2)} \sum_{\rho_1 \in \mathsf{supp}(\eta_1)} w(\rho_1, \rho_2) \, \mathsf{tdist}(\rho_2).$$

Using again the fact that w is a weighting function, we can simplify the inner sum above to obtain

$$\mathsf{tdist}(\epsilon_1) = \sum_{\rho_2 \in \mathsf{supp}(\eta_2)} \eta_2(\rho_2)\,\mathsf{tdist}(\rho_2).$$

This equals $\mathsf{tdist}(\epsilon_2)$ because, by the choice of η_2, we know that $\epsilon_2 = \mathsf{flatten}(\eta_2)$. □

The second lemma provides the inductive step needed in the proof of Theorem 4.6.

Lemma 4.5 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable closed task-PIOAs and let R be a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Furthermore, let c be a mapping witnessing the fact that R is a simulation relation. Let a finite task scheduler ρ_1 of $\mathcal{T}_1$ be given and set $\rho_2 = \mathsf{full}(\mathsf{c})(\rho_1)$. (Then ρ_2 is a finite task scheduler of $\mathcal{T}_2$.) Let ϵ_1 denote $\mathsf{apply}(\delta(\bar{q}_1), \rho_1)$ and let ϵ_2 denote $\mathsf{apply}(\delta(\bar{q}_2), \rho_2)$. Suppose that $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$.*

*Now let T be a task of $\mathcal{T}_1$. Let $\epsilon'_1 = \mathsf{apply}(\delta(\bar{q}_1), \rho_1 T)$ and let $\epsilon'_2 = \mathsf{apply}(\delta(\bar{q}_2), \rho_2\, \mathsf{c}(\rho_1, T))$.
Then $\epsilon'_1\ \mathcal{E}(R)\ \epsilon'_2$.*

Proof. Let η_1, η_2 and w be the measures and weighting function that witness $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$. Observe that $\epsilon'_1 = \mathsf{apply}(\epsilon_1, T)$ and $\epsilon'_2 = \mathsf{apply}(\epsilon_2, \mathsf{c}(\rho_1, T))$.

We apply Lemma 2.7: define the function f on discrete distributions on finite executions of $\mathcal{T}_1$ by $f(\epsilon) = \mathsf{apply}(\epsilon, T)$, and the function g on discrete distributions on finite executions of $\mathcal{T}_2$ by $g(\epsilon) = \mathsf{apply}(\epsilon, \mathsf{c}(\rho_1, T))$. We show that the hypothesis of Lemma 2.7 is satisfied, so we can invoke Lemma 2.7 to conclude that $\epsilon'_1\ \mathcal{E}(R)\ \epsilon'_2$.

Distributivity of f and g follows directly by Proposition 3.12. Let μ_1, μ_2 be two measures such that $w(\mu_1, \mu_2) > 0$. We must show that $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$. Since w is a weighting function for $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$, $\mu_1\ R\ \mu_2$. Observe that $\mathsf{supp}(\mu_1) \subseteq \mathsf{supp}(\epsilon_1)$ and $\mathsf{supp}(\mu_2) \subseteq \mathsf{supp}(\epsilon_2)$; thus, μ_1 is consistent with ρ_1 and μ_2 is consistent with ρ_2. By the step condition for R, $\mathsf{apply}(\mu_1, T)\ \mathcal{E}(R)\ \mathsf{apply}(\mu_2, \mathsf{c}(\rho_1, T))$. Observe that $\mathsf{apply}(\mu_1, T) = f(\mu_1)$ and that $\mathsf{apply}(\mu_2, \mathsf{c}(\rho_1, T)) = g(\mu_2)$. Thus, $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$, as needed. □

The following theorem, Theorem 4.6, is the main soundness result. The proof simply puts the pieces together, using Lemma 3.9 (which says that the probabilistic execution generated by an infinite task scheduler can be seen as the limit of the probabilistic executions generated by some of the finite prefixes of the task scheduler), Lemma 4.5 (the step condition), Lemma 4.4 (related probabilistic executions have the same trace distribution), and Lemma A.9 (limit commutes with $tdist$).

Theorem 4.6 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be comparable task-PIOAs that are closed and action-deterministic. If there exists a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$, then $\mathsf{tdists}(\mathcal{T}_1) \subseteq \mathsf{tdists}(\mathcal{T}_2)$.*

Proof (Theorem 4.6). Let R be the assumed simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Let ϵ_1 be the probabilistic execution of $\mathcal{T}_1$ generated by $\bar{q}_1$ and a (finite or infinite) task schedule, $T_1 T_2 \cdots$. For each $i > 0$, define ρ_i to be $\mathsf{c}(T_1 \cdots T_{i-1}, T_i)$. Let ϵ_2 be the probabilistic execution generated by $\bar{q}_2$ and the concatenation $\rho_1 \rho_2 \cdots$. It is sufficient to prove $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$.

For each $j \geq 0$, let $\epsilon_{1,j} = \mathsf{apply}(\bar{q}_1, T_1 \cdots T_j)$, and $\epsilon_{2,j} = \mathsf{apply}(\bar{q}_2, \rho_1 \cdots \rho_j)$. Then by Lemma 3.9, for each $j \geq 0$, $\epsilon_{1,j} \leq \epsilon_{1,j+1}$ and $\epsilon_{2,j} \leq \epsilon_{2,j+1}$; moreover, $\lim_{j\to\infty} \epsilon_{1,j} = \epsilon_1$ and $\lim_{j\to\infty} \epsilon_{2,j} = \epsilon_2$. Also, for every $j \geq 0$, $\mathsf{apply}(\epsilon_{1,j}, T_{j+1}) = \epsilon_{1,j+1}$ and $\mathsf{apply}(\epsilon_{2,j}, \rho_{j+1}) = \epsilon_{2,j+1}$.

Observe that $\epsilon_{1,0} = \delta(\bar{q}_1)$ and $\epsilon_{2,0} = \delta(\bar{q}_2)$. The start condition for a simulation relation and a trivial expansion imply that $\epsilon_{1,0}\ \mathcal{E}(R)\ \epsilon_{2,0}$. Then by induction, using Lemma 4.5 for the definition of a simulation relation in proving the inductive step, for each $j \geq 0$, $\epsilon_{1,j}\ \mathcal{E}(R)\ \epsilon_{2,j}$. Then, by Lemma 4.4, for each $j \geq 0$, $\mathsf{tdist}(\epsilon_{1,j}) = \mathsf{tdist}(\epsilon_{2,j})$.

By Lemma A.9, $\mathsf{tdist}(\epsilon_1) = \lim_{j\to\infty} \mathsf{tdist}(\epsilon_{1,j})$, and $\mathsf{tdist}(\epsilon_2) = \lim_{j\to\infty} \mathsf{tdist}(\epsilon_{2,j})$. Since for each $j \geq 0$, $\mathsf{tdist}(\epsilon_{1,j}) = \mathsf{tdist}(\epsilon_{2,j})$, we conclude that $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$, as needed. □

The second soundness result, Corollary 4.7, asserts soundness for (not necessarily closed) task-PIOAs, with respect to the $\leq_0$ relation.

Corollary 4.7 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable action-deterministic task-PIOAs. Suppose that, for every environment $\mathcal{E}$ for both $\mathcal{T}_1$ and $\mathcal{T}_2$, there exists a simulation relation R from $\mathcal{T}_1 \| \mathcal{E}$ to $\mathcal{T}_2 \| \mathcal{E}$. Then $\mathcal{T}_1 \leq_0 \mathcal{T}_2$.*

Proof. Immediate by Theorem 4.6 and the definition of $\leq_0$. □

4.3 Example: $Trapdoor$ vs. $Rand$

The following example, taken from our Oblivious Transfer case study, is a key motivation for generalizing prior notions of simulation relations. We consider two closed task-PIOAs, $Trapdoor$ and $Rand$. $Rand$ simply chooses a number in $\{1, \ldots, n\}$ randomly, from the uniform distribution (using a $choose$ internal action), and then outputs the chosen value k (using a $report(k)$ output action). $Trapdoor$, on the other hand, first chooses a random number, then applies a known permutation f to the chosen number, and then outputs the result. (The name $Trapdoor$ refers to the type of permutation f that is used in the OT protocol.)

More precisely, neither $Rand$ nor $Trapdoor$ has any input actions. $Rand$ has output actions $report(k)$, $k \in [n] = \{1, \ldots, n\}$ and internal action $choose$. It has tasks $Report = \{report(k) : k \in [n]\}$, and $Choose = \{choose\}$. Its state contains one variable $zval$, which assumes values in $[n] \cup \{\perp\}$, initially $\perp$. The $choose$ action is enabled when $zval = \perp$, and has the effect of setting $zval$ to a number in $[n]$, chosen uniformly at random. The $report(k)$ action is enabled when $zval = k$, and has no effect on the state (so it may happen repeatedly). Precondition/effect code for $Rand$ appears in Figure 1, and a diagram appears in Figure 2.

Rand:
Signature:

Input:
  none
Output:
  $report(k)$, $k \in \{1, \ldots, n\}$
Internal:
  $choose$

Tasks:
$Report = \{report(k) : k \in \{1, \ldots, n\}\}$, $Choose = \{choose\}$
States:
$zval \in \{1, \ldots, n\} \cup \{\perp\}$, initially $\perp$

Transitions:

$choose$
Precondition:
  $zval = \perp$
Effect:
  $zval :=$ random(uniform($\{1, \ldots, n\}$))

$report(k)$
Precondition:
  $zval = k$
Effect:
  none

Figure 1: Code for Task-PIOA $Rand$

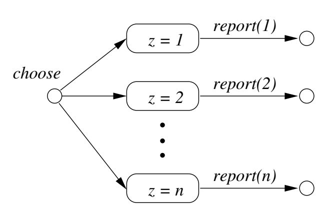

Figure 2: Task-PIOA $Rand$

$Trapdoor$ has the same actions as $Rand$, plus internal action $compute$. It has the same tasks as $Rand$, plus the task $Compute = \{compute\}$. $Trapdoor$'s state contains two variables, y and z, each of which takes on values in $[n] \cup \{\bot\}$, initially $\bot$. The $choose$ action is enabled when $y = \bot$, and sets y to a number in $[n]$, chosen uniformly at random. The $compute$ action is enabled when $y \neq \bot$ and $z = \bot$, and sets $z := f(y)$. The $report(k)$ action behaves exactly as in $Rand$. Precondition/effect code for $Trapdoor$ appears in Figure 3, and a diagram appears in Figure 4.

Trapdoor:
Signature:

Input:
none
Output:
$report(k), k \in \{1, \ldots, n\}$
Internal:
$choose, compute$

Tasks:
$Report = \{report(k) : k \in \{1, \ldots, n\}\}, Choose = \{choose\}, Compute = \{compute\}$
States:
$yval \in \{1, \ldots, n\} \cup \{\bot\}$, initially $\bot$
$zval \in \{1, \ldots, n\} \cup \{\bot\}$, initially $\bot$

Transitions:

$choose$
Precondition:
$yval = \bot$
Effect:
$yval :=$ random(uniform($\{1, \ldots, n\}$))

$compute$
Precondition:
$yval \neq \bot; zval = \bot$
Effect:
$zval := f(yval)$

$report(k)$
Precondition:
$zval = k$
Effect:
none

Figure 3: Code for Task-PIOA $Trapdoor$

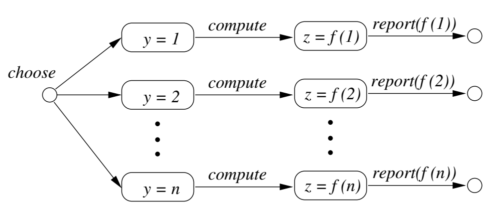

Figure 4: Task-PIOA $Trapdoor$

We want to use a simulation relation to prove that $\mathsf{tdists}(Trapdoor) \subseteq \mathsf{tdists}(Random)$. To do so, it is natural to allow the steps that define z to correspond in the two automata, which means the $choose$ steps of $Trapdoor$ (which define y) do not have corresponding steps in $Rand$. Note that, between the $choose$ and $compute$ in $Trapdoor$, a randomly-chosen value appears in the y component of the state of $Trapdoor$, but no such value appears in the corresponding state of $Rand$. Thus, the desired simulation relation should allow the correspondence between a probability measure on states of $Trapdoor$ and a single state of $Rand$.

We are able to express this correspondence using simulation relations in the sense of Definition 4.3: If ϵ_1

and ϵ_2 are discrete measures over finite execution fragments of $Trapdoor$ and $Rand$, respectively, then we say that ϵ_1 and ϵ_2 are related by R whenever the following conditions hold:

1. For every $s \in \mathsf{supp}(\mathsf{lstate}(\epsilon_1))$ and $u \in \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$, $s.z = u.z$.
2. For every $u \in \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$, if $u.z = \perp$ then either $\mathsf{lstate}(\epsilon_1).y$ is everywhere undefined or else it is the uniform distribution on $[n]$.

The task correspondence mapping c is defined by[2]

- $c(\rho, Choose) = \lambda$.
- If ρ contains the $Choose$ action, then $c(\rho, Compute) = Choose$; otherwise, $c(\rho, Compute) = \lambda$.
- $c(\rho, Report) = Report$.

5 Application to Security Protocols

In [CCK+06e], we use the task-PIOAs of this paper to model and analyze the Oblivious Transfer (OT) protocol of Goldreich et al. [GMW87].

In the OT problem, two input bits (x_0, x_1) are submitted to a Transmitter $Trans$ and a single input bit i to a Receiver Rec. After engaging in an OT protocol, Rec should output only the single bit x_i. Rec should not learn the other bit x_{1-i}, and $Trans$ should not learn i; moreover, an eavesdropping adversary should not, by observing the protocol messages, be able to learn anything about the inputs or the progress of the protocol. OT has been shown to be "complete" for multi-party secure computation, in the sense that, using OT as the only cryptographic primitive, one can construct protocols that securely realize any functionality.

The protocol of [GMW87] uses trap-door permutations (and hard-core predicates) as an underlying cryptographic primitive. It uses three rounds of communication: First, $Trans$ chooses a random trap-door permutation f and sends it to Rec. Second, Rec chooses two random numbers (y_0, y_1) and sends (z_0, z_1) to $Trans$, where z_i for the input index i is $f(y_i)$ and $z_{1-i} = y_{1-i}$. Third, $Trans$ applies the same transformation to each of z_0 and z_1 and sends the results back as (b_0, b_1) Finally, Rec decodes and outputs the correct bit. The protocol uses cryptographic primitives and computational hardness in an essential way. Its security is inherently only computational, so its analysis requires modeling computational assumptions.

Our analysis follows the *trusted party* paradigm of [GMW87], with a formalization that is close in spirit to [PW00, Can01]. We first define task-PIOAs representing the *real system (RS)* (the protocol) and the *ideal system (IS)* (the requirements). In RS, typical tasks include "choose random (y_0, y_1)", "send round 1 message", and "deliver round 1 message", as well as arbitrary tasks of environment and adversary automata. (The environment and adversary automata are purposely under-specified, so that our results are as general as possible.) Note that these tasks do not specify exactly what transition occurs. For example, the "choose" task does not specify the chosen values of (y_0, y_1). And the "send" task does not specify the message contents—these are computed by $Trans$, based on its own internal state.

Then we prove that RS implements IS. The proof consists of four cases, depending on which parties are corrupted[3]. In the two cases where $Trans$ is corrupted, we can show that RS implements IS unconditionally, using $\leq_0$. In the cases where $Trans$ is not corrupted, we can show implementation only in a "computational" sense, namely, (i) for resource-bounded adversaries, (ii) up to negligible differences, and (iii) under computational hardness assumptions. Modeling these aspects requires additions to the task-PIOA framework of this paper, namely, defining a *time-bounded* version of task-PIOAs, and defining a variation,

[2] In an extended abstract of this report[CCK+06b], the definition of c contains a small error. Namely, in the second clause, $c(\rho, Compute)$ is set to $Choose$ regardless of the condition on ρ.

[3] In [CCK+06e], only one case is treated in full detail—when only Rec is corrupted. We prove all four cases in [CCK+05], but using a less general definition of task-PIOAs than the one used here and in [CCK+06e], and with non-branching adversaries.

$\leq_{neg,pt}$, on the $\leq_0$ relation, which describes approximate implementation with respect to polynomial-time-bounded environments. Similar relations were defined in [LMMS98, PW01]. Our simulation relations are also sound with respect to $\leq_{neg,pt}$.

We also provide models for the cryptographic primitives (trap-door functions and hard-core predicates). Part of the specification for such primitives is that their behavior should look "approximately random" to outside observers; we formalize this in terms of $\leq_{neg,pt}$.

The correctness proofs proceed by levels of abstraction, relating each pair of models at successive levels using $\leq_{neg,pt}$. In the case where only Rec is corrupted, all but one of the relationships between levels are proved using simulation relations as defined in this paper (and so, they guarantee $\leq_0$). The only exception relates a level in which the cryptographic primitive is used, with a higher level in which the use of the primitive is replaced by a random choice. Showing this correspondence relies on our $\leq_{neg,pt}$-based definition of the cryptographic primitive, and on composition results for time-bounded task-PIOAs. Since this type of reasoning is isolated to one correspondence, the methods of this paper in fact suffice to accomplish most of the work of verifying OT.

Each of our system models, at each level, includes an explicit adversary component automaton, which acts as a message delivery service that can eavesdrop on communications and control the order of message delivery. The behavior of this adversary is arbitrary, subject to general constraints on its capabilities. In our models, the adversary is the same at all levels, so our simulation relations relate the adversary states at consecutive levels directly, using the identity function. This treatment allows us to consider arbitrary adversaries without examining their structure in detail (they can do anything, but must do the same thing at all levels).

Certain patterns that arise in our simulation relation proofs led us to extend earlier definitions of simulation relations [SL95, LSV03], by adding the expansion capability and by corresponding measures to measures:

1. We often correspond random choices at two levels of abstraction—for instance, when the adversary makes a random choice, from the same state, at both levels. We would like our simulation relation to relate the individual outcomes of the choices at the two levels, matching up the states in which the same result is obtained. Modeling this correspondence uses the expansion feature.

2. The $Trapdoor$ vs. $Rand$ example described in Section 4 occurs in our OT proof. Here, the low-level system chooses a random y and then computes $z = f(y)$ using a trap-door permutation f. The higher level system simply chooses the value of z randomly, without using value y or permutation f. This correspondence relates measures to measures and uses expansion.

3. In another case, a lower-level system chooses a random value y and then computes a new value by applying XOR to y and an input value. The higher level system just chooses a random value. We establish a correspondence between the two levels using the fact that XOR preserves the uniform distribution. This correspondence again relates measures to measures and uses expansion.

6 Local Schedulers

With the action-determinism assumption, our task mechanism is enough to resolve all nondeterminism. However, action determinism limits expressive power. Now we remove this assumption and add a second mechanism for resolving the resulting additional nondeterminism, namely, a *local scheduler* for each component task-PIOA. A local scheduler for a given component can be used to resolve nondeterministic choices among actions in the same task, using only information about the past history of that component. Here, we define one type of local scheduler, which uses only the current state, and indicate how our results for the action-deterministic case carry over to this setting.

Our notion of local scheduler is simply a "sub-automaton": We could add more expressive power by allowing the local scheduler to depend on the past execution. This could be formalized in terms of an explicit function of the past execution, or perhaps in terms of a refinement mapping or other kind of simulation relation.

Definition 6.1 *We say that task-PIOA $\mathcal{T}' = (\mathcal{P}', R')$ is a* sub-task-PIOA *of task-PIOA $\mathcal{T} = (\mathcal{P}, R)$ provided that all components are identical except that $D' \subseteq D$, where D and D' are the sets of discrete transitions of $\mathcal{P}$ and $\mathcal{P}'$, respectively. Thus, the only difference is that $\mathcal{T}'$ may have a smaller set of transitions.*

Definition 6.2 *A* local scheduler *for a task-PIOA $\mathcal{T}$ is any action-deterministic sub-task-PIOA of $\mathcal{T}$. A* probabilistic system *is a pair $\mathcal{M} = (\mathcal{T}, \mathcal{S})$, where $\mathcal{T}$ is a task-PIOA and $\mathcal{S}$ is a set of local schedulers for $\mathcal{T}$.*

Definition 6.3 *A* probabilistic execution *of a probabilistic system $\mathcal{M} = (\mathcal{T}, \mathcal{S})$ is defined to be any probabilistic execution of any task-PIOA $S \in \mathcal{S}$.*

We next define composition for probabilistic systems.

Definition 6.4 *If $\mathcal{M}_1 = (\mathcal{T}_1, \mathcal{S}_1)$ and $\mathcal{M}_2 = (\mathcal{T}_2, \mathcal{S}_2)$ are two probabilistic systems, and $\mathcal{T}_1$ and $\mathcal{T}_2$ are compatible, then their* composition *$\mathcal{M}_1 \| \mathcal{M}_2$ is the probabilistic system $(\mathcal{T}_1 \| \mathcal{T}_2, \mathcal{S})$, where $\mathcal{S}$ is the set of local schedulers for $\mathcal{T}_1 \| \mathcal{T}_2$ of the form $S_1 \| S_2$, for some $S_1 \in \mathcal{S}_1$ and $S_2 \in \mathcal{S}_2$.*

Definition 6.5 *If $\mathcal{M} = (\mathcal{T}, \mathcal{S})$ is a probabilistic system, then an* environment *for $\mathcal{M}$ is any environment (action-deterministic task-PIOA) for $\mathcal{T}$. If $\mathcal{M} = (\mathcal{T}, \mathcal{S})$ is a probabilistic system, then the* external behavior *of $\mathcal{M}$, $\mathsf{extbeh}(\mathcal{M})$, is the total function that maps each environment task-PIOA $\mathcal{E}$ for $\mathcal{M}$ to the set $\bigcup_{S \in \mathcal{S}} \mathsf{tdists}(S \| \mathcal{E})$.*

Thus, for each environment, we consider the set of trace distributions that arise from two choices: of a local scheduler of $\mathcal{M}$ and of a global task schedule ρ.

Definition 6.6 *Two probabilistic systems $(\mathcal{T}_1, \mathcal{S}_1)$ and $(\mathcal{T}_2, \mathcal{S}_2)$ are* comparable *if $\mathcal{T}_1$ and $\mathcal{T}_2$ are comparable task-PIOAs.*

We define an implementation relation for comparable probabilistic systems in terms of inclusion of sets of trace distributions for each probabilistic system based on an environment task-PIOA:

Definition 6.7 *If $\mathcal{M}_1 = (\mathcal{T}_1, \mathcal{S}_1)$ and $\mathcal{M}_2 = (\mathcal{T}_2, \mathcal{S}_2)$ are comparable probabilistic systems (i.e., $\mathcal{T}_1$ and $\mathcal{T}_2$ are comparable), then $\mathcal{M}_1$* implements *$\mathcal{M}_2$, written $\mathcal{M}_1 \leq_0 \mathcal{M}_2$, provided that $\mathsf{extbeh}(\mathcal{M}_1)(\mathcal{E}) \subseteq \mathsf{extbeh}(\mathcal{M}_2)(\mathcal{E})$ for every environment (action-deterministic) task-PIOA $\mathcal{E}$ for both $\mathcal{M}_1$ and $\mathcal{M}_2$.*

We obtain a sufficient condition for implementation of probabilistic systems, in which each local scheduler for the low-level system always corresponds to the same local scheduler of the high-level system.

Theorem 6.8 *Let $\mathcal{M}_1 = (\mathcal{T}_1, \mathcal{S}_1)$ and $\mathcal{M}_2 = (\mathcal{T}_2, \mathcal{S}_2)$ be two comparable probabilistic systems. Suppose there is a total function f from $\mathcal{S}_1$ to $\mathcal{S}_2$ such that, for every $S_1 \in \mathcal{S}_1$, $S_1 \leq_0 f(S_1)$. Then $\mathcal{M}_1 \leq_0 \mathcal{M}_2$.*

We also obtain a compositionality result for probabilistic systems. The proof is similar to that of Theorem 3.24, for the action-deterministic case.

Theorem 6.9 *Let $\mathcal{M}_1$, $\mathcal{M}_2$ be comparable probabilistic systems such that $\mathcal{M}_1 \leq_0 \mathcal{M}_2$, and let $\mathcal{M}_3$ be a probabilistic system compatible with each of $\mathcal{M}_1$ and $\mathcal{M}_2$. Then $\mathcal{M}_1 \| \mathcal{M}_3 \leq_0 \mathcal{M}_2 \| \mathcal{M}_3$.*

Proof. Let $\mathcal{T}_4 = (\mathcal{P}_4, R_4)$ be any environment (action-deterministic) task-PIOA for both $\mathcal{M}_1 \| \mathcal{M}_3$ and $\mathcal{M}_2 \| \mathcal{M}_3$. Let $\mathcal{M}_4$ be the trivial probabilistic system $(\mathcal{T}_4, \{\mathcal{T}_4\})$. Fix any task schedule ρ_1 for $(\mathcal{T}_1 \| \mathcal{T}_3) \| \mathcal{T}_4$ and local scheduler $\mathcal{P}'_{13}$ of $\mathcal{M}_1 \| \mathcal{M}_3$. Let τ be the trace distribution of $(\mathcal{T}_1 \| \mathcal{T}_3) \| \mathcal{T}_4$ generated by ρ_1 and $\mathcal{P}'_{13}$. It suffices to show that τ is also generated by some task schedule ρ_2 for $(\mathcal{T}_2 \| \mathcal{T}_3) \| \mathcal{T}_4$, local scheduler $\mathcal{P}'_{23}$ of $\mathcal{M}_2 \| \mathcal{M}_3$, and $\mathcal{P}_4$.

Note that ρ_1 is also a task schedule for $\mathcal{T}_1 \| (\mathcal{T}_3 \| \mathcal{T}_4)$. Since $\mathcal{P}'_{13}$ is a local scheduler of $\mathcal{M}_1 \| \mathcal{M}_3$, it is (by definition) of the form $\mathcal{P}'_1 \| \mathcal{P}'_3$, where $\mathcal{P}'_1 \in \mathcal{S}_1$ and $\mathcal{P}'_3 \in \mathcal{S}_3$. Let $\mathcal{P}'_{34} = \mathcal{P}'_3 \| \mathcal{P}_4$. Then $\mathcal{P}'_{34}$ is a local scheduler of $\mathcal{M}_3 \| \mathcal{M}_4$. Then, ρ_1, $\mathcal{P}'_1$, and $\mathcal{P}'_{34}$ generate the same trace distribution τ in the composed task-PIOA $\mathcal{T}_1 \| (\mathcal{T}_3 \| \mathcal{T}_4)$.

Define $\mathcal{T}_5$ to be the task-PIOA $\mathcal{T}_3 \| \mathcal{T}_4$. Note that $\mathcal{T}_5$ is an environment task-PIOA for each of $\mathcal{T}_1$ and $\mathcal{T}_2$. Define the probabilistic system $\mathcal{M}_5$ to be $(\mathcal{T}_5, \{\mathcal{P}'_{34}\})$, that is, we consider just a singleton set of local schedulers, containing the one scheduler we are actually interested in.

Now, by assumption, $\mathcal{M}_1 \leq_0 \mathcal{M}_2$. Therefore, there exists a task schedule ρ_2 for $\mathcal{T}_2 \| \mathcal{T}_5$ and a local scheduler $\mathcal{P}'_2$ for $\mathcal{P}_2$ such that ρ_2, $\mathcal{P}'_2$, and $\mathcal{P}'_{34}$ generate the same trace distribution τ in the task-PIOA $\mathcal{T}_2 \| \mathcal{T}_5$. Note that ρ_2 is also a task schedule for $(\mathcal{T}_2 \| \mathcal{T}_3) \| \mathcal{T}_4$. Let $\mathcal{P}'_{23} = \mathcal{P}'_2 \| \mathcal{P}'_3$. Then $\mathcal{P}'_{23}$ is a local scheduler of $\mathcal{M}_2 \| \mathcal{M}_3$. Also, $\mathcal{P}'_4$ is a local scheduler of $\mathcal{M}_4$. Then ρ_2, $\mathcal{P}'_{23}$ and $\mathcal{P}'_4$ also generate τ, which suffices to show the required implementation relationship. □

7 Conclusions

We have extended the traditional PIOA model with a task mechanism, which provides a systematic way of resolving nondeterministic scheduling choices without using information about past history. We have provided basic machinery for using the resulting task-PIOA framework for verification, including a compositional trace-based semantics and a new kind of simulation relation. We have proposed extending the framework to allow additional nondeterminism, resolved by schedulers that use only local information. We have illustrated the utility of these tools with a case study involving analysis of an Oblivious Transfer cryptographic protocol.

Although our development was motivated by concerns of cryptographic protocol analysis, the notion of partial-information scheduling is interesting in other settings. For example, some distributed algorithms work with partial-information adversarial schedulers, in part because the problems they address are provably unsolvable with perfect-information adversaries [Cha96, Asp03]. Also, partial-information scheduling is realistic for modeling large distributed systems, in which basic scheduling decisions are made locally, and not by any centralized mechanism.

Many questions remain in our study of task-PIOAs: First, our notions of external behavior and of implementation ($\leq$) for task-PIOAs are defined by considering the behavior of the task-PIOAs in all environments. It would be interesting to characterize this implementation relation using a smaller subclass of environments, that is, to find a small (perhaps minimal) subclass such that $\mathcal{T}_1 \leq_0 \mathcal{T}_2$ if and only if $\mathsf{extbeh}(\mathcal{T}_1)(\mathcal{E}) \subseteq \mathsf{extbeh}(\mathcal{T}_2)(\mathcal{E})$ for every $\mathcal{E}$ in the subclass.

Second, it would be interesting to develop other kinds of simulation relations, perhaps simpler than the one defined here. For example, we would like to reformulate our current simulation relation notion in terms of states rather than finite executions, and to understand whether there are simulation relations for task-PIOAs that have the power of *backward simulations* [LV95]. It will also be useful to identify a class of simulation relations that is *complete* for showing implementation ($\leq_0$) of task-PIOAs.

Third, our notion of local schedulers needs further development. Perhaps it can be generalized to allow history-dependence. We would like better connections between the results on local schedulers and the rest of the basic theory of action-deterministic task-PIOAs; in particular, we would like to be able to use results from the action-deterministic case to help prove results for the case with local schedulers. Finally, it remains to apply the model with local schedulers to interesting distributed algorithm or security protocol examples.

In general, it remains to consider more applications of task-PIOAs, for cryptographic protocol analysis and for other applications. A next step in cryptographic protocol analysis is to formulate and prove protocol composition results like those of [PW01, Can01] in terms of task-PIOAs. In particular, we would like to pursue a full treatment of Canetti's Universal Composability results [Can01] in terms of task-PIOAs. This would provide a full-featured modeling framework for security protocols, which can express computational notions as in [Can01], while inheriting the simplicity and modularity of the task-PIOAs foundation.

It would also be interesting to try to model perfect-information schedulers, as used for analyzing randomized distributed algorithms, using task-PIOAs. Finally, it remains to extend the definitions in this paper to incorporate timing-dependent behavior and hybrid continuous/discrete behavior, and to prove theorems analogous to the ones in this paper for those extensions. Preliminary results in this direction appear in [ML06].

Acknowledgments: We thank Frits Vaandrager for collaboration in early stages of this project, and Michael Backes, Anupam Datta, Joshua Guttman, Jon Herzog, Susan Hohenberger, Ralf Kuesters, John Mitchell, Birgit Pfitzmann, and Andre Scedrov for technical discussions that helped us in clarifying our ideas and their connections to other work in analysis of cryptographic protocols. We thank Silvio Micali for impressing upon us the importance of adaptive adversarial schedulers in the cryptographic setting. We thank Sayan Mitra both for technical discussions and for help in producing the paper.

References

[AH90] J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441–461, 1990.

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

[BO83] M. Ben-Or. Another advantage of free choice: completely asynchronous agreement protocols. In *Proc. 2nd ACM Symposium on Principles of Distributed Computing*, pages 27–30, Montreal, Quebec, Canada, August 1983.

[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 *Proc. 42nd IEEE Symposium on Foundations of Computing*, pages 136–145, 2001.

[CCK+05] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Using probabilistic I/O automata to analyze an oblivious transfer protocol. Technical Report MIT-LCS-TR-1001a or MIT-CSAIL-TR-2005-055, MIT CSAIL, 2005.

[CCK+06a] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Task-structured probabilistic I/O automata. Technical Report MIT-CSAIL-TR-2006-023, MIT CSAIL, 2006.

[CCK+06b] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Task-structured probabilistic I/O automata. In *Proceedings of the 8th International Workshop on Discrete Event Systems (WODES'06)*, 2006. Ann Arbor, Michigan, July 2006.

[CCK+06c] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Time-bounded task-PIOAs: a framework for analyzing security protocols. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC '06)*, 2006. Stockholm, Sweden, September 2006.

[CCK+06d] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Using probabilistic I/O automata to analyze an oblivious transfer protocol. Technical Report MIT-CSAIL-TR-2006-046, MIT CSAIL, 2006.

[CCK+06e] R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Using task-structured probabilistic I/O automata to analyze an oblivious transfer protocol. Technical Report MIT-CSAIL-TR-2006-047, MIT CSAIL, 2006.

[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 *Proc. 15th ACM Symposium on Principles of Distributed Computing*, pages 166–175, 1996.

[CLSV] L. Cheung, N. Lynch, R. Segala, and F. Vaandrager. Switched PIOA: Parallel composition via distributed scheduling. To appear in *TCS*, Special Issue on *FMCO 2004*.

[dA99] L. de Alfaro. The verification of probabilistic systems under memoryless partial-information policies is hard. In *Proc. PROBMIV 99*, pages 19–32, 1999.

[DKMR05] A. Datta, R. Küsters, J.C. Mitchell, and A. Ramanathan. On the relationships between notions of simulation-based security. In *Proceedings TCC 2005*, pages 476–494, 2005.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proc. 19th ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[Gol01] O. Goldreich. *Foundations of Cryptography: Basic Tools*, volume I. Cambridge University Press, 2001.

[KLC98] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1998.

[LMMS98] P. Lincoln, J.C. Mitchell, M. Mitchell, and A. Scedrov. A probabilistic poly-time framework for protocol analysis. In *ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[LSS94] N.A. Lynch, I. Saias, and R. Segala. Proving time bounds for randomized distributed algorithms. In *Proc. 13th 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 *Proc. 14th International Conference on Concurrency Theory (CONCUR 2003)*, volume 2761 of *LNCS*, 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.

[LV95] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations Part I: untimed systems. *Information and Computation*, 121(2):214–233, September 1995.

[ML06] S. Mitra and N.A. Lynch. Probabilistic timed I/O automata with continuous state spaces. Preliminary version available at `http://theory.lcs.mit.edu/~mitras/research/csptioa_preprint.pdf`, May 2006.

[MMS03] P. Mateus, J.C. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In *Proc. 14th International Conference on Concurrency Theory (CONCUR 2003)*, pages 323–345, 2003.

[MRST06] J. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *Theoretical Computer Science*, 353:118–164, 2006.

[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.

[Put94] M.L. Puterman. *Markov Decision Process – Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, 1994.

[PW00] B. Pfitzman and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proc. 7th ACM Conference on Computer and Communications Security (CCS 2000)*, pages 245–254, 2000.

[PW01] B. Pfitzman and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 184–200, 2001.

[Rab82] M. Rabin. The choice coordination problem. *Acta Informatica*, 17:121–134, 1982.

[SdV04] A. Sokolova and E.P. de Vink. Probabilistic automata: system types, parallel composition and comparison. In *Validation of Stochastic Systems*, volume 2925 of *LNCS*, 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.

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

A σ-Fields of Execution Fragments and Traces

In order to define probability measures on executions and traces, we need appropriate σ-fields. We begin with a σ-field over the set of execution fragments of a PIOA $\mathcal{P}$:

Definition A.1 *The* cone *of a finite execution fragment α, denoted by C_α, is the set $\{\alpha' \in \mathsf{Frags}(\mathcal{P}) \mid \alpha \leq \alpha'\}$. Then $\mathcal{F}_\mathcal{P}$ is the σ-field generated by the set of cones of finite execution fragments of $\mathcal{P}$.*

A *probability measure on execution fragments of* $\mathcal{P}$ is then simply a probability measure on the σ-field $\mathcal{F}_\mathcal{P}$.

Since Q, I, O, and H are countable, $\mathsf{Frags}^*(\mathcal{P})$ is countable, and hence the set of cones of finite execution fragments of $\mathcal{P}$ is countable. Therefore, any union of cones is measurable. Moreover, for each finite execution fragment α, the set $\{\alpha\}$ is measurable since it can be expressed as the intersection of C_α with the complement of $\cup_{\alpha':\alpha<\alpha'} C_{\alpha'}$. Thus, any set of finite execution fragments is measurable; in other words, the discrete σ-field of finite executions is included in $\mathcal{F}_\mathcal{P}$.

We often restrict our attention to probability measures on finite execution fragments, rather than those on arbitrary execution fragments. Thus, we define:

Definition A.2 *Let ϵ be a probability measure on execution fragments of $\mathcal{P}$. We say that ϵ is* finite *if $\mathsf{Frags}^*(\mathcal{P})$ is a support for ϵ.*

Since any set of finite execution fragments is measurable, any finite probability measure on execution fragments of $\mathcal{P}$ can also be viewed as a discrete probability measure on $\mathsf{Frags}^*(\mathcal{P}P)$. Formally, given any finite probability measure ϵ on execution fragments of $\mathcal{P}$, we obtain a discrete probability measure $\mathsf{finite}(\epsilon)$ on $\mathsf{Frags}^*(\mathcal{P})$ by simply defining $\mathsf{finite}(\epsilon)(\alpha) = \epsilon(\{\alpha\})$ for every finite execution fragment α of $\mathcal{P}$. The difference between $\mathsf{finite}(\epsilon)$ and ϵ is simply that the domain of ϵ is $\mathcal{F}_\mathcal{P}$, whereas the domain of $\mathsf{finite}(\epsilon)$ is $\mathsf{Execs}^*(\mathcal{P})$. Henceforth, we will ignore the distinction between $\mathsf{finite}(\epsilon)$ and ϵ.

Definition A.3 *Let ϵ and ϵ' be probability measures on execution fragments of PIOA $\mathcal{P}$. Then we say that ϵ is a* prefix *of ϵ', denoted by $\epsilon \leq \epsilon'$, if, for each finite execution fragment α of $\mathcal{P}$, $\epsilon(C_\alpha) \leq \epsilon'(C_\alpha)$.*

Definition A.4 *A* chain *of probability measures on execution fragments of PIOA $\mathcal{P}$ is an infinite sequence, $\epsilon_1, \epsilon_2, \cdots$ of probability measures on execution fragments of $\mathcal{P}$ such that, for each $i \geq 0$, $\epsilon_i \leq \epsilon_{i+1}$. Given a chain $\epsilon_1, \epsilon_2, \ldots$ of probability measures on execution fragments of $\mathcal{P}$, we define a new function ϵ on the σ-field generated by cones of execution fragments of $\mathcal{P}$ as follows: for each finite execution fragment α,*

$$\epsilon(C_\alpha) = \lim_{i \to \infty} \epsilon_i(C_\alpha).$$

Standard measure theoretic arguments ensure that ϵ can be extended uniquely to a probability measure on the σ-field generated by the cones of finite execution fragments. Furthermore, for each $i \geq 0$, $\epsilon_i \leq \epsilon$. We call ϵ the limit *of the chain, and we denote it by $\lim_{i\to\infty} \epsilon_i$.*

If α is a finite execution fragment of a PIOA $\mathcal{P}$ and a is an action of $\mathcal{P}$, then $C_{\alpha a}$ denotes the set of execution fragments of $\mathcal{P}$ that start with αa. The cone construction can also be used to define a σ-field of traces:

Definition A.5 *The* cone *of a finite trace β, denoted by C_β, is the set $\{\beta' \in E^* \cup E^\omega \mid \beta \leq \beta'\}$, where $\leq$ denotes the prefix ordering on sequences. The σ-field of traces of $\mathcal{P}$ is simply the σ-field generated by the set of cones of finite traces of $\mathcal{P}$.*

Again, the set of cones is countable and the discrete σ-field on finite traces is included in the σ-field generated by cones of traces. We often refer to a probability measure on the σ-field generated by cones of traces of a PIOA $\mathcal{P}$ as simply a *probability measure on traces of $\mathcal{P}$*.

Definition A.6 *Let τ be a probability measure on traces of $\mathcal{P}$. We say that τ is* finite *if the set of finite traces is a support for τ. Any finite probability measure on traces of $\mathcal{P}$ can also be viewed as a discrete probability measure on the set of finite traces.*

Definition A.7 *Let τ and τ' be probability measures on traces of PIOA $\mathcal{P}$. Then we say that τ is a* prefix *of τ', denoted by $\tau \leq \tau'$, if, for each finite trace β of $\mathcal{P}$, $\tau(C_\beta) \leq \tau'(C_\beta)$.*

Definition A.8 *A* chain *of probability measures on traces of PIOA $\mathcal{P}$ is an infinite sequence, $\tau_1, \tau_2, \cdots$ of probability measures on traces of $\mathcal{P}$ such that, for each $i \geq 0$, $\tau_i \leq \tau_{i+1}$. Given a chain $\tau_1, \tau_2, \ldots$ of probability measures on traces of $\mathcal{P}$, we define a new function τ on the σ-field generated by cones of traces of $\mathcal{P}$ as follows: for each finite trace β,*

$$\tau(C_\beta) = \lim_{i \to \infty} \tau_i(C_\beta).$$

Then τ can be extended uniquely to a probability measure on the σ-field of cones of finite traces. Furthermore, for each $i \geq 0$, $\tau_i \leq \tau$. We call τ the limit *of the chain, and we denote it by $\lim_{i\to\infty} \tau_i$.*

Recall from Section 2.3 the definition of the trace distribution $\mathsf{tdist}(\epsilon)$ of a probability measure ϵ on execution fragments. Namely, $\mathsf{tdist}(\epsilon)$ is the image measure of ϵ under the measurable function trace.

Lemma A.9 *Let $\epsilon_1, \epsilon_2, \cdots$ be a chain of measures on execution fragments, and let ϵ be $\lim_{i\to\infty} \epsilon_i$. Then $\lim_{i\to\infty} \mathsf{tdist}(\epsilon_i) = \mathsf{tdist}(\epsilon)$.*

Proof. It suffices to show that, for any finite trace β, $\lim_{i\to\infty} \mathsf{tdist}(\epsilon_i)(C_\beta) = \mathsf{tdist}(\epsilon)(C_\beta)$. Fix a finite trace β.

Let Θ be the set of minimal execution fragments whose trace is in C_β. Then $\mathsf{trace}^{-1}(C_\beta) = \cup_{\alpha\in\Theta} C_\alpha$, where all the cones are pairwise disjoint. Therefore, for $i \geq 0$, $\mathsf{tdist}(\epsilon_i)(C_\beta) = \sum_{\alpha\in\Theta} \epsilon_i(C_\alpha)$, and $\mathsf{tdist}(\epsilon)(C_\beta) = \sum_{\alpha\in\Theta} \epsilon(C_\alpha)$.

Since we have monotone limits here (that is, our limit are also suprema), limits commute with sums and our goal can be restated as showing:

$$\sum_{\alpha\in\Theta} \lim_{i\to\infty} \epsilon_i(C_\alpha) = \sum_{\alpha\in\Theta} \epsilon(C_\alpha).$$

Since $\lim_{i\to\infty} \epsilon_i = \epsilon$, we have $\lim_{i\to\infty} \epsilon_i(C_\alpha) = \epsilon(C_\alpha)$ for each finite execution fragment α. Therefore, the two sums above are in fact equal. □

The lstate function is a measurable function from the discrete σ-field of finite execution fragments of $\mathcal{P}$ to the discrete σ-field of states of $\mathcal{P}$. If ϵ is a probability measure on execution fragments of $\mathcal{P}$, then we define the lstate distribution of ϵ, $\mathsf{lstate}(\epsilon)$, to be the image measure of ϵ under the function lstate.