Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2006-023 March 31, 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[1,2], Ling Cheung[3], Dilsun Kaynar[1], Moses Liskov[4], Nancy Lynch[1], Olivier Pereira[5], and Roberto Segala[6]

[1] MIT Computer Science and Artificial Intelligence Laboratory, USA
[2] IBM TJ Watson Research Center, USA
[3] Radboud University of Nijmegen, the Netherlands, lcheung@cs.kun.nl
[4] College of William and Mary, USA
[5] Université catholique de Louvain, Belgium
[6] Università di Verona, Italy, roberto.segala@univr.it

Preliminary Technical Report Version of March 6, 2006

Abstract. In the Probabilistic I/O Automata (PIOA) framework, nondeterministic choices are resolved using perfect-information schedulers, which are similar to history-dependent policies for Markov decision processes (MDPs). These schedulers are too powerful in the setting of security analysis, leading to unrealistic adversarial behaviors. Therefore, we introduce in this paper a novel mechanism of task partitions for PIOAs. This allows us to define partial-information adversaries in a systematic manner, namely, via sequences of tasks.
The resulting task-PIOA framework comes with simple notions of external behavior and implementation, and supports simple compositionality results. A new type of simulation relation is defined and proven sound with respect to our notion of implementation. To illustrate the potential of this framework, we summarize our verification of an Oblivious Transfer protocol, where we combine formal and computational analyses. Finally, we present an extension with extra expressive power, using local schedulers of individual components.

1 Introduction

The framework of *Probabilistic Input/Output Automata (PIOA)* is a simple combination of I/O automata [LT89] and Markov decision processes [Put94]. As demonstrated in [LSS94,SV99,PSL00], PIOAs are well-suited for the analysis of distributed algorithms that use randomness as a computational primitive (e.g., [Rab82]). In this setting, distributed processes use random inputs to improve complexity results for important tasks such as consensus. Thus, each process is modeled as an automaton with randomized transitions, while the protocol is modeled as the (asynchronous) parallel composition of all participants.

This modeling paradigm combines nondeterministic and probabilistic choices in a very natural way. This is attractive because, in our experience, nondeterminism plays several important roles in modeling and verification. First, it provides a convincing option for modeling timing uncertainties in a distributed and highly

unpredictable environment. Moreover, it allows us to model systems at high levels of abstraction, where many details are left unspecified. These features in turn support descriptions of system decomposition, both "horizontally" (based on parallel composition) and "vertically" (based on levels of abstraction).

This paper presents an adaptation of the PIOA framework intended for the analysis of cryptographic protocols. These protocols are highly specialized examples of distributed algorithms, designed to protect sensitive data when they are transmitted over unreliable channels. Their correctness typically relies on computational assumptions, which state that certain problems cannot be solved (efficiently) by an adversarial entity with bounded computation resources [Gol01].

The Interactive Turing Machine (ITM) framework can be used to analyze cryptographic protocols on a very detailed level: participants are modeled as ITMs and messages as bit strings written on input tapes [Can01]. In this setting, complexity-theoretic arguments can be given rigorously, reducing the correctness of a protocol to its underlying cryptographic primitives. However, as protocols become more complex and involve more participants, such fine-grained analyses are cluttered by details and thus difficult to carry out. Practical protocols and adversaries are therefore seldom expressed in the language of ITMs. Rather, an informal (and at times ambiguous) higher-level language is used.

We aim to provide a framework in which security analysis can be carried out formally in the style of the ITM-based framework of [Can01], yet inessential details can be abstracted away in order to facilitate reasoning. The main challenge we face is the reconciliation of two different notions of scheduling.

Modeling Adversarial Scheduling The standard scheduling mechanism in the cryptographic community is that of an *adversarial* scheduler, namely, a resource-bounded algorithmic entity that determines the next move to be taken based on the adversarial view of the computation so far. In contrast, the standard in distributed computing is a *perfect-information* scheduler, which has access to local state and history of every component and has no limitations on its computation power. The latter is clearly too strong for cryptographic analysis.

Our solution is a clean separation of concerns. The adaptive adversarial scheduler is modeled explicitly as an automaton, which acts as a message delivery system and thus has access to dynamic information. On the other hand, the remaining nondeterministic choices are resolved by a sequence of *tasks* that is fixed nondeterministically in advance. These tasks are equivalence classes of action symbols, so that dynamic information is kept independent.

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 Oblivious Transfer (OT) protocol [GMW87], discussed later, both Transmitter and Receiver invoke an appropriate random source, but it is inconsequential which of the two does so first.

Our Contribution We define notions of external behavior and implementation for *task-PIOAs*, which are PIOAs augmented with *task partitions* on the set of actions. These notions are 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.

A new type of simulation relation, which incorporates the notion of tasks, is also proposed and proven sound. It is very different from simulation relations studied earlier [SL95,LSV03], in that it is a relation between probability distributions. This definition style is directly motivated by proof techniques in cryptography, where certain probability distributions are deemed "indistinguishable" by any resource-bounded entity.

The task-PIOA framework has been applied successfully in security analysis [CCK+06]. Specifically, it is used to analyze the Oblivious Transfer protocol mentioned above. That analysis involves a fair amount of additional work, such as definitions of computationally bounded computation, approximate implementation and computational hardness assumptions. These details are beyond the scope of the current paper, but we will outline and discuss the basic modeling approach in Section 4.

Related Work The literature contains numerous models that combine nondeterministic and probabilistic choices (see [SdV04] for a survey). However, very few tackle the issue of partial-information scheduling, as we do here. 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 [KLA98]. All of these accounts neglect partial information aspects of (parameterized) actions, therefore are not suitable in a cryptographic setting.

On the other hand, several attempts have been made in the security literature to incorporate cryptographic analysis into conventional models of concurrency [LMMS98,PW00,Can01,PW01,MMS03]. However, scheduling is usually determined under specific rules and assumptions, which do not always reflect properties of real-life distributed environments. In [LMMS98], for example, a uniform distribution is imposed on the set of possible reductions for each term. In [MMS03], internal reductions are prioritized over external communications and a host of independence assumptions are imposed. In [PW01], a distributed scheduling scheme is used to avoid unrealistic dependencies, but synchronization is modeled at a very detailed level, using explicit message buffers and scheduling ports. In our opinion, the approach taken here (separation of high- and low-level scheduling) is conceptually simpler and still yields a faithful model of concurrent behavior. Moreover, the fact that we allow ingrained nondeterministic choices allows us to apply well-established verification techniques from the area of distributed computing.

Road Map We define in Section 2 the notions of task-PIOAs, task schedules, composition and implementation. Also a compositionality result is given. Section 3 introduces our new simulation and the associated soundness theorem. Section 4 summarizes a case study in security protocol verification. Section 5 discusses an extension with local schedulers and concluding discussions follow in Section 6. Further details are available in appendices.

2 Task-PIOAs

Basic PIOAs We assume our reader is comfortable with basic notions of probability theory, such as σ-fields and (discrete) probability measures. A summary is provided in Appendix A.

A *probabilistic I/O automaton (PIOA)* $\mathcal{P}$ is a tuple $(Q, \bar{q}, I, O, H, D)$ where: (i) Q is a countable set of *states*, with *start state* $\bar{q} \in Q$; (ii) I, O and H are countable and pairwise disjoint sets of actions, referred to as *input, output and internal actions*, respectively; (iii) $D \subseteq (Q \times (I \cup O \cup H) \times \mathsf{Disc}(Q))$ 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 said to be *closed*. The set of *external* actions of $\mathcal{P}$ is $E := I \cup O$ and the set of *locally controlled* actions is $O \cup H$. We assume that $\mathcal{P}$ satisfies the following conditions.

- **Input enabling:** For every $q \in Q$ and $a \in I$, 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 $\mathsf{tran}_{q,a}$ for the transition $(q, a, \mu_{q,a})$.

An *execution fragment* of $\mathcal{P}$ is a sequence $\alpha = q_0\, a_1\, q_1\, a_2\, \ldots$ of alternating states and actions, such that (i) if α is finite, then it ends with a state; (ii) for every non-final i, there exists a transition $(q_i, a_{i+1}, \mu) \in D$ with $q_{i+1} \in \mathsf{supp}(\mu)$, where $\mathsf{supp}(\mu)$ denotes the support of μ. We write $\mathsf{fstate}(\alpha)$ for q_0, and, if α is finite, we write $\mathsf{lstate}(\alpha)$ for its last state. We use $\mathsf{Frags}(\mathcal{P})$ (resp. $\mathsf{Frags}^*(\mathcal{P})$) to denote the set of all (resp. finite) execution fragments of $\mathcal{P}$. An *execution* of $\mathcal{P}$ is an execution fragment beginning from the start state $\bar{q}$. Similarly, $\mathsf{Execs}(\mathcal{P})$ (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.

Nondeterministic choices in $\mathcal{P}$ are resolved by means of a *scheduler*, which is a function $\sigma : \mathsf{Frags}^*(\mathcal{P}) \longrightarrow \mathsf{SubDisc}(D)$ such that $(q, a, \mu) \in \mathsf{supp}(\sigma(\alpha))$ implies $q = \mathsf{lstate}(\alpha)$. Here $\mathsf{SubDisc}(D)$ is the set of discrete sub-probability distributions on D. Loosely speaking, σ decides (probabilistically) which transition 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 each cone $C_{\alpha'}$ is the set of execution fragments that have α' as a prefix. Detailed constructions can be found in Appendix B. The measure of a cone $C_{\alpha'}$ is defined recursively as follows:

$$\epsilon_{\sigma,\alpha}(C_{\alpha'}) := \begin{cases} 0 & \text{if } \alpha' \not\leq \alpha \text{ and } \alpha \not\leq \alpha' \\ 1 & \text{if } \alpha' \leq \alpha \\ \epsilon_{\sigma,\alpha}(C_{\alpha''})\mu_{\sigma(\alpha'')}(a, q) & \text{if } \alpha' = \alpha'' a q \text{ and } \alpha \leq \alpha'', \end{cases} \tag{1}$$

where $\mu_{\sigma(\alpha'')}(a,q) := \sigma(\alpha'')(\mathsf{tran}_{\mathsf{lstate}(\alpha''),a}) \cdot \mu_{\mathsf{lstate}(\alpha''),a}(q)$. That is, $\mu_{\sigma(\alpha'')}(a,q)$ is the probability that $\sigma(\alpha'')$ chooses a transition labeled by a and that the reached 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 $\mathcal{P}$.

Finally, we note that the trace function is a measurable function from $\mathcal{F}_\mathcal{P}$ to $\mathcal{F}$, where $\mathcal{F}$ is 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 denote by $\mathsf{tdists}(\mathcal{P})$ the set of trace distributions of (probabilistic executions of) $\mathcal{P}$.

Task-PIOAs We now augment the PIOA framework with task partitions, as a means to restrict resolution of nondeterminism. Unlike [CCK+05], we do not require that a task-PIOA has an equivalence relation on its states, nor that it satisfies the random-choice consistency, transition consistency and enabling consistency conditions that appear in [CCK+05]. The notion given here is also used in [CCK+06]. A *task-PIOA* is a pair $\mathcal{T} = (\mathcal{P}, R)$ where (i) $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ is a PIOA (satisfying transition determinism) and (ii) 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 referred to as *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 those of probabilistic executions and trace distributions.

For the time being, we impose the following assumption. This assumption will be removed in Section 5, when we introduce local schedulers. At the moment, it simplifies our technical development, because a sequence of tasks is sufficient to remove all nondeterministic choices. To make the later extensions easier, we will indicate explicitly where we are using the action-determinism hypothesis.

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

A *task schedule* for $\mathcal{T}$ is a finite or infinite sequence $\rho = T_1 T_2 \ldots$ of tasks in R. Such a task schedule resolves nondeterministic choices by repeatedly scheduling tasks, each of which determines at most one transition of the given task-PIOA. It is *static* (or *oblivious*), in the sense that it does not depend on dynamic information generated during execution. 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).

We claim that, under action determinism, every task schedule uniquely determines a probabilistic execution (and hence a trace distribution) in the underlying PIOA. To prove this, we define an operation that "applies" a task schedule to a task-PIOA.

Definition 1. *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 a probability measure on $\mathsf{Frags}(\mathcal{P})$. It is defined recursively as follows.*

1. $\mathsf{apply}(\mu, \lambda) := \mu$. *Here λ denotes the empty sequence.*
2. *For $T \in R$ and $\alpha \in \mathsf{Frags}^*(\mathcal{P})$, $\mathsf{apply}(\mu, T)(\alpha) := p_1(\alpha) + p_2(\alpha)$, where:*

$$p_1(\alpha) = \begin{cases} \mu(\alpha')\eta(q) & \text{if } \alpha = \alpha' a q,\ a \in T,\ (\mathsf{lstate}(\alpha'), a, \eta) \in D, \\ 0 & \text{otherwise;} \end{cases}$$

$$p_2(\alpha) = \begin{cases} \mu(\alpha) & \text{if } T \text{ is not enabled in } \mathsf{lstate}(\alpha) \\ 0 & \text{otherwise.} \end{cases}$$

3. *If ρ is finite and of the form $\rho' T$, then $\mathsf{apply}(\mu, \rho) := \mathsf{apply}(\mathsf{apply}(\mu, \rho'), T)$.*
4. *If ρ is infinite, let ρ_i denote the length-i prefix of ρ and let ϵ_i be $\mathsf{apply}(\mu, \rho_i)$. Then $\mathsf{apply}(\mu, \rho) := \lim_{i \to \infty}(\epsilon_i)$.*

It is routine to check that the limit in Case (4) is well-defined. More interesting is Case (2). The term $p_1(\alpha)$ represents the probability that α is executed as the result of applying task T at the end of α'. Due to transition- and action-determinism, the transition $(\mathsf{lstate}(\alpha'), a, \eta)$ is unique and thus p_1 is well-defined. The term $p_2(\alpha)$ is the original probability $\mu(\alpha)$, in case T is not enabled after α.

Proposition 1 below states that $\mathsf{apply}(\mu, \rho)$ is a generalized probabilistic execution fragment generated by μ and a scheduler σ for $\mathcal{P}$ in the usual sense. In other words, a task schedule for a task-PIOA is just a special kind of scheduler for the underlying PIOA. The proof can be found in Appendix B.4.

Proposition 1. *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. For each 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 $\epsilon_{\sigma,\mu}$.*

Any such $\mathsf{apply}(\mu, \rho)$ is said to be a *generalized probabilistic execution fragment* of $\mathcal{T}$. *Probabilistic execution fragments* and *probabilistic executions* are defined as for basic PIOAs. We write $\mathsf{tdist}(\mu, \rho)$ as shorthand for $\mathsf{tdist}(\mathsf{apply}(\mu, \rho))$ and $\mathsf{tdist}(\rho)$ for $\mathsf{tdist}(\mathsf{apply}(\delta(\bar{q}), \rho))$, where $\delta(\bar{q})$ denotes the measure that assigns probability 1 to $\bar{q}$. A *trace distribution* of $\mathcal{T}$ is any $\mathsf{tdist}(\rho)$. We use $\mathsf{tdists}(\mathcal{T})$ to denote the set $\{\mathsf{tdist}(\rho) : \rho \text{ is a task scheduler for } \mathcal{T}\}$.

Composition 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 (i) a is enabled in some q_i, and (ii) 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 composite and $i \in \{1, 2\}$, we use $q\lceil\mathcal{P}_i$ to denote q_i. Note that our definition of composition, as well as restriction $\lceil$, can be generalized to any finite number of arguments.

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. In this case, we define their *composition* $\mathcal{T}_1 \| \mathcal{T}_2$ to be the task-PIOA $(\mathcal{P}_1 \| \mathcal{P}_2, R_1 \cup R_2)$. Note that $R_1 \cup R_2$ is an equivalence relation because compatibility requires disjoint sets of locally controlled actions. It is also easy to check that action determinism is preserved under composition.

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 an amorphous set of trace distributions. Rather, it is a mapping that takes every possible "environment" for the given task-PIOA to sets of trace distributions that can arise when the task-PIOA is composed with the given environment. Then our notion of implementation is formulated in terms of inclusion of sets of trace distributions *for each environment automaton*.

Let $\mathcal{T}$ and $\mathcal{E}$ be action-deterministic task-PIOAs. We say that $\mathcal{E}$ is an *environment* for $\mathcal{T}$ if (i) $\mathcal{E}$ is compatible with $\mathcal{T}$ and (ii) the composition $\mathcal{T} \| \mathcal{E}$ is closed. Note that $\mathcal{E}$ may introduce new output actions that are not in the signature of $\mathcal{T}$. 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 a choice of a global task schedule σ. Note that these traces may include new external actions of $\mathcal{E}$ in addition to the external actions already present in $\mathcal{T}$.

Our definition of implementation is influenced by common notions in the security literature [Can01]. Namely, the implementation must "look like" the specification from the perspective of every possible environment. This style of definition enables us to prove simple compositionality results (Theorem 1).

Definition 2. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be* comparable *action-deterministic task-PIOAs, that is, $I_1 = I_2$ and $O_1 = O_2$. 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 E.*

The subscript "0" refers to the fact 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, this notion of implementation can be weakened to allow for negligible discrepancies between matching trace distributions [CCK+06].

Compositionality Here we give a simple compositionality result for our implementation relation. Since we formulate external behavior and implementation in terms of a mapping from environments to sets of trace distributions, this result is quite immediate.

Theorem 1. *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 σ_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 σ_1. It suffices to show that τ is also generated by some task schedule σ_2 for $(\mathcal{T}_2 || \mathcal{T}_3) || \mathcal{T}_4$.

Note that σ_1 is also a task schedule for $\mathcal{T}_1 || (\mathcal{T}_3 || \mathcal{T}_4)$, and that σ_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 σ_2 for $\mathcal{T}_2 || (\mathcal{T}_3 || \mathcal{T}_4)$ such that σ_2 generates the same trace distribution τ in the task-PIOA $\mathcal{T}_2 || (\mathcal{T}_3 || \mathcal{T}_4)$. Since σ_2 is also a task schedule for $(\mathcal{T}_2 || \mathcal{T}_3) || \mathcal{T}_4$ and σ_2 generates τ, this suffices to show that $\mathcal{T}_1 || \mathcal{T}_3 \leq_0 \mathcal{T}_2 || \mathcal{T}_3$. □

3 Simulation Relations

Here, we define a new kind of simulation relation for closed, action-deterministic task-PIOAs, and show that simulation relations of this kind are sound for showing implementation. Our definition is based on three operations involving probability measures: flattening, lifting, and expansion. All of these have been defined elsewhere, for example, in [LSV03].

The first operation, which we call "flattening", takes a discrete probability measure over probability measures and "flattens" it into a single probability measure. Formally, 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 $\mathsf{flatten}(\eta) = \sum_{\mu \in \mathsf{Disc}(X)} \eta(\mu)\mu$.

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. Intuitively, a measure μ_1 on X is related to a measure μ_2 on Y if μ_2 can be obtained by "redistributing" the probabilities masses assigned by μ_1, in such a way that the relation R is respected. Formally, the *lifting* of R, denoted by $\mathcal{L}(R)$, is a 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

1. for each $x \in X$ and $y \in Y$, $w(x, y) > 0$ implies $x \; R \; y$,
2. for each $x \in X$, $\sum_y w(x, y) = \mu_1(x)$, and
3. for each $y \in Y$, $\sum_x w(x, y) = \mu_2(y)$.

The third operation, called "expansion", takes a relation between discrete measures on two domains and returns a relation of the same kind that relates two measures whenever they can be decomposed into two $\mathcal{L}(R)$-related measures. Formally, let R be a relation from $\mathsf{Disc}(X)$ to $\mathsf{Disc}(Y)$. The *expansion* of R, written $\mathcal{E}(R)$, is a relation from $\mathsf{Disc}(X)$ to $\mathsf{Disc}(Y)$ 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 $\mu_1 = \mathsf{flatten}(\eta_1)$, $\mu_2 = \mathsf{flatten}(\eta_2)$, and $\eta_1 \; \mathcal{L}(R) \; \eta_2$.

We use expansions directly in our definition of simulation. Intuitively, $\mu_1\ R\ \mu_2$ means that it is possible to simulate from μ_1 what happens from μ_2. Furthermore, $\mu_1'\ \mathcal{E}(R)\ \mu_2'$ means that we can decompose μ_1' and μ_2' into pieces that can simulate each other, and thus also from μ_2' it is possible to simulate what happens from μ_2'. This intuition is at the base of the proof of our soundness result (cf. Theorem 1).

We need two other auxiliary definitions. The first definition is used to avoid useless proof obligations in our definition of simulation and expresses a consistency condition between a distribution over finite execution fragments and a task schedule. Informally, a distribution ϵ over finite execution fragments is said to be consistent with a task schedule ρ if it assigns non-zero probability only to those executions fragments that are possible under the task schedule ρ.

Definition 3. *Let $\mathcal{T} = (\mathcal{P}, R)$ be a closed, action-deterministic task-PIOA, ϵ be a discrete distribution over finite execution fragments of $\mathcal{P}$, and ρ be a finite task schedule for $\mathcal{T}$. Then we say that ϵ is* consistent with *ρ provided that $\mathsf{supp}(\epsilon) \subseteq \mathsf{supp}(\mathsf{apply}(\mathsf{fstate}(\epsilon), \rho))$, where $\mathsf{fstate}(\epsilon)$ is the image measure of ϵ under fstate.*

For the second definition, suppose we have a mapping c that, given a finite task schedule ρ and a task T of a task-PIOA $\mathcal{T}_1$, yields a task schedule of another task-PIOA $\mathcal{T}_2$. The idea is that $\mathsf{c}(\rho, T)$ describes how $\mathcal{T}_2$ matches task T provided that it has matched already the task schedule ρ. 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 schedule of $\mathcal{T}_2$ that matches all of ρ.

Definition 4. *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 a function that assigns a finite task schedule of $\mathcal{T}_2$ to each finite task schedule of $\mathcal{T}_1$ and task of $\mathcal{T}_1$. Define $\mathsf{full}(\mathsf{c}) : {R_1}^* \to {R_2}^*$ recursively as follows: $\mathsf{full}(\mathsf{c})(\lambda) := \lambda$, and $\mathsf{full}(\mathsf{c})(\rho T) := (\mathsf{full}(\mathsf{c})(\rho))(\mathsf{c}(\rho, T))$.*

We now define our new notion of simulation for task-PIOAs and establish its soundness for the $\leq_0$ relation.

Definition 5. *Let $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ and $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ be two comparable closed action-deterministic task-PIOAs. Let R be a relation from $\mathsf{Disc}(\mathsf{Frags}^*(\mathcal{P}_1))$ to $\mathsf{Disc}(\mathsf{Frags}^*(\mathcal{P}_2))$, satisfying the condition: if $\epsilon_1\ R\ \epsilon_2$ then $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$. We say that R is a simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$ if there exists $\mathsf{c} : ({R_1}^* \times R_1) \to {R_2}^*$, called a* Skolem *function for R, 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 $\epsilon_2' = \mathsf{apply}(\epsilon_2, \mathsf{c}(\rho_1, T))$.*

Theorem 2. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two closed action-deterministic task-PIOAs. If there exists a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$, then $\mathsf{tdists}(\mathcal{T}_1) \subseteq \mathsf{tdists}(\mathcal{T}_2)$.*

As a corollary we derive our main soundness result for task-PIOAs that are not necessarily closed.

Corollary 1. *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 $\mathcal{T}_1 \leq_0 \mathcal{T}_2$.*

4 Application to Security Protocols

In [CCK+06] the task-PIOAs of this paper are used to model and verify, in detail, the Oblivious Transfer (OT) protocol of Goldreich et al. [GMW87]. The analysis is based on the framework of [Can01]. The protocol uses trap-door permutations (and hard-core predicates for them) as the underlying cryptographic primitive. Even though the analyzed protocol and functionality are relatively simple, our exercise is interesting. OT is a powerful primitive that has been shown to be complete for multi-party secure protocols, in the sense that one can construct protocols for securely realizing any functionality using OT as the only cryptographic primitive. It is also interesting because it imposes secrecy requirements when either party is corrupted, in addition to correctness requirements. The protocol uses cryptographic primitives and computational hardness assumptions in an essential way, and the analysis requires general modeling formulations and verification methods that can be used in cryptographic analysis of *any* cryptographic protocol.

At a very high-level the analysis proceeds as follows. We define two PIOAs that represent the "real system", which captures the protocol execution, and the "ideal system", which captures the ideal specification for OT. We show that the real system implements the ideal system with respect to a notion of approximate, time-bounded, implementation for which our simulation relations are sound. The complete proof consists of four cases, depending on the set of parties that are corrupted. When only the transmitter is corrupted, and when both parties are corrupted, it is possible to show that the real system implements the ideal system unconditionally. However, in the other two cases, implementation can be shown only in a "computational" sense, namely, (i) for resource-bounded adversaries, (ii) up to negligible differences, and (iii) under computational hardness assumptions. Modeling these cases requires adding some structure to the basic task-PIOA framework of this paper. Still, the methods of this paper alone suffice for much of the analysis of OT.

We list some interesting technical aspects of the proof. First, the notion of correctness of [Can01] is based on adversaries whose behavior is arbitrary; thus, the proof of correctness should deal with any possible behavior of the adversary. In our framework we view an adversary as a task-PIOA that is composed in parallel with the real and ideal systems, respectively. Simulation relations, and their compositionality properties, work perfectly in this case since the adversary components of the two levels are mapped via the identity function, which is trivially a simulation. Thus, we do not need to explore the structure of the adversary while proving correctness. The adversary may use randomness, which leads us to extend earlier definitions of simulation relations [SL95,LSV03].

In a common pattern that appears in our OT proofs, a lower-level system chooses a random value y and then computes a new value z by applying a trapdoor permutation f to y or by combining y with an input value provided by the environment, e.g., using XOR. At the higher level, a random value is simply chosen, without any use of permutations or input values. However, applying a permutation or a XOR operation to a random value yields the same result as just choosing a random value. These examples motivated our extension of the simulation relation definition to relate distributions to distributions, rather than just states to distributions as in [LSV03].

In our OT models, typical tasks include "choose a random y value", "send the round 1 message", and "deliver the round 1 message", as well as arbitrary tasks of incompletely-specified components such as environment and adversary automata. These tasks do not specify exactly what action occurs; for example, the send task above does not specify the *contents* of the first round message, but just the fact that some round 1 message is being sent. The message contents are selected by the Transmitter process, based on its own internal state. The task mechanism provides a convenient way to specify such choices.

5 Adding Local Schedulers

The action-determinism property may prove to be more restrictive than we would like for applications. To obtain extra expressive power, we add a notion of "local scheduler" for each individual component. A local scheduler of a given component can be used to resolve nondeterministic choices among actions of the same task, using only information about the past computation of that component. In this section, we sketch one way of doing this, with local scheduling based on the current state only, and indicate how our results for the action-deterministic case carry over to this setting. We are currently investigating the use of local schedulers in our case study.

Our notion of local scheduler is simply a "sub-automaton". 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.

A *local scheduler* for a task-PIOA $\mathcal{T}$ is an action-deterministic sub-task-PIOA of $\mathcal{T}$. A *scheduled task-PIOA* is a pair $(\mathcal{T}, \mathcal{T}')$ consisting of a task-PIOA $\mathcal{T}$ and a local scheduler $\mathcal{T}'$ for $\mathcal{T}$. A *probabilistic system* is a pair $\mathcal{M} = (\mathcal{T}, \mathcal{S})$, where $\mathcal{P}$ is a task-PIOA and $\mathcal{S}$ is a set of local schedulers for $\mathcal{P}$. A *probabilistic execution* of a probabilistic system $\mathcal{M} = (\mathcal{T}, \mathcal{S})$ is defined to be any probabilistic execution of any task-PIOA $\mathcal{T}'$, where $\mathcal{T}' \in \mathcal{S}$.

Suppose that $(\mathcal{T}_1, \mathcal{T}_1')$ and $(\mathcal{T}_2, \mathcal{T}_2')$ are two scheduled compatible task-PIOAs. Then define their *composition* $(\mathcal{T}_1, \mathcal{T}_1')||(\mathcal{T}_2, \mathcal{T}_2')$to be the pair $(\mathcal{T}_1||\mathcal{T}_2, \mathcal{T}_1'||\mathcal{T}_2')$. Suppose that $\mathcal{M}_1 = (\mathcal{T}_1, \mathcal{S}_1)$ and $\mathcal{M}_2 = (\mathcal{T}_2, \mathcal{S}_2)$ are probabilistic systems, and $\mathcal{T}_1$ and $\mathcal{T}_2$ are compatible. Then define their *composition* $\mathcal{M}_1||\mathcal{M}_2$ to be the

probabilistic system $(\mathcal{T}_1||\mathcal{T}_2, \mathcal{S})$, where $\mathcal{S}$ is the set of local schedulers $\mathcal{T}'$ for $\mathcal{T}_1||\mathcal{T}_2$ such that $\mathcal{T}' = \mathcal{T}'_1||\mathcal{T}'_2$ for some $\mathcal{T}'_1 \in \mathcal{S}_1$ and $\mathcal{T}'_2 \in \mathcal{S}_2$.

If $\mathcal{M} = (\mathcal{T}, S)$ is a scheduled task-PIOA, then an *environment* for $\mathcal{M}$ is any environment (action-deterministic task-PIOA) for the underlying task-PIOA $\mathcal{P}$. Likewise, if $\mathcal{M} = (\mathcal{T}, \mathcal{S})$ is a probabilistic system, an *environment* for $\mathcal{M}$ is any environment for $\mathcal{T}$.

If $\mathcal{M} = (\mathcal{T}, \mathcal{S})$ is a probabilistic system, then define the *external behavior* of $\mathcal{M}$, written as $\mathsf{extbeh}(\mathcal{M})$, to be the total function that maps each environment task-PIOA $\mathcal{E}$ for $\mathcal{M}$ to the set $\bigcup_{\mathcal{T}' \in \mathcal{S}} \mathsf{tdists}(\mathcal{T}'||\mathcal{E})$. Thus, for each environment, we consider the set of trace distributions that arise from choices of a local scheduler of $\mathcal{M}$ and a global task schedule σ.

Implementation is then defined in the same style as for task-PIOAs.

Definition 6. *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 (i.e., $\mathcal{T}_1$ and $\mathcal{T}_2$ are comparable). Then we say that $\mathcal{M}_1$* 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 get 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 3. *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 that f is a total function 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 $\mathcal{M}_1 \leq_0 \mathcal{M}_2$.*

Simulation Relations It would be nice to get a simulation relation for probabilistic systems whose soundness follows from that for the simulation relation we have already studied, above, for task-PIOAs. Here are some thoughts:

The quantification of the implementation definition, system M1 implementing M2, goes as follows: "Forall E for M1 and M2, forall local schedulers S1 for M1, for all global task schedules sigma1, exists a local scheduler S2 for M2, exists a task schedule sigma2, such that the trace dist of $S1||E$ with sigma1 = trace dist of $S2||E$ with sigma2."

For a simulation relation, we pre-compose with E, which should take care of the universal quantification over E. Basically, after pre-composing, we get new, closed probabilistic systems N1 and N2. The remaining quantification looks like: "Forall local schedulers S1 for N1 for all global task schedules sigma1, exists a local scheduler S2 for N2, exists a task schedule sigma2, such that trace dist of S1 with sigma1 = trace dist of S2 with sigma2."

Now, if we try to use the simulation relation definition we have already, the mappings of steps of a local scheduler S1 would not automatically be consistent with a single local scheduler S2. To get consistency with some S2, all I can think of right now is to instead show: "Forall local schedulers S1 for N1 exists a local scheduler S2 for N2, for all global task schedules sigma1, exists a task schedule sigma2, such that trace dist of S1 with sigma1 = trace dist of S2 with sigma2."

This suggests that, in showing that M1 implements M2, we should hypothesize a function that maps from S1 to S2. That leaves us with two action-deterministic task-PIOAs S1 and S2. Then use our ordinary simulation relation to show that S1 implements S2. That will yield that M1 implements M2. The following theorem expresses this strategy.

Theorem 4. *Suppose* $\mathcal{M}_1 = (\mathcal{T}_1, \mathcal{S}_1)$ *and* $\mathcal{M}_2 = (\mathcal{T}_2, \mathcal{S}_2)$ *are two comparable probabilistic systems. Suppose that* f *is a total function from* $\mathcal{S}_1$ *to* $\mathcal{S}_2$.
Suppose that, for every environment $\mathcal{E}$ *for both* $\mathcal{T}_1$ *and* $\mathcal{T}_2$, *and for every* $S_1 \in \mathcal{S}_1$, *there exists a simulation relation* R *from* $S_1 || \mathcal{E}$ *to* $f(S_1) || \mathcal{E}$.
Then $\mathcal{M}_1 \leq_0 \mathcal{M}_2$.

We also get a compositionality result for local schedulers. The proof is a straightforward generalization of the one for the action-deterministic case.

Theorem 5. *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 σ_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 σ_1 and $\mathcal{P}'_{13}$. It suffices to show that τ is also generated by some task schedule σ_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 σ_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 can be expressed in 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, σ_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 σ_2 for $\mathcal{T}_2 || \mathcal{T}_5$ and a local scheduler $\mathcal{P}'_2$ for $\mathcal{P}_2$ such that σ_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 σ_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 σ_2, $\mathcal{P}'_{23}$ and $\mathcal{P}'_4$ also generate τ, which suffices to show the required implementation relationship. □

6 Conclusions

In order to define partial-information adversaries more systematically, we extend the PIOA framework with a mechanism of task partitions. Basic machineries for verification are provided, including a compositional trace-based semantics and a

sound notion of simulation relation. The utility of these tools is illustrated via the OT case study.

Although our development is largely motivated by concerns in cryptographic analysis, the notion of partial-information scheduling is of independent interest. In particular, partial-information adversaries are widely used to defeat lower bound results proven for perfect-information adversaries [Cha96,Asp03,AB04]. We intend to explore applications in this area, possibly extending the present task-PIOA framework.

Another good direction to pursue is a full treatment of Canetti's Universal Composability results [Can01] in terms of task-PIOAs. This will provide a full-featured modeling framework for security protocols, which is built upon the rigorous computational foundations of [Can01] and at the same time inherits the simplicity and modularity of task-PIOAs. Our OT case study confirms that it is indeed possible to carefully separate high- and low-level nondeterminism. This allows us to use low-level nondeterministic choices as a means of abstraction, as commonly done in the verification of non-cryptographic algorithms.

Finally, we list two technical improvements that are currently under investigation. First, our notion of implementation is currently defined by quantifying over all possible environment automata. We would like to reduce the complexity of this definition by identifying a (minimal) subclass of environments that is sufficient to characterize our implementation relation. Here we may use the idea of principal contexts of [Seg95]. Second, we would like to develop further the notion of local schedulers and evaluate via case studies the tradeoff between expressivity and usability.

References

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

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

[BPW03] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *Proceedings 10th ACM conference on Computer and Communications Security (CCS)*, pages 220–230. ACM, 2003. Extended version at the Cryptology ePrint archive, `http://eprint.iacr.org/2003/015/`.

[BPW04a] M. Backes, B. Pfitzmann, and M. Waidner. A general composition theorem for secure reactive systems. In M. Naor, editor, *First Theory of Cryptography Conference (TCC 2004)*, volume 2951 of *LNCS*, pages 336–354, Cambridge, MA, USA, February 2004. Springer-Verlag.

[BPW04b] M. Backes, B. Pfitzmann, and M. Waidner. Secure asynchronous reactive systems. Cryptology ePrint Archive Report 2004/082, 2004.

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

[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, MIT CSAIL, 2005.

[CCK+06] 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. Submitted to CSFW, 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 *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, pages 166–175, 1996.

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

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

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

[KLA98] 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 polytime 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 *Proceedings of the 13th Annual ACM Symposium on the Principles of Distributed Computing*, pages 314–323, 1994.

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

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

[MMS03] P. Mateus, J.C. Mitchell, and A. Scedrov. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In *Proceedings CONCUR 2003*, pages 323–345, 2003.

[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 *Proceedings CCS 2000*, 2000.

[PW01] B. Pfitzman and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the 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.

[RMST04] A. Ramanathan, J.C. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *Proceedings FOSSACS 2004*, 2004.

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

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

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

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

A Mathematical Foundations

A.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.

If X is any set, then 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).

A.2 Probability measures

We present the basic definitions that we need for probability measures. For completeness, we recall from Section 3 the definitions of flattening, lifting, and expansion, and we prove several related lemmas.

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 $\mu : \mathcal{F} \to [0, \infty]$ 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 denote 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; $\mathsf{supp}(\mu)$ 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 $\mu_1 \times \mu_2(x, y) = \mu_1(x) \times \mu_2(y)$ 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 $C \in \mathcal{F}_X$, $\rho(C) = \sum_{i \in I} p_i \rho_i(C)$.

A function $f : X \to Y$ is said to be measurable from $(X, \mathcal{F}_X) \to (Y, \mathcal{F}_Y)$ if the inverse image of each element of $\mathcal{F}_Y$ is an element of $\mathcal{F}_X$, that is, for each $C \in \mathcal{F}_Y$, $f^{-1}(C) \in \mathcal{F}_X$. In such a case, given 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.

Flattening The first operation, which we call "flattening", takes a discrete probability measure over probability measures and "flattens" it into a single probability measure.

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 1. *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. By the definition of flattening, $f(\mathsf{flatten}(\eta)) = f(\sum_{\mu \in \mathsf{Disc}(X)} \eta(\mu)\mu)$. By distributing f, we obtain that this is equal to $\sum_{\mu \in \mathsf{Disc}(X)} \eta(\mu) f(\mu)$. By rearranging terms in this last expression, we obtain that

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

Now, $\sum_{\mu \in f^{-1}(\sigma)} \eta(\mu) = f(\eta)(\sigma)$, which implies that

$$f(\mathsf{flatten}(\eta)) = \sum_{\sigma \in \mathsf{Disc}(Y)} f(\eta)(\sigma)\sigma.$$

But the right-hand expression is the definition of $\mathsf{flatten}(f(\eta))$, as needed. □

Lemma 2. *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 $\mathsf{flatten}(\sum_{i \in I} p_i \eta_i) = \sum_{i \in I} p_i \mathsf{flatten}(\eta_i)$.*

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. Intuitively, a measure μ_1 on X is related to a measure μ_2 on Y if μ_2 can be obtained by "redistributing" the probabilities masses assigned by μ_1, in such a way that the relation R is respected.

Formally, the *lifting* of R, denoted by $\mathcal{L}(R)$, is a 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

1. for each $x \in X$ and $y \in Y$, $w(x, y) > 0$ implies $x \; R \; y$,
2. for each $x \in X$, $\sum_y w(x, y) = \mu_1(x)$, and
3. for each $y \in Y$, $\sum_x w(x, y) = \mu_2(y)$.

Expansion Finally, we have the third operation, called "expansion", which we use directly in our new definition of simulation relations. Let R be a relation from $\mathsf{Disc}(X)$ to $\mathsf{Disc}(Y)$. The *expansion* of R, denoted by $\mathcal{E}(R)$, is again 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

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

Intuitively, 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 very useful characterization of the expansion relation.

Lemma 3. *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 $\{\mu_{1,i}\}_I$,$\{\mu_{2,i}\}_I$ such that*

1. $\mu_1 = \sum_{i \in I} p(i)\mu_{1,i}$,
2. $\mu_2 = \sum_{i \in I} p(i)\mu_{2,i}$, *and*
3. *for each $i \in I$, $\mu_{1,i} \; R \; \mu_{2,i}$.*

Proof. Let $\mu_1 \; \mathcal{E}(R) \; \mu_2$, and let η_1, η_2 and w be the measures and weighting functions 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 $\mathcal{E}(R)$. □

B Probabilistic I/O Automata

In Section 2, we saw basic definitions of the PIOA framework. Here we provide more details.

B.1 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 7. *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 8. *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 $\mathsf{Frags}^*(\mathcal{P})$. 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 9. *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 10. *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 11. *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 12. *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 13. *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 14. *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 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 4. *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 supremums), 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.

B.2 Probabilistic Executions and Trace Distributions

Having established some groundwork in Section B.1, we now turn to our behavioral semantics based on probability measures on executions and traces. Recall from Section 2 that nondeterministic choices in a PIOA is resolved by means of a scheduler. The following lemmas give some simple equations expressing basic relationships involving the probabilities of various sets of execution fragments.

Lemma 5. *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 $\alpha \leq \alpha'$, the equation above 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 6. *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 5 after observing that $\epsilon_{\sigma,\alpha}(C_\alpha) = 1$. □

Lemma 7. *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 5, by definition of $\epsilon_{\sigma,\alpha'}(C_\alpha)$, and by definition of $\mu_{\sigma(\tilde{\alpha})}(a,q)$, $\epsilon_{\sigma,\mu}(C_\alpha) = \mu(C_\alpha) + \sum_{\alpha'<\alpha} \mu(\alpha')\epsilon_{\sigma,\alpha'}(C_{\tilde{\alpha}})\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q)$. Observe that the factor $\sigma(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q)$ is a constant with respect to α', and thus can be moved out of the sum, so

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

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 6 (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 $\sigma(\alpha)(\perp)$ to denote the probability of terminating after α, namely, $1 - \sigma(\alpha)(D)$.

Lemma 8. *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 6. □

Lemma 9. *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 $C_{\alpha a} = \cup_q C_{\alpha a q}$. Thus, $\epsilon_{\sigma,\mu}(C_{\alpha a}) = \sum_q \epsilon_{\sigma,\mu}(C_{\alpha a q})$. By Lemma 7, the right-hand side is equal to

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

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

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

Combining the equations yields the result. □

Next, we consider limits of generalized probabilistic execution fragments.

Proposition 2. *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 9, if a is enabled in $\mathsf{lstate}(\alpha)$ then $p^i_\alpha \sigma_i(\alpha)(\mathsf{tran}_{\alpha,a}) = p^i_{\alpha a}$, and so, 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, also $\lim_{i\to\infty} \sum_a \sigma_i(\alpha)(\mathsf{tran}_{\alpha a}) \leq 1$. By interchanging the limit and the sum, we obtain $\sum_a \lim_{i\to\infty} \sigma_i(\alpha)(\mathsf{tran}_{\alpha a}) \leq 1$.

We claim that $\sigma(\alpha)(\mathsf{tran}_{\alpha,a}) \leq \lim_{i\to\infty} \sigma_i(\alpha)(\mathsf{tran}_{\alpha,a})$, which immediately 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, $\sigma(\alpha)(\mathsf{tran}_{\alpha,a}) = \lim_{i\to\infty} \sigma_i(\alpha)(\mathsf{tran}_{\alpha,a})$, which implies the claim. On the other hand, if $p_\alpha = 0$, then $\sigma(\alpha)(\mathsf{tran}_{\alpha,a})$ is defined to be zero, so that $\sigma(\alpha)(\mathsf{tran}_{\alpha,a}) = 0$, which is less than or equal to $\lim_{i\to\infty} \sigma_i(\alpha)(\mathsf{tran}_{\alpha,a})$, which again implies the claim.

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 5, $\epsilon_{\sigma,\mu}(C_q) = \mu(C_q)$, 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 $\alpha = \tilde{\alpha}aq$. By Lemma 7,

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

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 7, we obtain that

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

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 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 hypothesis. □

B.3 Hiding

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

Definition 15. *Let $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ be a PIOA and let $S \subseteq O$ be given. 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 $H_{\mathcal{P}} = H_{\mathcal{P}} \cup S$.*

B.4 Proof of Proposition 1

The proof of Proposition 1 uses a series of auxiliary lemmas.

Lemma 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 $\mathcal{P}$ 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,

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

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 αaq is an execution fragment, we claim that $p_1(\alpha aq) = 0$. 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 $(\mathsf{lstate}(\alpha), a, \rho) \in \mathcal{D}_{\mathcal{P}}$, 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 $\mu(\alpha) = \sum_{(a,q)} p_1(\alpha aq)$. By action determinism, only one action $b \in T$ is enabled from $\mathsf{lstate}(\alpha)$. By definition of p_1, $p_1(\alpha aq) = 0$ if $a \neq b$ (either $a \notin T$ or a is not enabled from $\mathsf{lstate}(\alpha)$). Thus,

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

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

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

Lemma 11. *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. The key part of the inductive step consists of the claim 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 1. 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 1,

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

Rearranging terms, we obtain

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

By Lemma 10, the right side becomes $\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon(\alpha)$, which equals 1 by the inductive hypothesis. Therefore $\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = 1$, as needed. □

Lemma 12. *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 10, $\mu(C_\alpha) = \sum_{\alpha'|\alpha \leq \alpha'}(p_2(\alpha') + \sum_{(a,q):\alpha' aq \in \mathsf{Frags}^*(\mathcal{P})} p_1(\alpha' aq)) = \sum_{\alpha'|\alpha \leq \alpha'}(p_1(\alpha') + p_2(\alpha')) - p_1(\alpha)$. 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 10, 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 13. *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let μ be a discrete measure over finite execution fragments, T a task, and $\mu' = \mathsf{apply}(\mu, T)$. Then $\mu \leq \mu'$.*

Proof. Follows directly by Lemma 12. □

Lemma 14. *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 13 for the inductive step. □

Lemma 15. *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-determministic task-PIOA. Let μ be a discrete 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. □

Lemma 16. *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 1, $\mathsf{apply}(\mu, \rho) = \mathsf{apply}(\epsilon, T)$. 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 11, 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 1. 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 10, the right side of the equation above is $\sum_{\alpha:q\leq\alpha}\epsilon(\alpha)$, which is precisely $\epsilon(C_q)$. □

Lemma 17. *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 $\mathsf{apply}(\epsilon, T)$. Let σ' be a new scheduler such that, for each finite execution fragment α,

- 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 $\mathsf{act}(\mathsf{tran}) \in T$,

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

  - otherwise,

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

Here $D(\mathsf{lstate}(\alpha))$ denotes the set of transitions of D with source state $\mathsf{lstate}(\alpha)$ and $\mathsf{act}(\mathsf{tran})$ denotes the action that occurs in tran. We first prove that σ', thus defined, is a scheduler. 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 5, $\epsilon_{\sigma,\mu}(C_q) = \mu(C_q)$ and $\epsilon_{\sigma',\mu}(C_q) = \mu(C_q)$. Thus, $\epsilon_{\sigma',\mu}(C_q) = \epsilon_{\sigma,\mu}(C_q)$. By definition, the right-hand-side is equal to $\epsilon(C_q)$, which is equal to $\epsilon'(C_q)$ by Lemma 16. Thus, $\epsilon_{\sigma',\mu}(C_q) = \epsilon'(C_q)$, as needed.

For the inductive step, let $\alpha = \tilde{\alpha}aq$. By Lemma 5 and the definition of the measure of a cone (Equation (1)), we get

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

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

$$\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha) + \sum_{\alpha'\leq\tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}})\sigma'(\tilde{\alpha})(\mathsf{tran}_{\tilde{\alpha},a})\mu_{\tilde{\alpha},a}(q). \qquad (2)$$

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 7, $\epsilon_{\sigma',\mu}(C_\alpha) = \mu(C_\alpha)$. It is therefore sufficient to show that $\epsilon'(C_\alpha) = \mu(C_\alpha)$.
   By Lemma 13, $\epsilon(C_{\tilde{\alpha}}) \leq \epsilon'(C_{\tilde{\alpha}})$. 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 6 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 7 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 13 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 12 (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 (2) 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 6 and algebraic manipulation, $\sum_{\alpha' \leq \tilde{\alpha}} \mu(\alpha')\epsilon_{\sigma',\alpha'}(C_{\tilde{\alpha}}) = \epsilon_{\sigma',\mu}(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\})$. By 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 get

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

   By definition, $\epsilon = \epsilon_{\sigma,\mu}$. Therefore, by Lemma 7, the right side of the equation above is $\epsilon(C_\alpha)$. Moreover, $\epsilon(C_\alpha) = \epsilon'(C_\alpha)$ by Lemma 12, 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

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

   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,

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

   By definition, $\epsilon = \epsilon_{\sigma,\mu}$. Using Lemma 8, we may substitute $\epsilon(\tilde{\alpha})$ for $(\epsilon(C_{\tilde{\alpha}}) - \mu(C_{\tilde{\alpha}} - \{\tilde{\alpha}\}))\sigma(\tilde{\alpha})(\bot)$. 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 12, Item (3).

□

Lemma 18. *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 15 for the base case and Lemma 17 for the inductive step. □

Lemma 19. *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 18 and 14, the sequence $\epsilon_0, \epsilon_1, \ldots$ is a chain of generalized probabilistic execution fragments generated by μ. By Proposition 2, $\lim_{i \to \infty} \epsilon_i$ 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. □

Now we can prove Proposition 1, our main target. It says that, for any μ and ρ, the probability measure on execution fragments generated by $\mathsf{apply}(\mu, \rho)$ is "standard", in that it can be obtained from μ and a scheduler as defined in Section 2 for basic PIOAs.

Proof of Proposition 1. Follows directly by Lemmas 18 and 19. □

C Soundness of Simulation

We begin with some auxiliary results.

Lemma 20. *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 schedulers and let μ be a discrete probability measure on finite execution fragments. For each $i > 0$, 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 ρ be the concatenation of all the ρ_i's, and 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 13. 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 sub-sequence of the sequence used in the definition of $\mathsf{apply}(\mu, \rho_1 \rho_2 \ldots)$, therefore they have the same limit. □

Lemma 21. *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable closed action-deterministic task-PIOAs, R a simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Let ϵ_1 and ϵ_2 be discrete distributions on finite execution fragments of $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, such that $\epsilon_1 \; \mathcal{E}(R) \; \epsilon_2$. Then $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$.*

The next proposition states that $\mathsf{apply}(\cdot, \rho)$ distributes over convex combinations of probability measures. We start with a few preliminary lemmas.

Lemma 22. *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 it follows that $\mathsf{apply}(\sum_i p_i\mu_i, T)(\alpha) = \sum_i p_i \mathsf{apply}(\mu_i, T)(\alpha)$ since $\mathsf{apply}(\sum_i p_i\mu_i, T)(\alpha)$ is defined to be $p_1(\alpha)+p_2(\alpha)$, and $\sum_i p_i \mathsf{apply}(\mu_i, T)(\alpha) = \sum_i p_i(p_1^i(\alpha) + p_2^i(\alpha)) = \sum_i p_i p_1^i(\alpha) + \sum_i p_i p_2^i(\alpha) = p_1(\alpha) + p_2(\alpha)$.

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 1, $p_1(\alpha) = (\sum_i p_i\mu_i)(\alpha')\rho(q)$, 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 1, $p_1(\alpha) = 0$, 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 1, $p_2(\alpha) = (\sum_i p_i\mu_i)(\alpha)$, 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 1, $p_2(\alpha) = 0$, and, for each i, $p_2^i(\alpha) = 0$. Thus, $p_2(\alpha) = \sum_i p_i p_2^i(\alpha)$ trivially. □

Proposition 3. *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 1, $\mathsf{apply}(\sum_i p_i\mu_i, \rho' T) = \mathsf{apply}(\mathsf{apply}(\sum_i p_i\mu_i, \rho'), T)$. By induction, $\mathsf{apply}(\sum_i p_i\mu_i, \rho') = \sum_i p_i \mathsf{apply}(\mu_i, \rho')$. Thus, we obtain $\mathsf{apply}(\sum_i p_i\mu_i, \rho' T) = \mathsf{apply}(\sum_i p_i \mathsf{apply}(\mu_i, \rho'), T)$. By Lemma 11, for each i, $\mathsf{apply}(\mu_i, \rho')$ is a discrete probability measure over finite execution fragments. By Lemma 22, $\mathsf{apply}(\sum_i p_i \mathsf{apply}(\mu_i, \rho'), T) = \sum_i p_i \mathsf{apply}(\mathsf{apply}(\mu_i, \rho'), T)$, and by Definition 1, 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. □

Lemma 23. *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, that distribute 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 $\sum_i p_i = 1$, $f(\sum_i p_i\rho_i) = \sum_i p_i f(\rho_i)$, and similarly, for each countable family $\{\rho_i\}_i$ of discrete measures on Y and each countable family of probabilities $\{p_i\}_i$ such that $\sum_i p_i = 1$, $g(\sum_i p_i\rho_i) = \sum_i p_i g(\rho_i)$. Let μ_1 and μ_2 be two measures on X and Y respectively, such that $\mu_1\ \mathcal{E}(R)\ \mu_2$, and let η_1, η_2, and w be a pair of measures and a weighting function witnessing that $\mu_1\ \mathcal{E}(R)\ \mu_2$. Suppose further that, for*

any two distributions $\rho_1 \in \mathsf{supp}(\eta_1)$ *and* $\rho_2 \in \mathsf{supp}(\eta_2)$ *such that* $w(\rho_1, \rho_2) > 0$, $f(\rho_1)\ \mathcal{E}(R)\ g(\rho_2)$.
Then $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$.

Proof. For each $\rho_1 \in \mathsf{supp}(\eta_1)$ and $\rho_2 \in \mathsf{supp}(\eta_2)$ such that $w(\rho_1, \rho_2) > 0$, let $(\eta_1)_{\rho_1,\rho_2}$, $(\eta_2)_{\rho_1,\rho_2}$, and $w_{\rho_1\rho_2}$ be a pair of measures and a weighting function that prove that $f(\rho_1)\ \mathcal{E}(R)\ g(\rho_2)$. We know that these are well-defined since, by assumption, $f(\rho_1)\ \mathcal{E}(R)\ g(\rho_2)$ whenever $w(\rho_1, \rho_2) > 0$. Let W denote the set of pairs (ρ_1, ρ_2) such that $w(\rho_1, \rho_2) > 0$.

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' prove that $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$.

1. $f(\mu_1) = \mathsf{flatten}(\eta_1')$.
   By 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, 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 definition 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 definition of w', $\sum_{\rho_2'\in\mathsf{Disc}(Y)} w'(\rho_1', \rho_2') = \sum_{\rho_2'\in\mathsf{Disc}(Y)} \sum_{(\rho_1,\rho_2)} w(\rho_1, \rho_2) w_{\rho_1,\rho_2}(\rho_1', \rho_2')$. By rearranging sums and using the fact that w_{ρ_1,ρ_2} is a weighting function, we obtain that $\sum_{\rho_2'\in\mathsf{Disc}(Y)} w'(\rho_1', \rho_2') = \sum_{(\rho_1,\rho_2)} w(\rho_1, \rho_2)(\eta_1)_{\rho_1,\rho_2}(\rho_1')$. (Specifically, this uses the fact that $\sum_{\rho_2'\in\mathsf{Disc}(Y)} w_{\rho_1,\rho_2}(\rho_1', \rho_2') = (\eta_1)_{\rho_1,\rho_2}(\rho_1')$.) This suffices since the right-hand side is the definition of $\eta_1'(\rho_1')$.
   (c) Symmetric to the previous case.

□

Lemma 24. *Let* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *be two comparable closed task-PIOAs, let* R *be a simulation relation from* $\mathcal{T}_1$ *to* $\mathcal{T}_2$*, and let* c *be a mapping that satisfies the conditions required for a simulation relation.*
Let ρ_1 *and* ρ_2 *be finite task schedulers of* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *respectively, such that*

$\rho_2 = \mathsf{full}(\mathsf{c})(\rho_1)$. *Let* $\epsilon_1 = \mathsf{apply}(\delta(\bar{q}_1), \rho_1)$ *and* $\epsilon_2 = \mathsf{apply}(\delta(\bar{q}_2), \rho_2)$ *be the respective discrete distributions on finite executions of* $\mathcal{T}_1$ *and* $\mathcal{T}_2$ *generated by* ρ_1 *and* ρ_2. *Suppose that* $\epsilon_1 \; \mathcal{E}(R) \; \epsilon_2$.

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 23: Define the function f on discrete distributions on finite execution fragments of $\mathcal{T}_1$ by $f(\epsilon) = \mathsf{apply}(\epsilon, T)$, and the function g on discrete distributions on finite execution fragments of $\mathcal{T}_2$ by $g(\epsilon) = \mathsf{apply}(\epsilon, \mathsf{c}(\rho_1, T))$. We show that the hypothesis of Lemma 23 is satisfied, which implies that, by Lemma 23, $\epsilon_1' \; \mathcal{E}(R) \; \epsilon_2'$, as needed.

Distributivity of f and g follows directly by Proposition 3. 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. □

Proof of Theorem 2. 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 $\rho_1 \rho_2 \cdots$. We claim that $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$, which suffices.

For each $j \geq 0$, let $\epsilon_{1,j} = \mathsf{apply}(\bar{q}_1, T_1 \cdots T_j)$, and $\epsilon_{2,j} = \mathsf{apply}(\bar{q}_2, \rho_1 \cdots \rho_j)$. By Lemma 20, for each $j \geq 0$, $\epsilon_{1,j} \leq \epsilon_{1,j+1}$ and $\epsilon_{2,j} \leq \epsilon_{2,j+1}$, and furthermore, $\lim_{j \to \infty} \epsilon_{1,j} = \epsilon_1$ and $\lim_{j \to \infty} \epsilon_{2,j} = \epsilon_2$. Also, note that 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 $\epsilon_{2,0} = \delta(\bar{q}_2)$. By the start condition for a simulation relation and a trivial expansion, we see that $\epsilon_{1,0} \; \mathcal{E}(R) \; \epsilon_{2,0}$. Then by induction, using Lemma 24 for the inductive step, for each $j \geq 0$, $\epsilon_{1,j} \; \mathcal{E}(R) \; \epsilon_{2,j}$. Then, by Lemma 21, for each $j \geq 0$, $\mathsf{tdist}(\epsilon_{1,j}) = \mathsf{tdist}(\epsilon_{2,j})$. By Lemma 4, $\mathsf{tdist}(\epsilon_1) = \lim_{j \to \infty} \mathsf{tdist}(\epsilon_{1,j})$, 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 $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$, as needed. □

The lemma below captures a special case of the simulation relation definition we have given above. Any relation that satisfies the hypotheses of this lemma is a simulation relation. We use this special case in proving the correctness of the OT protocol.

Lemma 25. *Let* $\mathcal{T}_1 = (\mathcal{P}_1, R_1)$ *and* $\mathcal{T}_2 = (\mathcal{P}_2, R_2)$ *be two comparable closed action-deterministic task-PIOAs. Let* R *be a relation from discrete distributions over finite execution fragments of* $\mathcal{P}_1$ *to discrete distributions over finite executions fragments of* $\mathcal{P}_2$, *satisfying: If* $\epsilon_1 \; R \; \epsilon_2$ *then* $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$. *Let* $c : ({R_1}^* \times R_1) \to {R_2}^*$. *Suppose further that the following conditions 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 there exist*
   - *a probability measure* p *on a countable index set* I,
   - *probability measures* ϵ'_{1j}, $j \in I$, *on finite execution fragments of* $\mathcal{P}_1$, *and*
   - *probability measures* ϵ'_{2j}, $j \in I$, *on finite execution fragments of* $\mathcal{P}_2$,

   such that:
   - *for each* $j \in I$, $\epsilon'_{1j}\ R\ \epsilon'_{2j}$,
   - $\sum_{j\in I} p(j)(\epsilon'_{1j}) = \mathsf{apply}(\epsilon_1, T)$, *and*
   - $\sum_{j\in I} p(j)(\epsilon'_{2j}) = \mathsf{apply}(\epsilon_2, c(\rho_1, T))$.

Then R *is a simulation relation from* $\mathcal{T}_1$ *to* $\mathcal{T}_2$ *using* c.

Proof. By a straightforward application of Lemma 3. See[CCK+05] for details. □

D Early/Late/Toss Examples

We use the following example to illustrate features of our task-PIOA framework.

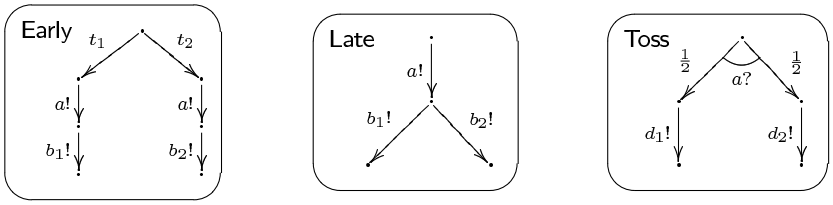

Fig. 1. Probabilistic automata Early, Late and Toss

Example 1. Task-PIOAs: Early, Late and Toss.
Automaton Early first performs either internal action t_1 or t_2, then output action a, and finally, either output action b or output action c, depending on whether it first performed t_1 or t_2.

Early:
Signature:

Input:
none
Output:
a, b_1, b_2
Internal:
t_1, t_2

Tasks:
$T_1 = \{t_1\}$, $T_2 = \{t_2\}$, $A = \{a\}$, $B = \{b_1, b_2\}$
States:
counter $\in \{0, 1, 2, 3\}$, initially 0.
chosen $\in \{b_1, b_2, \perp\}$, initially $\perp$

Transitions:

$t1$
Precondition:
counter $= 0$
Effect:
counter $:= 1$; chosen $:= b_1$

$t2$
Precondition:
counter $= 0$
Effect:
counter $:= 1$; chosen $:= b_2$

a
Precondition:
counter $= 1$
Effect:
counter $:= 2$

b_1
Precondition:
counter $= 2$; chosen $= b_1$
Effect:
counter $:= 3$

c
Precondition:
counter $= 2$; chosen $= b_2$
Effect:
counter $:= 3$

Automaton Late first performs a, and then performs either b_1 or b_2, nondeterministically.

Late:
Signature:

Input:
none
Output:
a, b_1, b_2
Internal:
none

Tasks:
$A = \{a\}$, $B_1 = \{b_1\}$, $B_2 = \{b_2\}$
States:
counter $\in \{1, 2, 3\}$, initially 1

Transitions:

a
Precondition:
    counter $= 1$
Effect:
    counter $:= 2$

b_1
Precondition:
    counter $= 2$
Effect:
    counter $:= 3$

b_2
Precondition:
    counter $= 2$
Effect:
    counter $:= 3$

Finally, on input a, automaton Toss chooses d_1 or d_2 randomly, which enables output either d_1 or d_2.

Toss:
Signature:

Input:
    a
Output:
    d_1, d_2
Internal:
    none

Tasks:
$D = \{d_1, d_2\}$
States:
counter $\in \{1, 2, 3\}$, initially 1
chosen $\in \{d_1, d_2, \perp\}$, initially $\perp$

Transitions:

a
Effect:
    if counter $= 1$ then {
        counter $:= 2$;
        chosen $:=$ random(unif$\{d_1, d_2\}$)
        }

d_1
Precondition:
    counter $= 2$; chosen $= d_1$
Effect:
    counter $:= 3$

d_2
Precondition:
    counter $= 2$; chosen $= d_2$
Effect:
    counter $:= 3$

Example 2. **Trace distributions of Early and Late.**

Early has task schedules that are arbitrary finite or infinite sequences of the tasks T_1, T_2, A, and B. We provide a list of "minimal" sequences, in which each task is enabled in some state that can result after the previous schedule. For each, we give the resulting trace distribution. (Other sequences give rise to the same trace distribution as a reduced version that appears in the following list.)

Task schedule λ (the empty sequence of tasks) gives rise to the trace distribution that is the Dirac measure for λ (the empty sequence of actions). Likewise, task schedules T_1 and T_2 give rise to the Dirac measure for λ. Task schedule T_1, A and task schedule T_2, A both give rise to the Dirac measure for the sequence (a). Task schedule T_1, A, B gives rise to the Dirac measure for the sequence (a, b_1). Finally, task schedule T_2, A, B gives rise to the Dirac measure for the sequence (a, b_2). Thus, tdists($Early$) consists of four Dirac measures, on the traces λ, a, (a, b_1) and (a, b_2), respectively.

Similarly for Late. Task schedule λ gives rise to the Dirac measure for λ; task schedule A gives rise to the Dirac measure for (a); and A, B_1 and A, B_2 give rise to the Dirac measures for (a, b_1) and (a, b_2), respectively. Thus, tdists(Late) consists of the same four Dirac measures as tdists(Early).

Example 3. **Composition: Early, Late, and Toss:** $Early$ and $Toss$ are compatible task-PIOAs, so we may compose them. The tasks of $Early||Toss$ are then $T1 = \{t1\}$, $T2 = \{t2\}$, $A = \{a\}$, $BC = \{b, c\}$, and $DE = \{d, e\}$. The trace distributions of $Early||Toss$ include the four trace distributions generated by $Early$ alone. In addition, five new trace distributions appear: First, we have the distribution described by $\{1/2 : a, d; 1/2 : a, e\}$, which represents the situation when $Toss$ completes its steps but $Early$ does not. (Here the notation refers to the probabilities with which the two traces appear.) Likewise, we have the trace distributions $\{1/2 : a, b, d; 1/2 : a, b, e\}$, $\{1/2 : a, c, d; 1/2 : a, c, e\}$, $\{1/2 : a, d, b; 1/2 : a, e, b\}$, and $\{1/2 : a, d, c; 1/2 : a, e, c\}$, all of which represent possible trace distributions for situations in which both components complete their steps.

Similarly we may compose $Late$ and $Toss$, with resulting tasks A, B, C, and DE. The trace distributions of $Late||Toss$ are the same as for $Early||Toss$.

Note that none of the trace distributions of $Early||Toss$ and $Late||Toss$ describe correlations between the choice of b vs. c and the choice of d vs. e: in each distribution, the choice between b or c is determined, and each of d and e is equally likely. In contrast, in previous work based on more powerful schedulers, the scheduler can create correlations between the choices of b vs. c and the choice of d vs. e. Such correlations were used to demonstrate that compositionality failed to hold in these models.

Example 4. **External behavior: Early and Late:** We describe the external behavior of $Early$ and $Late$; more specifically, we describe how these two automata behave when composed with three particular environments.

The first environment, $\mathcal{E}_1$, is the trivial environment consisting of an automaton with a single state and no transitions. Then tdists($Early||\mathcal{E}_1$) is simply

the set $\mathsf{tdists}(Early)$, and $\mathsf{tdists}(Late||\mathcal{E}_1) = \mathsf{tdists}(Late)$. Since $\mathsf{tdists}(Early) = \mathsf{tdists}(Late)$, we see that $Early$ and $Late$ exhibit the same trace distributions in environment $\mathcal{E}_1$.

The second environment, $\mathcal{E}_2$, is a special "reporting environment" which has inputs a, b, and c, and outputs $report(\beta)$, where β is a finite sequence of elements of $\{a, b, c\}$. All the $report$ actions constitute a single task. This automaton simply records the sequence of inputs it has seen so far. From any state, it is enabled to perform $report(\beta)$, where β is the recorded sequence. Then $\mathsf{tdists}(Early||\mathcal{E}_2)$ is the set of trace distributions that can be obtained from trace distributions in $\mathsf{tdists}(Early)$ by inserting appropriate $report$ actions uniformly in all of the traces. Again, $\mathsf{tdists}(Late||\mathcal{E}_2)$ is the same set of trace distributions.

The third environment, $\mathcal{E}_3$, is the task-PIOA $Toss$. Now $\mathsf{tdists}(Early||\mathcal{E}_3)$ is described in Example 3. Again, $\mathsf{tdists}(Late||\mathcal{E}_3)$ is the same set of trace distributions. We claim that, in fact, $Early$ and $Late$ have the same trace distributions for any environment $\mathcal{E}$.

Example 5. **Implementation: Early and Late:** We claimed in Example 4 that $Early$ and $Late$ have the same trace distributions for any environment $\mathcal{E}$; that is, for any environment $\mathcal{E}$, $\mathsf{tdists}(Early||\mathcal{E}) = \mathsf{tdists}(Late||\mathcal{E})$. This implies that $Early \leq_0 Late$ and $Late \leq_0 Early$

The following slightly intricate example illustrates why it is important to include the environments in the definition of implementation. Namely, it presents two closed action-deterministic task-PIOAs $\mathcal{T}_1$ and $\mathcal{T}_2$ such that $\mathsf{tdists}(\mathcal{T}_1) \subseteq \mathsf{tdists}(\mathcal{T}_2)$, but such that some environment can distinguish them.

Example 6. **Insufficiency of trace distribution inclusion:** Let $\mathcal{T}_1$ have output actions a, b, and c, and internal action tc. Tasks are $A = \{a\}$, $B = \{b\}$, and $C = \{c, tc\}$. Transitions are as described by the following diagram, where the a actions are accompanied with random choices with equal probability.

The trace distributions of $\mathcal{T}_1$ are $\{1 : \lambda\}$, $\{1 : a\}$, $\{1/2 : a; 1/2 : ac\}$ (produced, for example, by schedule A, C), and $\{1/2 : ab; 1/2 : ac\}$ (produced, for example, by schedule A, C, B).

Let $\mathcal{T}_2$ have output actions a, b, and c, and internal actions $t1$, $t2$, and tb. Tasks are $T1 = \{t1\}$, $T2 = \{t2\}$, $A = \{a\}$, $B = \{b, tb\}$, and $C = \{c\}$. Transitions are described by the following diagram. Then every trace distribution of $\mathcal{T}_1$ is produced by some task schedule of $\mathcal{T}_2$: $\{1 : \lambda\}$ by λ, $\{1 : a\}$ by T_1, A, $\{1/2 : a; 1/2 : ac\}$ by T_2, A, C, and $\{1/2 : ab; 1/2 : ac\}$ by T_1, A, B, C.

Now consider the reporting environment $\mathcal{E}_2$ defined in Example 4. Let R denote the report task. Task schedule A, R, C, R, B, R for $\mathcal{T}_1||\mathcal{E}_2$ then produces the trace distribution $\{1/2 : areport(a)report(a)breport(ab); 1/2 : areport(a)creport(ac)report(ac)\}$.

We claim that no task schedule of $\mathcal{T}_2||\mathcal{E}_2$ yields this trace distribution. We can argue this by contradiction. Suppose such a schedule σ exists. Then it must contain either T_1 or T_2. If T_2 occurs first, then there is no way for the output action b to be generated, so it must be that T_1 comes first. Then A must appear before the first R, in order to generate the first $report(a)$ action. That leads to a random choice between the two branches leading to tb, c and to just b. Then

before the second R, we must have a B followed by a C, in order to generate the $report(ac)$ that appears on one of the branches. But then on the other branch, a b output would be generated before the second report, leading to a report that contains the action b. Since no such report appears on the other branch, we again obtain a contradiction.

Example 7. **Simulation relation: Early and Late:** It is easy to show that $\mathsf{tdists}(Early) \subseteq \mathsf{tdists}(Late)$, using a simulation relation. Namely, if ϵ_1 and ϵ_2 are discrete distributions over finite execution fragments of *Early* and *Late*, respectively, then we define $(\epsilon_1, \epsilon_2) \in R$ provided that the following condition holds:
For every $s \in \mathsf{lstate}(\epsilon_1)$ and $u \in \mathsf{lstate}(\epsilon_2)$, if $s.\mathsf{counter} = 0$ then $u.\mathsf{counter} = 1$, and otherwise $s.\mathsf{counter} = u.\mathsf{counter}$. This is a simulation relation from *Early* to *Late* using the task correspondence mapping c defined by: $c(\sigma, T_1) = c(\sigma, T_2) = \lambda$, $c(\sigma, A) = A$, $c(\sigma, BC) = B, C$.

This idea can be extended to show that $Early \leq_0 Late$, again using simulation relations. Namely, fix any environment $\mathcal{E}$ for *Early* and *Late*. Now if ϵ_1 and ϵ_2 are discrete distributions over finite execution fragments of $Early||\mathcal{E}$ and $Late||\mathcal{E}$, respectively, then we define $(\epsilon_1, \epsilon_2) \in R$ provided that, for every $s \in \mathsf{lstate}(\epsilon_1)$ and $u \in \mathsf{lstate}(\epsilon_2)$, the following hold:

1. If $s.\mathsf{counter} = 0$ then $u.\mathsf{counter} = 1$, and otherwise $s.\mathsf{counter} = u.\mathsf{counter}$.
2. $s.\mathcal{E} = u.\mathcal{E}$.[1]

The task correspondence mapping is the same as before, with the additional clause: $c(\sigma, T) = T$ for every task T of $\mathcal{E}$.

Although we also know that $Late \leq_0 Early$, this cannot be shown using a simulation relation of the kind we have defined. In fact, the following example shows that we cannot even show that $\mathsf{tdists}(Late) \subseteq \mathsf{tdists}(Early)$ in this way. It remains to develop more general notions of simulation relation that are complete for showing $\leq_0$.

Example 8. **No simulation relation: Late and Early:** We show that no simulation relation exists from *Late* to *Early*. Suppose for contradiction that a simulation relation R does exist, using task correspondence c. Then if σ_1 is any task schedule of *Late*, it follows that $\mathsf{tdist}(\sigma_1) = \mathsf{tdist}(c * (\sigma_1))$.

Now let σ_1 be the task schedule A, B of *Late*. Then $\mathsf{tdist}(\sigma_1)$ is the Dirac distribution for the sequence ab. Then $c * (ab)$ must also give the same trace distribution in *Early*. So $c * (ab)$ must be some variant of T_1, A, BC. (The only variation allowed here is to insert other tasks at points where they are not enabled.) Similarly, $c * (a)$ must be a variant of T_1, A or T_2, A; however, since $c*$ is monotonic, we must have $c * (a)$ a variant of T_1, A.

Symmetrically, $c*(ac)$ must be some variant of T_2, A, BC; then monotonicity implies that $c * (a)$ is a variant of T_2, A. This yields a contradiction.

[1] This notation refers to the entire state of $\mathcal{E}$.

Example 9. **Compositionality: Early, Late, and Toss:** We claimed, in Example 5 that $Early \leq_0 Late$ and $Late \leq_0 Early$. Theorem 1 then implies that $Early||Toss \leq_0 Late||Toss$.

E OT Example

The following example is an abstract version of one that arises in the Oblivious Transfer proof. This motivated our definition of simulation relations in terms of probability distributions on execution fragments.

Example 10. **Simulation relation: Trapdoor and Random** We consider two task-PIOAs, $Trapdoor$ and $Random$. $Random$ 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 results. More precisely:

Random:
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	$report(k)$
Precondition:	Precondition:
$zval = \perp$	$zval = k$
Effect:	Effect:
$zval :=$ random(uniform($\{1, \ldots, n\}$))	none

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:
$yval \in \{1, \ldots, n\} \cup \{\perp\}$, initially $\perp$
$zval \in \{1, \ldots, n\} \cup \{\perp\}$, initially $\perp$

Transitions:

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

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

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

We can show that $\mathsf{tdists}(Trapdoor) \subseteq \mathsf{tdists}(Random)$, again using a simulation relation. In defining the correspondence, it seems most natural to allow the steps that define $zval$ to correspond in the two automata. That means that the step that defines $yval$ in $Trapdoor$ should map to an empty sequence of steps in $Random$. However, after $yval$ has been defined in $Trapdoor$, we have a randomly-chosen value recorded in state of $Trapdoor$, whereas no corresponding value is recorded in the corresponding state of $Random$. Thus, we are led to correspond the entire distribution on execution fragments of $Trapdoor$ to a single state or $Random$. Our simulation relation notion allows us to do this.

Specifically, if ϵ_1 and ϵ_2 are discrete distributions over finite execution fragments of $Trapdoor$ and $Random$, respectively, then we define $(\epsilon_1, \epsilon_2) \in R$ provided that the following conditions hold:

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

The task correspondence mapping c is defined by: $c(\sigma, Choose) = \lambda$, $c(\sigma, Compute) = Choose$, $c(\sigma, Report) = Report$.

F Comparison of task-PIOAs with existing frameworks

In this section we compare task-PIOAs with existing frameworks for modeling cryptographic protocols. In particular, we discuss the *Secure Asynchronous Reactive Systems* of [PW01,BPW04b].

To appear: discussion of the *Probabilistic Polynomial-Time Process Calculus* of [LMMS98,MMS03,RMST04].

F.1 Secure Asynchronous Reactive Systems

We first discuss the relations between task-PIOAs and the version of PIOA introduced by Backes, Pfitzmann and Waidner [PW01,BPW03,BPW04b,BPW04a].

System Types The reactive systems in [PW01,BPW04b] are given by collections of *machines*. Each machine M is specified (in part) by a transition function of the following form:

$$\Delta : S \times I \to \mathsf{Disc}(S \times O).$$

Here S is the state space and I and O denote the set of input and output signals, respectively. Each signal is a tuple of finite strings over a fixed finite alphabet Σ. Every tuple in I has a fixed arity, which is determined by the number of input ports associated with M. Similarly for O.

The transition function of a PIOA defined here has a very different form (cf. Section 2):

$$\Delta : S \times (I \cup O \cup H) \to (\{\bot\} + \mathsf{Disc}(S)).$$

If $\Delta(s, a) = \bot$, then a is not enabled in s; otherwise, $\Delta(s, a)$ specifies a probability distribution on the resulting state after the execution of a.

We highlight three differences.

- Machines in [PW01,BPW04b] are of a decidedly *functional* character: given a tuple of input strings, some randomized computation is performed, producing a tuple of output strings. In contrast, the representation of a PIOA is entirely *operational*: the actions in $I \cup O \cup H$ are abstractions of activities of a system, rather than values manipulated by the system. Thus, in a PIOA, inputs and outputs need not correspond as tightly as they do in machines.
- Machines in [PW01,BPW04b] do not have internal/hidden transitions. This is because internal computations are abstracted away (provided the computation does not exceed certain limits on time and space).
- The only form of nondeterminism in a machine resides in the choices between inputs. In other words, nondeterminism can only be used to model uncertainties in the external environment. PIOAs, on the other hand, allow nondeterministic choices *within* a system (e.g., between different output actions enabled in the same state). Therefore, a closed system of machines is completely specified (up to coin tosses), whereas a closed system of PIOAs may contain many nondeterministic choices.

Communication In [PW01,BPW04b], machines are divided into three classes: *simple machines*, *master schedulers* and *buffers*. These machines communicate with each other via *ports*, which are classified as: *simple*, *buffer* and *clock*.

Each buffer B specifies a high-level connection between a *unique* pair of non-buffer machines. Messages placed in B are delivered only when B is scheduled via its clock port. This scheduling is designated to a unique non-buffer machine (one that "owns" the clock port of B), which also determines the order in which messages in B are delivered.

Notice, both low-level (based on port names) and high-level (based on buffers) connections are "handshakes". That is, at most two machines are involved in a synchronization (barring the machine responsible for scheduling the buffer).

In the case of PIOAs, communications may be "broadcasts". Each action a is "owned" by a unique PIOA P (i.e., a is an output action of P), but any number of other PIOAs can have a as an input action. A hiding operation is available, which reclassifies certain output actions as hidden actions, thus preventing synchronization with PIOAs outside the scope of hiding.

Also, actions in our setting need not represent messages. They can be used to model synchronized steps of different processes.

Channels In [PW01,BPW04b], three kinds of communication channels are considered: secure, authenticated and insecure.

By default, a communication channel between two machines is secure: no third party can access message contents. Another party might still be able to schedule the buffer for this channel. Authenticated channels are modeled by creating a copy of the out-port corresponding to the authenticated channel, and by connecting this new port to the adversary. Insecure channels are directly connected to the adversary.

In the case of PIOAs, the basic communication channels correspond to authenticated channels: every output action may synchronize with several input actions, including adversarial ones. As for machines, insecure channels are modeled by routing messages through the adversary. We use two techniques to define secure channels: the first option is to hide output actions in the composition of the PIOAs corresponding to the sending and receiving ends of the channels, and the second is to specify constraints on the signature of specific (adversarial) PIOAs to prevent them from synchronizing with the corresponding output actions.

In our analysis of the OT protocol, we consider a slightly different kind of communication channels: all protocol messages are routed through the adversary, but we define the adversary in such a way that it can only send messages it received before. In the [PW01,BPW04b] communication model, this would correspond to an authenticated channel with buffer scheduled by the adversary.

Composition In [PW01,BPW04b], *collections* of machines are defined as sets of machines, with the intuition that machines in a collection will interact. These

interactions can be the transmission of a message (through a buffer) or the scheduling of buffers.

In the case of PIOAs, we can compose several PIOAs, yielding a new PIOA. This enables, for instance, defining the role of a protocol participant through several simple PIOAs, each of these PIOAs describing a different aspect of this protocol role. Thus, decomposition of PIOAs can be used to analyze systems in a more modular way.

Scheduling In [PW01,BPW04b], scheduling is *distributed*: it is performed collectively by all machines via scheduling of buffers. The following algorithm is used.

(1) The current active machine M reads all of its inputs and carries out changes dictated by its transition function.
(2) All of the outputs produced in step (1) are copied to corresponding buffers.
(3) If in step (1) M schedules at least one buffer, then let B be the first such buffer and M' be the receiving machine of B. (This is possible because an ordering of ports is given in the specification of M.) The message scheduled by M is delivered to M' and M' is the next active machine. If no such message exists, then the unique master scheduler is the next active machine.
(4) If in step (1) M does not schedule a buffer, then the unique master scheduler is the next active machine.

Under this algorithm, every closed collection of machines induces a unique probability space on the set of runs. In essence, there is no "real" scheduling to be done once a collection is closed, because all scheduling decisions are already specified by machines in the collection.

In the present paper, scheduling for task-PIOAs is *centralized*: it is determined by a global task sequence. Each task determines a unique component and at most one transition in that component. If such a transition is not enabled, then no changes take place.

In our setting, we distinguish between two kinds of nondeterministic choices:

1. High-level choices, such as timing of messages. These are resolved algorithmically by the adversary, which acts as the network.
2. Low-level choices representing inessential ordering of events. In the underlying semantics, these are resolved by a global task sequence.

However, since we have defined a sound notion of simulation relation, these choices can remain unresolved throughout our proofs. That is, it is sufficient to define a simulation relation between two systems, rather than first resolving all nondeterministic choices in one and then trying to mimic the same behavior in the other.

Thus, PIOAs are (purposely) under-specified, so that inessential ordering of events can be abstracted away from our proofs. We believe this is a major difference between [PW01,BPW04b] and our work.

Security Properties In [PW01,BPW04b], several versions of reactive simulatability are defined. All of them compare views of a user for complete, closed systems, that is, systems with purely probabilistic behavior.

More precisely, a structure $(\hat{M}_1, S)$ (that is, a set of simple machines with specified service ports available for the user) is said to be (at least) as secure as another structure $(\hat{M}_2, S)$, if and only if: for every adversary A_1 for $(\hat{M}_1, S)$, there is an adversary A_2 for $(\hat{M}_2, S)$ such that the views of any user H in the configurations $(\hat{M}_1, S, H, A_1)$ and $(\hat{M}_2, S, H, A_2)$ cannot be distinguished. (The quantifiers we use here are those of the *universal simulatability* notion, which is closest to the one we use.) Different indistinguishability notions can be used here, requiring the views to be equal for perfect security, or computationally indistinguishable for computational security for instance.

Even though the security properties we prove in our task-PIOA model are very similar to those described in [PW01,BPW04b], an important difference lies in the fact that our definitions involve comparing task-PIOAs before resolving the non-determinism. As a result, our implementation definitions quantify over all possible task schedules.

A second difference is that, in our computational definition of implementation, we do not compare the views of the user, but the probabilities that the environment (which is the task-PIOA corresponding to the user) outputs a specific *accept* flag. This choice, which sticks more closely to the formulation in the UC framework [Can01], simplified the statement of some properties of our computational implementation notion (see the composition property for instance).

Complexity In [PW01,BPW04b], a computational realization in terms of interactive probabilistic Turing machines is proposed for the machines presented. A machine is said to be polynomial-time iff it has a realization as a polynomial-time probabilistic Turing machine. Other complexity measures are defined, always using the notion of computational realization.

Our notion of time-bounded task-PIOAs uses a bit-strings encoding of actions, tasks and states, and requires bounds on different probabilistic Turing machines used to decode these actions. It also requires bounds on the time needed for determining the enabled action in a given task, and for computing the next state from the current state and an action.

Machines have a security parameter k as input, and their computational realization provides a single (uniform) Turing machine used for every value of the security parameter. We conjecture that this notion is equivalent to our notion of uniformly polynomial-time-bounded task-PIOA family.

Our notion of (non-uniform) polynomial-time-bounded task-PIOA family, which is the one we use in practice, is more closely related to the notion of parameterized systems defined in [BPW04a]. The correspondence is not immediate however: systems are sets of structures, which are pairs of machines and service ports.

Proof techniques In [BPW03], simulatability is proved by using bisimulation with error sets. More precisely, a mapping between states of the two considered systems is defined, and one show that the same input in corresponding states leads to the same output and corresponding states again, except in some specific error cases. Then, it is proved that the probability of these error cases to occur is negligible via reductions on attacks against the underlying cryptographic primitives.

We establish our security proofs in a different way. We use two implementation relations. The first one, $\leq_0$, guarantees that every trace distribution of the first related task-PIOA is also a trace distribution of the second related task-PIOA (without any error probability). We prove this by establishing simulation relations, then by using a soundness result stating that the existence of a simulation relation between two systems guarantees that they are $\leq_0$-related. Rather than relating states, our simulation relation relates probability distributions of execution fragments. Also, we do not prove that probability distributions of execution fragments are related before and after executing any input action, but that, starting from related probability distributions of execution fragments, we reach a countable number of related probability distributions of execution fragments when we execute any task on the first system and a corresponding sequence of tasks in the second system. This type of relation allows us to relate systems with random choices performed at different times, for example. It also allows us to manage the quantifier over all possible task-schedulers in a natural way.

This technique in only used for relating systems in the absence of any computational assumption. Besides this, we define computational assumptions in terms of our approximate implementation relation $\leq_{neg,pt}$ (and prove the equivalence between the standard, computational, formulation of these assumptions and our PIOA version of it). Then, using the composition properties of the $\leq_{neg,pt}$ relation, we prove that the larger task-PIOAs corresponding to the whole protocol model are also $\leq_{neg,pt}$-related. This step does not involve any argument "by contradiction".

F.2 Probabilistic Polynomial-Time Process Calculus

This section is still under construction. It should be available shortly.