Computer Science and Artificial Intelligence Laboratory

Technical Report

MIT-CSAIL-TR-2013-006 April 11, 2013

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
IBM T.J. Watson Research Center

Ling Cheung
MIT and Radboud University of Nijmegen

Dilsun Kaynar
Carnegie Mellon University
dilsunk@cmu.edu

Moses Liskov
College of William and Mary

Nancy Lynch
MIT

Olivier Pereira
Université catholique de Louvain

Roberto Segala
University of Verona

May 28, 2009

Abstract

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

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

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

1 Introduction

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

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

In order to formulate and to prove probabilistic properties of distributed algorithms, one needs mechanisms for resolving nondeterministic choices. The most common mechanism is a *perfect-information* event scheduler, which has access to local state and history of all system components and has unlimited computational power. Thus, probabilistic properties of distributed algorithms are typically asserted with respect to worst-case, adversarial schedulers who can choose the next event based on complete knowledge of the past (e.g., [PSL00, Seg95, SL95, BK98]).

One would expect that a similar modeling paradigm, including both probabilistic and nondeterministic choices, would be similarly useful for modeling *cryptographic protocols*, which are special kinds of distributed algorithms that use cryptographic primitives and that guarantee properties such as secrecy and authentication. However, the traditional probabilistic and nondeterministic modeling paradigm does not readily apply to cryptographic protocols. One major reason is that the perfect-information scheduler mechanism used for distributed algorithms is too powerful for this setting. In the presence of nondeterminism (or entropy) the schedulers that are used to resolve nondeterminism may be exploited to create additional channels of informational flow. A scheduler that has perfect information of the history would be able to access sensitive information in the states of the non-corrupted protocol participants such as their random choices, and be able to "divulge" that information to adversarial entities by encoding it in the ordering of events.

Most existing works on cryptographic protocol verification address this issue by requiring that all entities are fully specified up to inputs and coin tosses. This solution does not scale well to large protocols that handle implementation issues explicitly, because a full specification would involve many details that are irrelevant in the security analysis. In this paper, we take a different approach: instead of restricting the presence of nondeterminism, we try to find less powerful mechanisms for resolving nondeterminism. The result is an adaptation of PIOAs called *task-PIOAs*. In this new framework, cryptographic protocols may be specified with nondeterminism, yet without the danger of introducing semantic inconsistencies such as hidden information flow. Using this flexibility, one can simplify the description of practical protocols by leaving inessential choices unspecified. Furthermore, with the help of proof techniques such as probabilistic simulation, these inessential choices may remain unspecified throughout the entire correctness analysis.

Task-PIOAs A task-PIOA is simply a PIOA augmented with a partition of non-input actions into equivalence classes called *tasks*.[1] A task is typically a set of related actions that perform the "same kind of activity" in a protocol. For example, in a protocol with several rounds of message exchange, all the actions that send a round 1 message (with possibly different message contents) would constitute a task. Similarly, if a protocol involves a step in which a random choice is made from a particular domain, we could group all the actions that make a random choice from that domain (yielding possibly different values) into a task.

Tasks are units of scheduling, as for I/O automata; they are scheduled by oblivious, global *task schedule*

[1] The terminology of "tasks" and "task partition" traces back to the original I/O Automata framework of Lynch and Tuttle [LT89].

sequences. We think of a task schedule as simply a way of representing the order in which different system activities happen to occur. This order can be determined, for example, by variations in speeds of different system components, or by unpredictable network delays, rather than by a purposeful scheduler entity. This simple, non-adaptive task schedule mechanism eliminates the problem of creating undesirable channels of information flow through schedulers. At first sight, it may seem that our task schedules are insufficient to describe adversarial scheduling patterns of the kind that occur in security protocols. However, we model such patterns in a different way, which we explain in detail later in the paper.

For task-PIOAs, we define notions of *external behavior* and *implementation*, by adapting the trace distribution semantics of Segala [Seg95] to task-based scheduling. We define parallel composition in the usual way and show that our implementation relation is compositional. We also define a new type of *simulation relation*, which incorporates the notion of tasks, and show that it is sound for proving implementation relations between task-PIOAs. This new definition differs from simulation relations studied earlier [SL95, LSV03], in that it relates probability measures rather than states.

In many cases, including our work on cryptographic protocols (see below), tasks alone suffice for resolving nondeterminism. However, for extra expressive power, we define a second mechanism called *local schedulers*, which uses local information only to resolve nondeterminism within system components. This mechanism is based on earlier work in [CLSV06].

Task-PIOAs are clearly suitable for modeling and analyzing cryptographic protocols; they may also be useful for other kinds of distributed algorithms in which the perfect information assumption is unrealistically strong.

Adversarial Scheduling The standard scheduling mechanism in the cryptographic community is an *adversarial scheduler*, namely, a resource-bounded algorithmic entity that determines the next move adaptively, based on its own view of the computation so far. Clearly, this is weaker than the perfect-information scheduler used for distributed algorithms. It is however stronger than our notion of global task schedule sequences, which are essentially *oblivious schedulers* that fix the entire schedule nondeterministically in advance.

In order to capture the adaptivity of adversarial schedulers within our framework, we separate scheduling concerns into two parts.

1. The adaptive adversarial scheduler is modeled as a system component represented as a task-PIOA, for example, a message delivery service that can eavesdrop on the communications and control the order of message delivery. Such a system component has access to partial information about the execution: it sees information that other components communicate to it during execution, but not "secret information" that is found only in the internal state of these components.

2. Low-level scheduling choices are resolved by a task schedule sequence that is chosen nondeterministically in advance. For example, in a typical protocol, many different parties make independent random choices, and it is inconsequential which of them does so first. Each of these coin tosses would correspond to a single task, which does not contain any information about the actual outcome of the coin toss. A task schedule then fixes a particular order in which the different coin tosses occur.

In short, the high-level adversarial scheduler is responsible for choices that are essential in security analysis, while the low-level schedule of tasks resolves inessential choices. We believe this separation is conceptually meaningful and we illustrate it with a small example in Section 3, where an adaptive adversary uses eavesdropped information to gain advantage. This confirms that adaptive adversarial scheduling can indeed be captured in the task-PIOA framework.

Cryptographic Protocol Analysis In [CCK+05b, CCK+06c], we apply the task-PIOA framework to analyze the Oblivious Transfer (OT) protocol of Goldreich et al. [GMW87]. This framework can be decomposed into two "layers": (i) a general foundational layer, not specific to security protocols and (ii) a security layer that follows the general outline of simulation-based security [GMR85, GMW87, GL90, Bea91, MR91,

PW94, Can95]. Task-PIOAs serve as the basis of the foundational layer. To express computational limitations, we augment task-PIOAs with additional structures, such as probabilistic Turing machines responsible for computing a next state from any given source state and action. This leads to the notions of *time-bounded task-PIOAs* and *approximate implementation* with respect to time-bounded environments. These are used, for example, to express computational indistinguishability assumptions for cryptographic primitives. Detailed definitions involving time bounds are beyond the scope of this paper. However, for those readers interested in our modeling, we provide a summary of the OT case study in Section 5. An abstract of our general approach to cryptographic protocol modeling can be found in [CLK$^+$06].

We observe that, although our correctness proofs for OT are somewhat lengthy, most of the complexity lies in the establishment of simulation relations between specifications at various levels of abstraction. This is consistent with the fact that we use nondeterminism to model implementation freedom (e.g., whether to toss a coin before or after an expected input arrives). Our correctness claims are guaranteed to hold, no matter how an implementer chooses to resolve such choices. Therefore, our modeling is more general than typical models in the literature, where low-level nondeterministic choices are fixed and hard-coded into specifications (e.g. the OT model of [MMS03]).

The correctness theorem that we used in our analysis of OT, stated in terms of our approximate implementation relation, follows the idea of simulation-based security. For that analysis, we use a composition theorem that applies to a constant number of substitutions. In [CCK$^+$07] we generalize this composition theorem to any polynomial number (in the security parameter) of substitutions.[2] This result complements the fundamental theory of task-PIOAs presented in this paper in that it allows modular (and hence scalable) analysis of cryptographic protocols.

Aside from the OT case study, we have also proposed a novel approach for verifying statistical zero-knowledge (SZK) properties [CMP07]. This approach uses recently developed techniques based on approximate simulation relations [ML07]. Specifically, statistical indistinguishability is formulated as an implementation relation in the Task-PIOA framework, which can then be proven using approximate simulation relations. This technique separates proof obligations into two categories: those requiring probabilistic reasoning, as well as those that do not. The latter is a good candidate for mechanization. We illustrate the general method by verifying the SZK property of the well-known identification protocol proposed by Girault, Poupard and Stern [GPS06].

The Task-PIOA framework was also used by Nagao et al. [NMO08] in order to compare three cryptographic channels. While these channels were proven to be equivalent in a purely probabilistic communication model widely used in cryptography, the analysis of Nagao et al. [NMO08] shows that this equivalence only holds under a very specific subclass of schedulers as soon as nondeterminism is considered.

1.1 Road Map

Section 2 presents mathematical preliminaries, as well as basic definitions and results for PIOAs. Section 3 defines task-PIOAs, task schedules, composition, and implementation, and presents a compositionality result for implementation. Section 4 presents our simulation relation definition and the associated soundness theorem, as well as an example that illustrates the use of simulation relations. Section 5 summarizes our OT protocol case study. Section 6 discusses local schedulers and Section 7 treats related work in greater detail. Concluding remarks follow in Section 8.

2 Preliminaries

2.1 Notation for Sets and Sequences

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

[2]The polynomial composition is proven for a slightly stronger variant of our approximate implementation relation $\leq_{neg,pt}$.

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

If $\gamma \in X^*$ and $\gamma' \in X^* \cup X^\omega$, then we write $\gamma \frown \gamma'$ for the concatenation of the sequences γ and γ'. Sometimes, when no confusion seems likely, we omit the $\frown$ symbol, writing just $\gamma\gamma'$. We also write $\gamma_1\gamma_2\ldots\gamma_i$ for the concatenation of i sequences and we extend the notation to infinite concatenations.

2.2 Probability Measures

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. For a set $C \subseteq 2^X$, we call the σ-field *generated* by C the smallest σ-field $\mathcal{F} \subseteq 2^X$ that includes C, that is, such that $C \subseteq \mathcal{F}$. 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 refer to the measure of a singleton set.

A *support* of a sub-probability measure μ on $(X, \mathcal{F})$ is a measurable set C such that $\mu(X - C) = 0$. If μ is a discrete sub-probability measure, then we denote by $\mathsf{supp}(\mu)$ the set of elements that have non-zero measure (thus $\mathsf{supp}(\mu)$ is a support of μ). We let $\delta(x)$ denote the *Dirac measure* for x, the probability measure that assigns probability 1 to $\{x\}$.

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

Measurable Functions 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 $C \in \mathcal{F}_Y$, $f^{-1}(C) \in \mathcal{F}_X$ where $f^{-1}(C)$ is the set of elements x such that $f(x) \in C$. Note that, if $\mathcal{F}_X$ is 2^X, then any function $f : X \to Y$ is measurable from $(X, \mathcal{F}_X) \to (Y, \mathcal{F}_Y)$ for any $\mathcal{F}_Y$.

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

Combination of Measures, Flattening If $\{\mu_i\}_{i\in I}$ 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 $\sum_{i\in I} p_i\mu_i$ denotes a measure μ on $(X, \mathcal{F}_X)$ such that, for each $C \in \mathcal{F}_X$, $\mu(C) = \sum_{i\in I} p_i \cdot \mu_i(C)$. In particular, if all the μ_i's are sub-probability measures and $\sum_{i\in I} p_i \leq 1$, then $\sum_{i\in I} p_i\mu_i$ is a sub-probability measure; furthermore, if all the μ_i's are probability measures and $\sum_{i\in I} p_i = 1$, then $\sum_{i\in I} p_i\mu_i$ is a probability measure. We state and prove an elementary property of measurable functions and summation.

Proposition 2.1 *Let $\{\mu_i\}_{i\in I}$ be a family of sub-probability measures on $(X, \mathcal{F}_X)$, and let $\{p_i\}_{i\in I}$ be a family of non-negative values. Let f be a measurable function from $(X, \mathcal{F}_X)$ to $(Y, \mathcal{F}_Y)$. Then $f(\sum_{i\in I} p_i\mu_i) = \sum_{i\in I} p_i f(\mu_i)$.*

Proof. Let C be an element of $\mathcal{F}_Y$. Then

$$f(\sum_{i\in I} p_i\mu_i)(C) = (\sum_{i\in I} p_i\mu_i)(f^{-1}(C)) = \sum_{i\in I} p_i\mu_i(f^{-1}(C)) = \sum_{i\in I} p_i f(\mu_i)(C),$$

where the first and last steps follow by definition of image measure, and the second step follows by definition of $\sum_{i\in I} p_i\mu_i$. □

Sometimes, rather than denoting explicitly all μ_i's and p_i's, we consider a discrete probability measure η on the set of measures on $(X, \mathcal{F}_X)$, and define a new measure $\mathsf{flatten}(\eta)$, called the *flattening* of η, as

$$\mathsf{flatten}(\eta) := \sum_{\mu\in\mathsf{supp}(\eta)} \eta(\mu)\mu.$$

We also say that η is a *decomposition* of $\mathsf{flatten}(\eta)$, in the sense that $\mathsf{flatten}(\eta)$ can be expressed as the convex combination of the measures in $\mathsf{supp}(\eta)$. Observe that, since the η-measure of any element outside $\mathsf{supp}(\eta)$ is 0, we can take the sum over arbitrarily larger sets. Thus, in the specific case where $\mathcal{F}_X = 2^X$, we can compute safely $\mathsf{flatten}(\eta)$ as $\sum_{\mu\in\mathsf{Disc}(X)} \eta(\mu)\mu$. When we need to reason with flattened measures, we choose the expression that appears to be most useful for.

A useful property of the flattening operator is that it commutes with image measures. We state and prove the result explicitly for discrete probability measures.

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

Proof. By definition of flattening and Proposition 2.1,

$$f(\mathsf{flatten}(\eta)) = f\left(\sum_{\mu\in\mathsf{supp}(\eta)} \eta(\mu)\mu\right) = \sum_{\mu\in\mathsf{supp}(\eta)} \eta(\mu)f(\mu).$$

Let $\mu \in \mathsf{supp}(\eta)$. Then $\mu \in f^{-1}(f(\mu))$ and $f(\mu) \in \mathsf{supp}(f(\eta))$. Thus, $\mathsf{supp}(\eta) \subseteq \cup_{\rho\in\mathsf{supp}(f(\eta))} f^{-1}(\rho)$, where the $f^{-1}(\rho)$ sets are pairwise disjoint since f is a function. Thus,

$$\sum_{\mu\in\mathsf{supp}(\eta)} \eta(\mu)f(\mu) = \sum_{\rho\in\mathsf{supp}(f(\eta))} \sum_{\mu\in f^{-1}(\rho)\cap\mathsf{supp}(\eta)} \eta(\mu)f(\mu).$$

Observe that $f(\mu) = \rho$ in the right-hand expression above. Furthermore, since $\eta(\mu) = 0$ outside $\mathsf{supp}(\eta)$, the condition on $\mathsf{supp}(\eta)$ can be removed from the inner sum. Thus, the right-hand expression can be rewritten into $\sum_{\rho\in\mathsf{supp}(f(\eta))} \sum_{\mu\in f^{-1}(\rho)} \eta(\mu)\rho$. Since $\sum_{\mu\in f^{-1}(\rho)} \eta(\mu) = \eta(f^{-1}(\rho)) = f(\eta)(\rho)$, we obtain

$$f(\mathsf{flatten}(\eta)) = \sum_{\rho\in\mathsf{supp}(f(\eta))} f(\eta)(\rho)\rho = \mathsf{flatten}(f(\eta)),$$

where the last equality follows by definition of flattening. □

We also state and prove distributivity of flattening over summation.

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

Proof. Let η denote $\sum_{i\in I} p_i\eta_i$. By definition of flattening and definition of summation, $\mathsf{flatten}(\eta) = \sum_{\mu\in\mathsf{supp}(\eta)} \eta(\mu)\mu = \sum_{\mu\in\mathsf{supp}(\eta)} \sum_{i\in I} p_i\eta_i(\mu)\mu$. By exchanging sums and rearranging terms, $\mathsf{flatten}(\eta) = \sum_{i\in I} p_i \sum_{\mu\in\mathsf{supp}(\eta)} \eta_i(\mu)\mu$. Observe that, for each i, if $p_i = 0$ then the inner sum has no influence on the result, and if $p_i > 0$, then $\mathsf{supp}(\eta_i) \subseteq \mathsf{supp}(\eta)$. Thus, $\mathsf{flatten}(\eta) = \sum_{i\in I} p_i \sum_{\mu\in\mathsf{supp}(\eta_i)} \eta_i(\mu)\mu$. Finally, since by definition of flattening the inner sum is $\mathsf{flatten}(\eta_i)$, we obtain $\mathsf{flatten}(\eta) = \sum_{i\in I\in I} p_i\mathsf{flatten}(\eta_i)$ as needed. □

2.3 Relations on Probability Measures

We define here two operators on relations that deal with probabilities. The first operator is taken from [Seg95] and is a generalization of a similar operator of [JL91], though it has strong connections with work of Kantorovitch [Kan58]. It lifts one relation on sets to a relation on probability measures. The second operator is a closure operator for relations on probability measures. The reasons for the need of these operators will become clear later in the paper. Yet, we can give a few hints here. A probabilistic automaton is like an ordinary automaton, except that transitions lead to probability measures on states rather than to single states. Probabilistic automata are compared by defining relations on states that are "preserved" by transitions. However, given two related states, the outcomes of transitions are probability measures. Thus we need some ways to lift a relation on states to a relation on probability measures. The operators are presented here because they can be of independent interest; however, the reader may as well skim through this section and get back to it when the operators are used for the first time.

Lifting Given a relation R between two domains X_1 and X_2, the question is whether it is possible to define a meaningful *lifting* of R on $\mathsf{Disc}(X_1)$ and $\mathsf{Disc}(X_2)$. The approach we follow here is to state that a measure μ_1 on X_1 is related to a measure μ_2 on X_2 if μ_2 can be obtained by "redistributing" the probability masses assigned by μ_1 in an R-respecting way.

Definition 2.4 *The* lifting *of a relation R from a set X_1 to a set X_2, denoted by $\mathcal{L}(R)$, is the relation from $\mathsf{Disc}(X_1)$ to $\mathsf{Disc}(X_2)$ defined by: $\mu_1\ \mathcal{L}(R)\ \mu_2$ iff there exists a function $w : X \times Y \to \mathbf{R}^{\geq 0}$ such that the following hold:*

1. *For each $x_1 \in X_1$ and $x_2 \in X_2$, $w(x_1, x_2) > 0$ implies $x_1\ R\ x_2$.*
2. *For each $x_1 \in X_1$, $\sum_{x_2 \in X_2} w(x_1, x_2) = \mu_1(x_1)$.*
3. *For each $x_2 \in X_2$, $\sum_{x_1 \in X_1} w(x_1, x_2) = \mu_2(x_2)$.*

We call w a weighting function *for $\mu_1\ \mathcal{L}(R)\ \mu_2$, or alternatively, a weighting function for $\mathcal{L}(R)$.*

The weight $w(x_1, x_2)$ represents the probability (mass) that is transferred from x_1 to x_2. Condition 1 states that probability is transferred only between R-related states; Condition 2 states that the total probability transferred from from x_1 is $\mu_1(x_1)$; Condition 3 states that the total probability transferred to x_2 is $\mu_2(x_2)$. The weighting function can also be seen as a joint probability measure on $X_1 \times X_2$ that is supported on R and whose marginal measures are μ_1 and μ_2, respectively. If R is an equivalence relation, then we can represent it as a metric on the disjoint union of X_1 and X_2 where two related elements are at distance 0 and two unrelated elements are at distance ∞. In such case, the lifting operator is exactly the metric (relation) defined by Kantorovitch in [Kan58].

Closure under Convex Combination If we are given a relation from $\mathsf{Disc}(X_1)$ to $\mathsf{Disc}(X_2)$, we can use the lifting operator of the previous paragraph to obtain a relation from $\mathsf{Disc}(\mathsf{Disc}(X_1))$ to $\mathsf{Disc}(\mathsf{Disc}(X_2))$. Given two measures η_1, η_2 such that $\eta_1\ \mathcal{L}(R)\ \eta_2$, it is not necessarily the case that $\mathsf{flatten}(\eta_1)\ R\ \mathsf{flatten}(\eta_2)$, though in several cases we are interested in relations that are closed under such operation.

We define here a closure operator, called *expansion*, that given a relation R from $\mathsf{Disc}(X_1)$ to $\mathsf{Disc}(X_2)$ returns the smallest relation that includes R and is closed under the construction above: two measures are related whenever they can be decomposed into two $\mathcal{L}(R)$-related measures.

Definition 2.5 *Let R be a relation from $\mathsf{Disc}(X_1)$ to $\mathsf{Disc}(X_2)$. The* expansion *of R, denoted by $\mathcal{E}(R)$, is a relation from $\mathsf{Disc}(X_1)$ to $\mathsf{Disc}(X_2)$ defined as follows: $\mu_1\ \mathcal{E}(R)\ \mu_2$ iff there exist two discrete measures η_1 and η_2 on $\mathsf{Disc}(X_1)$ and $\mathsf{Disc}(X_2)$, respectively, such that the following hold:*

1. *$\mu_1 = \mathsf{flatten}(\eta_1)$.*

2. $\mu_2 = \mathsf{flatten}(\eta_2)$.

3. $\eta_1 \; \mathcal{L}(R) \; \eta_2$.

We say that η_1 and η_2 are witnesses *for $\mu_1 \; \mathcal{E}(R) \; \mu_2$ or for $\mathcal{E}(R)$. If w is a weighting function for $\eta_1 \; \mathcal{L}(R) \; \eta_2$, then we say as well that η_1, η_2, and w are witnesses for $\mu_1 \; \mathcal{E}(R) \; \mu_2$ or for $\mathcal{E}(R)$.*

In other words, we enlarge R by adding pairs of measures that can be "decomposed" into weighted sums of measures so that the weights can be "redistributed" in an R-respecting manner. Taking this intuition one step further, the following proposition provides a useful characterization of the expansion relation.

Proposition 2.6 *Let R be a relation on $\mathsf{Disc}(X_1) \times \mathsf{Disc}(X_2)$. 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, $\{\mu_{i,1}\}_I$ and $\{\mu_{i,2}\}_I$, such that*

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

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

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

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

Conversely, let p, $\{(\mu_{i,1})\}_{i \in I}$, and $\{(\mu_{i,2})\}_{i \in I}$ satisfy Items 1, 2, and 3. Define $\eta_1(\mu)$ to be the sum $\sum_{i|\mu=\mu_{i,1}} p(i)$ and $\eta_2(\mu)$ to be $\sum_{i|\mu=\mu_{i,2}} p(i)$. Moreover, define $w(\mu_1', \mu_2')$ to be $\sum_{i|\mu_1'=\mu_{i,1}, \mu_2'=\mu_{i,2}} p(i)$. Then, η_1, η_2 and w are witnesses for $\mu_1 \; \mathcal{E}(R) \; \mu_2$. □

The next, rather technical lemma gives us a sufficient condition for showing that a pair of functions f and g preserve the relation $\mathcal{E}(R)$; that is, if $\mu_1 \; \mathcal{E}(R) \; \mu_2$, then $f(\mu_1) \; \mathcal{E}(R) \; g(\mu_2)$. The required condition is that, when μ_1 and μ_2 are decomposed into weighted sums of measures as in the definition of $\mu_1 \; \mathcal{E}(R) \; \mu_2$, f and g convert each pair (ρ_1, ρ_2) of R-related probability measures to $\mathcal{E}(R)$-related probability measures. We will use this lemma in the soundness proof for our new kind of simulation relation (Lemma 4.6), where the two functions f and g apply corresponding sequences of tasks to corresponding measures on executions.

We say that a function f from $\mathsf{Disc}(X)$ to Y *distributes over convex combinations* if for each countable family $\{\rho_i\}_i$ of discrete measures on X and each countable family of probabilities $\{p_i\}_i$ such that $\sum_i p_i = 1$, $f(\sum_i p_i \rho_i) = \sum_i p_i f(\rho_i)$.

Lemma 2.7 *Let R be a relation from $\mathsf{Disc}(X_1)$ to $\mathsf{Disc}(X_2)$, and let f, g be two endo-functions on $\mathsf{Disc}(X_1)$ and $\mathsf{Disc}(X_2)$, respectively, that distribute over convex combinations. Let μ_1 and μ_2 be measures on X_1 and X_2, respectively, such that $\mu_1 \; \mathcal{E}(R) \; \mu_2$. Let η_1, η_2, and w be witnesses for $\mu_1 \; \mathcal{E}(R) \; \mu_2$. Suppose further that, for any two distributions $\rho_1 \in \mathsf{Disc}(X_1)$ and $\rho_2 \in \mathsf{Disc}(X_2)$, $w(\rho_1, \rho_2) > 0$ implies $f(\rho_1) \; \mathcal{E}(R) \; g(\rho_2)$. Then $f(\mu_1) \; \mathcal{E}(R) \; g(\mu_2)$.*

Proof. Let W denote the set of pairs (ρ_1, ρ_2) such that $w(\rho_1, \rho_2) > 0$. Then $f(\rho_1) \; \mathcal{E}(R) \; g(\rho_2)$ whenever $(\rho_1, \rho_2) \in W$. For each $(\rho_1, \rho_2) \in W$, choose a pair of measures $(\eta_1)_{\rho_1,\rho_2}$, $(\eta_2)_{\rho_1,\rho_2}$ and a weighting function $w_{\rho_1\rho_2}$ that witness $f(\rho_1) \; \mathcal{E}(R) \; g(\rho_2)$. Let

$$\begin{aligned}
\eta_1' &:= \sum_{(\rho_1,\rho_2) \in W} w(\rho_1, \rho_2)(\eta_1)_{\rho_1,\rho_2} \\
\eta_2' &:= \sum_{(\rho_1,\rho_2) \in W} w(\rho_1, \rho_2)(\eta_2)_{\rho_1,\rho_2} \\
w' &:= \sum_{(\rho_1,\rho_2) \in W} w(\rho_1, \rho_2) w_{\rho_1,\rho_2}.
\end{aligned}$$

We show that η_1', η_2', and w' are witnesses for $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$ by verifying the three conditions of Definition 2.5.

- $f(\mu_1) = \mathsf{flatten}(\eta_1')$.

  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 2.3, this is in turn equal to $\sum_{(\rho_1,\rho_2)\in W} w(\rho_1,\rho_2)\mathsf{flatten}((\eta_1)_{(\rho_1,\rho_2)})$. By the choice of $(\eta_1)_{(\rho_1,\rho_2)}$ and Item 1 of Definition 2.5, $\mathsf{flatten}((\eta_1)_{(\rho_1,\rho_2)}) = f(\rho_1)$, 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-hand side is equal to $f(\mu_1)$. Since, by Item 1 of Definition 2.5, $\mu_1 = \mathsf{flatten}(\eta_1)$, by the definition of flattening, $\mu_1 = \sum_{\rho_1\in\mathsf{Disc}(X_1)} \eta_1(\rho_1)\rho_1$. 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, Item 2 of Definition 2.4, $\eta_1(\rho_1) = \sum_{\rho_2\in\mathsf{Disc}(X_2)} w(\rho_1,\rho_2)$. Therefore, $f(\mu_1) = \sum_{\rho_1\in\mathsf{Disc}(X_1)}\sum_{\rho_2\in\mathsf{Disc}(X_2)} w(\rho_1,\rho_2)f(\rho_1)$, which in turn is equal to $\sum_{(\rho_1,\rho_2)\in W} w(\rho_1,\rho_2)f(\rho_1)$ since $w(\rho_1,\rho_2)$ is non-zero only in W.

- $g(\mu_2) = \mathsf{flatten}(\eta_2')$.

  Analogous to the previous case.

- $\eta_1'\ \mathcal{L}(R)\ \eta_2'$ with weighting function w'.

  We verify that w' satisfies the three conditions of Definition 2.4:

  1. Let $w'(\rho_1',\rho_2') > 0$. 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 for $\mathcal{L}(R)$, $\rho_1'\ R\ \rho_2'$ as needed.
  2. By definition of w',

$$\begin{aligned}\sum_{\rho_2'\in\mathsf{Disc}(X_2)} w'(\rho_1',\rho_2') &= \sum_{\rho_2'\in\mathsf{Disc}(X_2)}\ \sum_{(\rho_1,\rho_2)\in W} w(\rho_1,\rho_2)w_{\rho_1,\rho_2}(\rho_1',\rho_2') \\ &= \sum_{(\rho_1,\rho_2)\in W}\ \sum_{\rho_2'\in\mathsf{Disc}(X_2)} w(\rho_1,\rho_2)w_{\rho_1,\rho_2}(\rho_1',\rho_2') \\ &= \sum_{(\rho_1,\rho_2)\in W}\left(w(\rho_1,\rho_2)\cdot\sum_{\rho_2'\in\mathsf{Disc}(X_2)} w_{\rho_1,\rho_2}(\rho_1',\rho_2')\right).\end{aligned}$$

     Since w_{ρ_1,ρ_2} is a weighting function for $(\eta_1)_{\rho_1,\rho_2}\ \mathcal{L}(R)\ (\eta_2)_{\rho_1,\rho_2}$, by Item 1 of Definition 2.4, $\sum_{\rho_2'\in\mathsf{Disc}(X_2)} w_{\rho_1,\rho_2}(\rho_1',\rho_2') = (\eta_1)_{\rho_1,\rho_2}(\rho_1')$. Continuing the derivation above,

$$\sum_{\rho_2'\in\mathsf{Disc}(X_2)} w'(\rho_1',\rho_2') \;=\; \sum_{(\rho_1,\rho_2)\in W} w(\rho_1,\rho_2)(\eta_1)_{\rho_1,\rho_2}(\rho_1') \;=\; \eta_1'(\rho_1')$$

     since the middle expression above is precisely the definition of $\eta_1'(\rho_1')$.
  3. Symmetric to the previous case.

□

2.4 Probabilistic I/O Automata

In this subsection, we review basic definitions for Probabilistic I/O Automata and prove several properties that will be needed in the rest of the paper.

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

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

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

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

- *Input enabling:* For every state $q \in Q$ and input action $a \in I$, a is enabled in q.
- *Transition determinism:* For every $q \in Q$ and $a \in A$, there is at most one $\mu \in \mathsf{Disc}(Q)$ such that $(q, a, \mu) \in D$.

A probabilistic I/O automaton is a special case of a probabilistic automaton [Seg95] where the Input/Output distinction and the input enabling restriction of I/O automata [LT89] are introduced. Transition determinism is the classical automata theoretical notion of determinism, though a deterministic automaton would not have any internal actions. It is used in this paper to make sure that an action name denotes a unique transition. In this respect, actions here may assume as well a role similar to the role they play in Markov Decision Processes [Der70]. The restriction to countably many states and actions is just technical to avoid unnecessary and potentially misleading complications with measure theory. Yet, the theory presented here works with uncountable spaces as well.

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

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

We write $\mathsf{fstate}(\alpha)$ for q_0, and, if α is finite, we write $\mathsf{lstate}(\alpha)$ for the last state of α. We use $\mathsf{Frags}(\mathcal{P})$ (resp., $\mathsf{Frags}^*(\mathcal{P})$) to denote the set of all (resp., all finite) execution fragments of $\mathcal{P}$. An *execution* of $\mathcal{P}$ is an execution fragment beginning from the start state $\bar{q}$. $\mathsf{Execs}(\mathcal{P})$ (resp., $\mathsf{Execs}^*(\mathcal{P})$) denotes the set of all (resp., finite) executions of $\mathcal{P}$. The *trace* of an execution fragment α, written $\mathsf{trace}(\alpha)$, is the restriction of α to the set of external actions of $\mathcal{P}$. We say that β is a *trace* of $\mathcal{P}$ if there is an execution α of $\mathcal{P}$ with $\mathsf{trace}(\alpha) = \beta$. The symbol $\leq$ denotes the prefix relation on sequences, which applies in particular to execution fragments and traces.

Notational Conventions We denote the generic elements of a PIOA $\mathcal{P}$ by $Q, \bar{q}, I, O, H, D$ and propagate primes and indices as well. Thus, the elements of a PIOA $\mathcal{P}'_i$ are $Q'_i, \bar{q}'_i, I'_i, O'_i, H'_i, D'_i$, and the set of actions of $\mathcal{P}'_i$ is A'_i.

For a transition relation D, we denote by $D(q)$ the set of transitions that leave from state q, by $D(a)$ the set of transitions labeled by a, and by $D(q, a)$ the set of transitions labeled by a that leave from q. We also write $D(\alpha)$ for $D(\mathsf{lstate}(\alpha))$ and $D(\alpha, a)$ for $D(\mathsf{lstate}(\alpha), a)$. By transition determinism, $D(q, a)$ and $D(\alpha, a)$ contain at most one transition. We denote such transition, if it exists, by tr_{qa} or $tr_{\alpha a}$, and denote its target measure by μ_{qa} or $\mu_{\alpha a}$, respectively.

We let s and q range over generic states, and let a range over generic actions. We denote a generic transition by tr, and we refer to the three elements of tr by q_{tr}, a_{tr}, and μ_{tr}, respectively.

Schedulers and Probabilistic Executions Nondeterministic choices in $\mathcal{P}$ are resolved using a *scheduler*, which is a function that chooses a transition to schedule, possibly randomly, based on the knowledge of the whole past history. Formally, a *scheduler* for $\mathcal{P}$ is a function $\sigma : \mathsf{Frags}^*(\mathcal{P}) \longrightarrow \mathsf{SubDisc}(D)$ such that $(q, a, \mu) \in \mathsf{supp}(\sigma(\alpha))$ implies $q = \mathsf{lstate}(\alpha)$.

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

Once we fix a starting point, that is a state of $\mathcal{P}$, a scheduler can be used to choose the transitions to schedule at each point. Once a transition is chosen, the outcome of the transition is determined by the target probability measure of the transition. Thus, a scheduler together with a starting point induce a probability measure on execution fragments, which we call *probabilistic execution fragments* in general, and *probabilistic executions* whenever the starting point is the start state of $\mathcal{P}$. We define first a σ-field over execution fragments, following the cone construction of [Seg95], which has strong connections with product probability spaces.

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 cone C_α represents the event that α has occurred, possibly followed by something else. The set $\{\alpha\}$, instead, represents the fact that α has occurred and the computation terminated immediately afterwards. Thus, the set of finite execution fragments represents termination. Since the sets of states and actions of PIOAs are countable, the set of finite execution fragments is countable, and thus measurable if we can show that each singleton is measurable. Furthermore, each union of cones is measurable. Each set $\{\alpha\}$, for α a finite execution fragment, is indeed measurable because it can be expressed as $C_\alpha - \cup_{\alpha' | \alpha < \alpha'} C_{\alpha'}$.

Observe that the discrete σ-field over finite execution fragments is a sub-σ-field of $\mathcal{F}_\mathcal{P}$, whose cones are just the cones of $\mathcal{F}_\mathcal{P}$ intersected with the set of finite execution fragments of $\mathcal{P}$. Thus, whenever a sub-probability measure is supported on the set of finite execution fragments, we can safely work with it as if it is a discrete sub-probability measure. We also say that a sub-probability measure ϵ is *finite* if $\mathsf{Frags}^*(\mathcal{P})$ is a support for ϵ. This is consistent with the fact that finite execution fragments capture termination.

We now fix a scheduler σ and a state s. We denote the sub-probability measure induced by σ and s by $\epsilon_{\sigma,s}$ and define it recursively on cones first. The measure of a cone is defined as:

$$\epsilon_{\sigma,s}(C_\alpha) := \begin{cases} 0 & \text{if } \mathsf{fstate}(\alpha) \neq s \\ 1 & \text{if } \alpha = s; \\ \epsilon_{\sigma,s}(C_{\tilde{\alpha}}) \sum_{tr \in D(a)} \sigma(\tilde{\alpha})(tr)\mu_{tr}(q) & \text{if } \mathsf{fstate}(\alpha) = s \text{ and } \alpha = \tilde{\alpha}\ a\ q. \end{cases} \quad (1)$$

Standard applications of the *measure extension theorem* ensure that the measure defined on cones, which is indeed σ-additive, extends uniquely to $\mathcal{F}_\mathcal{P}$. Details can be found in [Seg95] also for the uncountable case, though here we are slightly more general since in [Seg95] definitions are given just for starting states rather than starting finite execution fragments.

Observe that, by definition of σ, the range of the sum in Equation (1) can be restricted to $D(\tilde{\alpha}, a)$. Furthermore, by transition determinism, the set $D(\tilde{\alpha}, a)$ contains at most one element. This leads to a simple and useful property.

Proposition 2.8 *Let σ be a scheduler for PIOA $\mathcal{P}$, and s be a state of $\mathcal{P}$. Let α be a finite execution fragment of $\mathcal{P}$. If $\alpha = \tilde{\alpha} a q$ and a is enabled from $\mathsf{lstate}(\tilde{\alpha})$, then*

$$\epsilon_{\sigma,s}(C_\alpha) = \epsilon_{\sigma,s}(C_{\tilde{\alpha}}) \cdot \sigma(\tilde{\alpha})(D(\tilde{\alpha}, a)) \cdot \mu_{\tilde{\alpha}a}(q).$$

Proof. By definition of measure of a cone, $\epsilon_{\sigma,s}(C_\alpha) = \epsilon_{\sigma,s}(C_{\tilde{\alpha}}) \sum_{tr \in D(a)} \sigma(\tilde{\alpha})(tr)\mu_{tr}(q)$. By action-determinism, there is only one transition enabled from $\mathsf{lstate}(\tilde{\alpha})$ and labeled by a. Such transition is denoted by $tr_{\tilde{\alpha}a}$, and its target probability measure is denoted by $\mu_{\tilde{\alpha}a}$. Thus, $\epsilon_{\sigma,\alpha}(C_{\alpha'}) = \epsilon_{\sigma,s}(C_{\tilde{\alpha}})\sigma(\tilde{\alpha})(tr_{\tilde{\alpha}a})\mu_{\tilde{\alpha}a}(q)$. Since $tr_{\tilde{\alpha}a}$ is the only element of $D(\tilde{\alpha}, a)$, the result follows. □

We extend our cone notation also to finite execution fragments extended with an action, that is, to alternating sequences of states and actions that start with a state and end with an action. The cone $C_{\alpha a}$ is the set of execution fragments that start with the sequence αa. This is a measurable set since it can be seen as the union of the cones $C_{\alpha'}$ such that αa is a prefix of the sequence α'. An equivalent of Proposition 2.8 can be stated for these new cones as well.

Proposition 2.9 *Let σ be a scheduler for PIOA $\mathcal{P}$, and s be a state 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*

$$\epsilon_{\sigma,\epsilon}(C_{\alpha a}) = \epsilon_{\sigma,\epsilon}(C_\alpha) \cdot \sigma(\alpha)(D(\alpha, a)).$$

Proof. Since $C_{\alpha a}$ is the disjoint union of cones $\cup_q C_{\alpha a q}$, $\epsilon_{\sigma,s}(C_{\alpha a}) = \sum_q \epsilon_{\sigma,s}(C_{\alpha a q})$. By Proposition 2.8, $\epsilon_{\sigma,s}(C_{\alpha a q}) = \epsilon_{\sigma,s}(C_\alpha) \cdot \sigma(\alpha)(D(\alpha, a)) \cdot \mu_{\alpha a}(q)$. Since $\mu_{\alpha a}$ is a probability measure on states, the result follows by replacing $\sum_q \mu_{\alpha a}(q)$ by 1. □

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

Notational Conventions We denote by $\sigma(\alpha)(\bot)$ the probability $1 - \sigma(\alpha)(D)$ that scheduler σ induces termination after α.

Recall that a measurable function can be applied to probability measures as well, leading to image measures. Function fstate is trivially measurable, and in particular, when applied to a probabilistic execution fragment, always leads to a Dirac probability measure. We will often abuse of notation and write $\mathsf{fstate}(\epsilon) = s$ instead of $\mathsf{fstate}(\epsilon) = \delta(s)$.

Function lstate is defined on finite execution fragments only and is a measurable function when we work in the σ-field of finite execution fragments. In the sequel we will use several times the notation $\mathsf{lstate}(\epsilon)$ for a finite execution fragment ϵ, meaning the image measure under lstate of ϵ viewed as a measure on finite execution fragments. In essence, $\mathsf{lstate}(\epsilon)(q) = \epsilon(\{\alpha \mid \mathsf{lstate}(\alpha) = q\})$. Alternatively, we can define arbitrarily lstate on infinite executions to be $\bar{q}$ and view lstate as a measurable function on the σ-field of all execution fragments. The final result is the same.

Prefixes We extend the notion of prefix to probabilistic execution fragments, and even more generally to probability measures on execution fragments. Let ϵ and ϵ' be 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 $\mathcal{P}$, $\epsilon(C_\alpha) \leq \epsilon'(C_\alpha)$.

Informally speaking, an execution fragment α is a prefix of an execution fragment α' if α terminates not later than α'. When we add probabilities, we say that ϵ is a prefix of ϵ' if for every finite execution fragment α the probability that ϵ terminates at α or afterwards is not higher than the probability that ϵ' terminates at α or afterwards. In other words, ϵ always terminates not later than ϵ'. Observe that the ordering is imposed only on cones, which allows us to have a non-trivial notion of prefix.

The next proposition states that the ordering on cones of execution fragments holds for cones of execution fragments extended with actions as well.

Proposition 2.10 *If $\epsilon \leq \epsilon'$, then, for each finite execution fragment α and each action a, $\epsilon(C_{\alpha a}) \leq \epsilon'(C_{\alpha a})$.*

Proof. $\epsilon(C_{\alpha a}) = \sum_q \epsilon(C_{\alpha a q}) \leq \sum_q \epsilon'(C_{\alpha a q}) = \epsilon'(C_{\alpha a})$. □

Chains and Limits We prove here a few results about chains of probabilistic execution fragments and their limits. This allows us to define constructions on infinite probabilistic executions based on their finite approximations. Again, we work with generic probability measures on execution fragments, and prove later that in our cases we obtain probabilistic execution fragments.

A *chain* of probability measures on execution fragments of a 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 $i \geq 0$, $\epsilon_i \leq \epsilon_{i+1}$. The limit $\lim_{i\to\infty} \epsilon_i$ of the chain is defined to be the unique probability measure ϵ such that, for each finite execution fragment α,

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

Proposition 2.11 *The limit $\epsilon = \lim_{i\to\infty} \epsilon_i$ is a well defined probability measure. Furthermore, for each $i > 0$, $\epsilon_i \leq \epsilon$.*

Proof. For each finite execution fragment α, $\epsilon(C_\alpha)$ is well defined, non-negative, and bounded by 1 by the monotone convergence theorem since $\epsilon_1(C_\alpha), \epsilon_2(C_\alpha), \ldots$ is a monotone sequence of non-negative reals bounded by 1. Furthermore, for each $i > 0$, $\epsilon_i(C_\alpha) \leq \epsilon(C_\alpha)$, thus showing that for each $i > 0$, $\epsilon_i \leq \epsilon$.

Observe that $\mathsf{Execs}(\mathcal{P}) = \cup_{q\in Q} C_q$. Thus, for each i, $\epsilon_i(\cup_{q\in Q} C_q) = 1$, and consequently, for each i and each state q, $\epsilon_1(C_q) = \epsilon_i(C_q)$. This implies that, for each state q, $\epsilon(C_q) = \epsilon_1(C_q)$. Thus, $\epsilon(\cup_{q\in Q} C_q) = 1$.

We show that ϵ is σ-additive on the cones. Let $C_\alpha = \cup_j C_{\alpha_j}$, where the C_{α_j} sets are pairwise disjoint. Then $\epsilon(C_\alpha) = \lim_{i\to\infty} \epsilon_i(C_\alpha) = \lim_{i\to\infty} \sum_j \epsilon_i(C_{\alpha_j})$, where the first step follows by definition of ϵ and the second step by σ-additivity of ϵ_i. Since all limits are suprema (the ϵ_i's form a chain), limits commute with sums. Thus, $\epsilon(C_\alpha) = \sum_j \lim_{i\to\infty} \epsilon_i(C_{\alpha_j}) = \sum_j \epsilon(\alpha_j)$ as needed.

Finally, we show monotonicity of ϵ. Let $C_\alpha \supseteq \cup_j C_{\alpha_j}$, where the C_{α_j} sets are pairwise disjoint. Then $\epsilon(C_\alpha) = \lim_{i\to\infty} \epsilon_i(C_\alpha) \geq \lim_{i\to\infty} \sum_j \epsilon_i(C_{\alpha_j})$, where the first step follows by definition of ϵ and the second step by monotonicity of ϵ_i. Since limits commute with sums, $\epsilon(C_\alpha) \geq \sum_j \lim_{i\to\infty} \epsilon_i(C_{\alpha_j}) = \sum_j \epsilon(\alpha_j)$ as needed. □

The next proposition states that the limit of a chain of probabilistic execution fragments is also a probabilistic execution fragment.

Proposition 2.12 *Let $\epsilon_1 \leq \epsilon_2 \leq \ldots$ be a chain of probabilistic execution fragments of a PIOA $\mathcal{P}$. Then $\lim_{i\to\infty} \epsilon_i$ is a probabilistic execution fragment of $\mathcal{P}$.*

Proof. Let ϵ denote $\lim_{i\to\infty} \epsilon_i$, and let s be $\mathsf{fstate}(\epsilon)$. Then s is also the first state of each of the ϵ_i's. For each $i \geq 1$, let σ_i be a scheduler such that $\epsilon_i = \epsilon_{\sigma_i,s}$. By Proposition 2.9,

$$\epsilon_{\sigma_i,s}(C_{\alpha a}) = \epsilon_{\sigma_i,s}(C_\alpha) \cdot \sigma_i(\alpha)(D(\alpha, a)). \tag{2}$$

Thus,

$$\sigma_i(\alpha)(D(\alpha, a)) = \frac{\epsilon_{\sigma_i,s}(C_{\alpha a})}{\epsilon_{\sigma_i,s}(C_\alpha)} \qquad \text{if } \epsilon_{\sigma_i,s}(C_\alpha) \neq 0. \tag{3}$$

Define

$$\sigma(\alpha)(tr) := \begin{cases} \dfrac{\epsilon(C_{\alpha a_{tr}})}{\epsilon(C_\alpha)} & \text{if } \epsilon(C_\alpha) > 0; \\ \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Informally, if we ever arrive at α, a transition labeled by a is scheduled from α with the probability that α is extended with a.

We show that σ is a scheduler and that $\epsilon_{\sigma,s} = \epsilon$. To show that σ is a scheduler, we must show that, for each finite execution fragment α, $\sigma(\alpha)$ is a sub-probability measure. By Proposition 2.11, ϵ is a probability measure on execution fragments. Thus, since $C_{\alpha a} \subseteq C_\alpha$, $\epsilon(C_{\alpha a}) \leq \epsilon(C_\alpha)$, which implies $\sigma(\alpha)(tr) \in [0, 1]$ for each tr. We need to show also that $\sigma(\alpha)(D) \leq 1$.

Given a finite execution fragment α and an action a, we know by action-determinism that $D(\alpha, a)$ contains at most one element. Thus, from (4) we get

$$\sigma(\alpha)(D(\alpha,a)) = \begin{cases} \dfrac{\epsilon(C_{\alpha a})}{\epsilon(C_\alpha)} & \text{if } \epsilon(C_\alpha) > 0; \\ \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Observe that $\sigma(\alpha)(D) = \sum_a \sigma(\alpha)(D(\alpha, a))$. If $\epsilon(C_\alpha) = 0$, then trivially $\sigma(\alpha)(D) = 0$ by Equation (6). Otherwise,

$$\sum_a \sigma(\alpha)(D(\alpha,a)) = \frac{\sum_a \epsilon(C_{\alpha a})}{\epsilon(C_\alpha)} = \frac{\epsilon(\cup_a C_{\alpha a})}{\epsilon(C_\alpha)} \leq 1,$$

where the first step follows by Equation (5), the second step follows by σ-additivity of ϵ, and the third step follows by $\cup_a C_{\alpha a} \subseteq C_\alpha$.

Before showing that $\epsilon_{\sigma,s} = \epsilon$, we prove an auxiliary fact. Suppose that $\epsilon(C_\alpha) > 0$. Then,

$$\begin{aligned} \sigma(\alpha)(D(\alpha,a)) &= \epsilon(C_{\alpha a})/\epsilon(C_\alpha) \\ &= \lim_{i\to\infty} \epsilon_{\sigma_i,s}(C_{\alpha a}) / \lim_{i\to\infty} \epsilon_{\sigma_i,s}(C_\alpha) \\ &= \lim_{i\to\infty} \sigma_i(\alpha)(D(\alpha,a)), \end{aligned}$$

where the first step follows by Equation (5), the second step follows by definition of ϵ, and the third step follows by Equation (3) and the fact that $\epsilon_{\sigma_i,s}(C_\alpha) > 0$ definitely, that is, from some point onwards. Thus, from Equation (5),

$$\sigma(\alpha)(D(\alpha,a)) = \begin{cases} \lim_{i\to\infty} \sigma_i(\alpha)(D(\alpha,a)) & \text{if } \epsilon(C_\alpha) > 0; \\ \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

To show that $\epsilon_{\sigma,s} = \epsilon$, we show by induction on the length of a finite execution fragment α that $\epsilon_{\sigma,s}(C_\alpha) = \epsilon(C_\alpha)$. For the base case, let α consist of a single state q. By definition of measure of a cone, Equation (1), the probability of C_q does not depend on the scheduler. Thus, $\epsilon_{\sigma,s}(C_\alpha) = \epsilon(C_\alpha)$ trivially.

For the inductive step, let $\alpha = \tilde{\alpha}aq$. By Proposition 2.8,

$$\epsilon_{\sigma,s}(C_\alpha) = \epsilon_{\sigma,s}(C_{\tilde{\alpha}}) \cdot \sigma(\tilde{\alpha})(D(\tilde{\alpha},a)) \cdot \mu_{\tilde{\alpha}a}(q) \tag{7}$$

and

$$\lim_{i\to\infty} \epsilon_{\sigma_i,s}(C_\alpha) = \lim_{i\to\infty} \Big(\epsilon_{\sigma_i,s}(C_{\tilde{\alpha}}) \cdot \sigma_i(\tilde{\alpha})(D(\tilde{\alpha},a)) \cdot \mu_{\tilde{\alpha}a}(q) \Big).$$

By definition of ϵ, the left-hand side of the equation above is $\epsilon(C_\alpha)$, and $\lim_{i\to\infty} \epsilon_{\sigma_i,s}(C_{\tilde{\alpha}}) = \epsilon(C_{\tilde{\alpha}})$. By induction, $\epsilon(C_{\tilde{\alpha}}) = \epsilon_{\sigma,s}(C_{\tilde{\alpha}})$. Therefore,

$$\epsilon(C_\alpha) = \lim_{i\to\infty} \Big(\epsilon_{\sigma,s}(C_{\tilde{\alpha}}) \cdot \sigma_i(\tilde{\alpha})(D(\tilde{\alpha},a)) \cdot \mu_{\tilde{\alpha}a}(q) \Big). \tag{8}$$

We claim that the right-hand sides of Equations (7) and (8) are equal, which leads to $\epsilon_{\sigma,s}(C_\alpha) = \epsilon(C_\alpha)$, as needed. We distinguish two cases. If $\epsilon(C_{\tilde{\alpha}}) > 0$, then, by Equation (6), the right-hand side of Equation (8) can be rewritten as the right-hand side of Equation (7). If, on the other hand, $\epsilon(C_{\tilde{\alpha}}) = 0$, then by induction, $\epsilon_{\sigma,s}(C_{\tilde{\alpha}}) = 0$ as well. Hence the two right-hand sides are both 0, and thus equal. □

The notion of finiteness, prefix, and chain can be defined on trace distributions as well. In particular, we say that a probability measure τ on traces of $\mathcal{P}$ is *finite* if the set of finite traces is a support for τ, and we say that τ is a *prefix* of τ', denoted by $\tau \leq \tau'$, if, for each finite trace β of $\mathcal{P}$, $\tau(C_\beta) \leq \tau'(C_\beta)$.

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 $i \geq 0$, $\tau_i \leq \tau_{i+1}$. The limit $\lim_{i\to\infty} \tau_i$ of the chain is defined to be the unique probability measure τ such that, for each finite trace β,

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

The interesting property is that the trace function commutes with limits of chains.

Lemma 2.13 *Let $\epsilon_1, \epsilon_2, \cdots$ be a chain of measures on execution fragments. Then $\lim_{i\to\infty} \mathsf{tdist}(\epsilon_i) = \mathsf{tdist}(\lim_{i\to\infty} \epsilon_i)$.*

Proof. Let ϵ denote $\lim_{i\to\infty} \epsilon_i$. 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, and let Θ be the set of minimal execution fragments whose trace is in C_β. Then $\mathsf{trace}^{-1}(C_\beta) = \cup_{\alpha\in\Theta} C_\alpha$, where all the cones are pairwise disjoint. Therefore, for $i \geq 0$, $\mathsf{tdist}(\epsilon_i)(C_\beta) = \sum_{\alpha\in\Theta} \epsilon_i(C_\alpha)$, and $\mathsf{tdist}(\epsilon)(C_\beta) = \sum_{\alpha\in\Theta} \epsilon(C_\alpha)$. Since we have monotone limits here (that is, our limit are also suprema), limits commute with sums and our goal can be restated as showing:

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

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

Composition We define composition of PIOAs as follows.

Definition 2.14 *Two PIOAs $\mathcal{P}_i = (Q_i, \bar{q}_i, I_i, O_i, H_i, D_i)$, $i \in \{1, 2\}$, are said to be* compatible *if $A_i \cap H_j = O_i \cap O_j = \emptyset$ whenever $i \neq j$. In that case, we define their* composition *$\mathcal{P}_1 \| \mathcal{P}_2$ to be the PIOA $(Q_1 \times Q_2,\ (\bar{q}_1, \bar{q}_2),\ (I_1 \cup I_2) \setminus (O_1 \cup O_2),\ O_1 \cup O_2,\ H_1 \cup H_2, D)$, where D is the set of triples $((q_1, q_2), a, \mu_1 \times \mu_2)$ such that*

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

Given a state $q = (q_1, q_2)$ in the composition and $i \in \{1, 2\}$, we use $q\lceil\mathcal{P}_i$ to denote q_i.

The definition of composition can be extended to any finite or countable number of PIOAs rather than just two. Note that if an input of one PIOA is an output of another, then it becomes an output action of the composed automaton.

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

Definition 2.15 *Let $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$ be a PIOA and let $S \subseteq O$. Then $\mathsf{hide}(\mathcal{P}, S)$ is the PIOA $\mathcal{P}'$ that is the same as $\mathcal{P}$ except that $O_{\mathcal{P}'} = O_{\mathcal{P}} - S$ and $H_{\mathcal{P}'} = H_{\mathcal{P}} \cup S$.*

3 Task-PIOAs

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

3.1 Task-PIOA definition

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

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

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

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

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

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

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

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

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

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

In general, one could define various classes of task schedules by specifying what dynamic information may be used in choosing the next task, and this is definitely an interesting topic to investigate in further work. Here, however, we opt for the oblivious version because we intend to model system dynamics separately, via high-level nondeterministic choices as highlighted in Section 1. This is indeed the typical approach followed in the literature on cryptographic protocols, and thus we know that we are already sufficiently general.

The effect of a task schedule on a finite probabilistic execution fragment is described by a function apply that we define below (Definition 3.3). Informally, if the next task to schedule is T, then from each finite execution fragment α we schedule the only transition labeled by an action from T enabled from $\mathsf{lstate}(\alpha)$, if it exists, and we schedule no transition otherwise. The definition described above is not the only possible one. For example, we could consider α as deadlocked whenever task T is not enabled from $\mathsf{lstate}(\alpha)$, and yet have a reasonable definition. We choose not to consider α as deadlocked because we find our proposed definition more elegant and also more adequate to describe a scheduler such as a Round Robin scheduler, where a process that has nothing to do during one of its turns simply skips the turn.

Observe that a task schedule does not give us any ability to schedule any input action of a task-PIOA. This is not a problem since we will use task schedules only on closed task-PIOAs, that is, task-PIOAs with no input actions.

We give the formal definition of function apply on arbitrary probability measures on execution fragments since we will prove only later that the outcome of apply is a probabilistic execution fragment whenever it is applied to a finite probabilistic execution fragment.

Definition 3.3 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA where $\mathcal{P} = (Q, \bar{q}, I, O, H, D)$. Let ϵ be a probability measure on finite execution fragments of $\mathcal{P}$, and γ be a task schedule. Then* $\mathsf{apply}(\epsilon, \gamma)$ *is the probability measure on* $\mathsf{Frags}(\mathcal{P})$ *defined recursively by:*

1. *If* $\gamma = \lambda$, *then* $\mathsf{apply}\,(\epsilon, \gamma) := \epsilon$. *(recall that* λ *denotes the empty sequence.)*
2. *If* $\gamma = T, T \in R$, *then, for each* $\alpha \in \mathsf{Frags}^*(\mathcal{P})$, $\mathsf{apply}\,(\epsilon, T)\,(\alpha) := p_1(\alpha, T) + p_2(\alpha, T)$, *where:*

$$p_1(\alpha, T) = \begin{cases} \epsilon(\tilde{\alpha})\mu(q) & \text{if } \alpha \text{ is of the form } \tilde{\alpha}aq, \text{ with } a \in T \text{ and } (\mathsf{lstate}(\tilde{\alpha}), a, \mu) \in D; \\ 0 & \text{otherwise.} \end{cases}$$

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

3. *If* $\gamma = \gamma' T$, $T \in R$, *then* $\mathsf{apply}\,(\epsilon, \gamma) := \mathsf{apply}\,(\mathsf{apply}\,(\epsilon, \gamma'), T)$.
4. *If* γ *is infinite, then* $\mathsf{apply}\,(\epsilon, \gamma) := \lim_{i \to \infty} \mathsf{apply}\,(\epsilon, \gamma_i)$, *where* γ_i *denotes the length-*i *prefix of* γ.

The first three cases handle finite schedules, while the fourth case handles infinite schedules. The first three cases work only on finite execution fragments, and it is indeed easy to see by induction that the outcome of a finite schedule is a measure supported on finite execution fragments (cf. Lemma 3.5). Case (1) handles the empty schedule, Case (2) describes the outcome of a single task, and Case (3) provides a recursive definition of the outcome of a finite schedule. In Case (2) above, the probability of a finite execution fragment α is computed as the sum of two pieces: the probability of being in α already and not moving from it (term p_2), which is relevant only if T is not enabled after α, and the probability of reaching α by performing a transition from T from a prefix $\tilde{\alpha}$ of α (term p_1). The term p_1 is well defined since, by transition-determinism and action-determinism, the transition $(\mathsf{lstate}(\tilde{\alpha}), a, \eta)$ is unique.

Before showing in Section 3.3 that $\mathsf{apply}\,(\epsilon, \gamma)$ is a well defined probabilistic execution fragment of $\mathcal{P}$, we illustrate an example of an adaptive adversary within task-PIOAs.

3.2 Example: An Adaptive Adversary

In this example, we illustrate how nondeterminism in task-PIOAs is resolved by oblivious task schedules (called low-level scheduling in Section 1) and how system components can act as purposeful, adaptive schedulers by ordering their actions based on the information they see during execution (called high-level adversarial scheduling in Section 1).

Protocol Consider a toy protocol with one sender Sender and two receivers Rec_0 and Rec_1 who exchange messages via an adversarial entity Adv (cf. Figure 1). In addition to sending messages to Rec_0 and Rec_1, the party Sender chooses two random bits b and s independently. The first bit b is announced to the adversary Adv and the second bit s is kept secret until Sender receives an acknowledgment from either Rec_0 or Rec_1. If the acknowledgment from Rec_b arrives *before* the acknowledgment from Rec_{1-b}, that is, if it arrives when the acknowledgment from Rec_{1-b} has not arrived yet, then Sender reveals s to Adv, otherwise s remains secret. A detailed description of Sender is given in Figure 2.

The adversary Adv acts as a message delivery system between Sender, Rec_0 and Rec_1. The messages from Sender to Rec_i are delivered whenever they are available, while the acknowledgments from Rec_i to Sender are not delivered until Sender announces b. Then Adv delivers $\mathsf{ack}_{\mathsf{Sb}}$ *before* delivering $\mathsf{ack}_{\mathsf{S}(1-\mathsf{b})}$. A detailed description of Adv is given in Figure 3.

Finally, a receiver Rec_i simply accepts the message from Sender and responds with an acknowledgment. This is described in Figure 4.[3]

Informally speaking, the nondeterminism in this toy protocol is used for mainly three purposes:

- To allow implementation freedom. For example, Sender can choose to send its messages in any order.

[3] In our descriptions of task-PIOAs in Figures 2, 3, and 4, we use the set notation to specify the tasks explicitly. In the discussion, we follow the notational convention of omitting the parentheses for singletons and use expressions of the form $\mathsf{a}(*)$ to denote a task consisting of those actions that can be obtained by replacing $*$ with a valid argument to a, as specified in the detailed task-PIOA description.

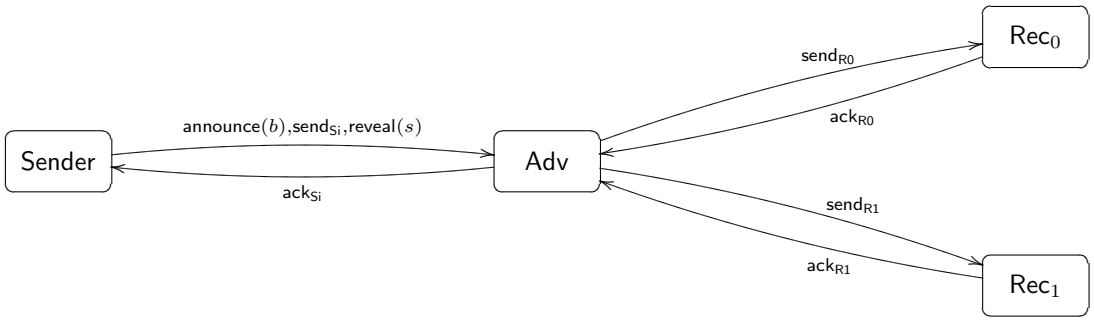

Figure 1: A Toy Protocol

- To allow modeling of different speeds for processes. For example, if $\mathsf{ack}_{\mathsf{R1}}$ of Rec_1 and $\mathsf{ack}_{\mathsf{R2}}$ of Rec_2 are simultaneously enabled then either of them can happen first.
- To model unknown input behavior for components. For example, it is not specified a priori whether Sender will input $\mathsf{ack}_{\mathsf{S1}}$ or $\mathsf{ack}_{\mathsf{S2}}$ first. It depends on the behavior of the component that provides these inputs to Sender, which is Adv in our example.

Adaptive Scheduling by the Adversary Adv can be viewed as resolving the nondeterminism that is used for modeling unknown input behavior for the components with which it interacts. We take a closer look at the output actions of Adv, namely, $\mathsf{send}_{\mathsf{R0}}$,$\mathsf{send}_{\mathsf{R1}}$, $\mathsf{ack}_{\mathsf{S0}}$, and $\mathsf{ack}_{\mathsf{S1}}$. The first two of these actions are used by Adv to relay messages received from Sender to Rec_i. The messages can be delivered in any order by Adv once they are received from the Sender. Note, however, that Adv imposes a finer control on the order of occurrence of the actions $\mathsf{ack}_{\mathsf{S0}}$ and $\mathsf{ack}_{\mathsf{S1}}$. These are the actions that are directly relevant to Adv's ability to learn the secret value s. The conditions that determine whether Adv will send to Sender the acknowledgment from Rec_0 or Rec_1 first, depend on b, which is generated dynamically by Sender and obtained by Adv as a result of the input $\mathsf{announce}(x)$. Adv waits until it learns the value of b as a result of $\mathsf{announce}(x)$, and then it delivers the acknowledgment from Rec_b. This ensures that Sender will reveal s if the task $\mathsf{reveal}(*)$ is scheduled subsequently. In fact, it is easy to check that the task schedule

$$\gamma = \mathsf{choose}.\ \mathsf{announce}(*).\ \mathsf{send}_{\mathsf{S0}}\ .\ \mathsf{send}_{\mathsf{S1}}\ .\ \mathsf{send}_{\mathsf{R0}}.\ \mathsf{send}_{\mathsf{R1}}.\mathsf{ack}_{\mathsf{R0}}.\ \mathsf{ack}_{\mathsf{R1}}\ .\ \mathsf{ack}_{\mathsf{S0}}\ .\ \mathsf{ack}_{\mathsf{S1}}.\ \mathsf{reveal}(*)$$

allows Adv to learn s with probability 1: if $b = 0$ then tasks $\mathsf{ack}_{\mathsf{S0}}$ and $\mathsf{ack}_{\mathsf{S1}}$ are performed in the expected order, and if $b = 1$, then $\mathsf{ack}_{\mathsf{S0}}$ is not performed when scheduled since it is not enabled, and thus $\mathsf{ack}_{\mathsf{S1}}$ occurs first as expected.

The fact that Adv determines the order in which acknowledgments are sent, which affects security, shows that we use Adv to model an adversarial scheduler. Also, the fact that Adv uses b in its scheduling decision shows Adv is adaptive. In general, modeling adversarial schedulers as a system component yields a natural way of modeling adaptiveness. That is, we model the adversary as a task-PIOA that can interact with protocol parties and can compute based on what it sees during execution.

Low-level scheduling by task schedules Task schedules constitute a natural way of resolving the nondeterminism due to unknown conditions such as the relative speeds of processors or network delays. Task schedules were also designed to resolve nondeterminism due to implementation freedom; with the axiom that at most one action in a task is enabled in a given state, specifying a task in turn specifies which action is to be performed if enabled at all.

For instance, the ordering between $\mathsf{send}_{\mathsf{S0}}$ and $\mathsf{send}_{\mathsf{S1}}$ is inessential in the security analysis, provided they are both performed by Sender. Similarly, the ordering between $\mathsf{announce}(*)$ and $\mathsf{send}_{\mathsf{Si}}$ is also inessential.

Sender

Signature

Input:
$\mathsf{ack}_{\mathsf{S0}}$, $\mathsf{ack}_{\mathsf{S1}}$
Output:
$\mathsf{send}_{\mathsf{S0}}$, $\mathsf{send}_{\mathsf{S1}}$
$\mathsf{announce}(x)$, $x \in \{0,1\}$
$\mathsf{reveal}(x)$, $x \in \{0,1\}$
Internal:
choose

Tasks

$\{\mathsf{send}_{\mathsf{S0}}\}$, $\{\mathsf{send}_{\mathsf{S1}}\}$, $\{\mathsf{choose}\}$
$\{\mathsf{announce}(x) | x \in \{0,1\}\}$
$\{\mathsf{reveal}(x) | x \in \{0,1\}\}$

States

$b, s \in \{0, 1, \perp\}$, initially $\perp$
$c \in \{0, 1, \perp\}$, initially $\perp$

Transitions

choose
Precondition:
$b = \perp \wedge s = \perp$
Effect:
$b := \mathsf{unif}(\{0,1\})$
$s := \mathsf{unif}(\{0,1\})$

$\mathsf{announce}(b)$
Precondition:
$b \neq \perp$
Effect:
None

$\mathsf{send}_{\mathsf{Si}}$
Precondition:
True
Effect:
None

$\mathsf{ack}_{\mathsf{Si}}$
Effect:
if $c = \perp$ then $c := i$

$\mathsf{reveal}(s)$
Precondition:
$s \neq \perp \wedge c \neq \perp \wedge b = c$
Effect:
None

Figure 2: Code for Task-PIOA Sender

All of these are examples that represent implementation freedom in Sender. That is, an actual implementation of Sender may perform $\mathsf{announce}(*)$, $\mathsf{send}_{\mathsf{S0}}$ and $\mathsf{send}_{\mathsf{S1}}$ in any order.

Consider the following task schedules, that only differ by the order of the last two tasks.

$$\gamma_1 = \mathsf{send}_{\mathsf{S0}} \,.\, \mathsf{choose}.\ \mathsf{send}_{\mathsf{R0}} \,.\ \mathsf{send}_{\mathsf{S1}} \,.\ \mathsf{send}_{\mathsf{R1}} \,.\ \mathsf{ack}_{\mathsf{R1}} \,.\, \mathsf{announce}(*).\ \mathsf{ack}_{\mathsf{R0}} \,.\ \mathsf{ack}_{\mathsf{S0}} \,.\ \mathsf{ack}_{\mathsf{S1}} \,.\, \mathsf{reveal}(*)$$

$$\gamma_2 = \mathsf{send}_{\mathsf{S0}} \,.\, \mathsf{choose}.\ \mathsf{send}_{\mathsf{R0}} \,.\ \mathsf{send}_{\mathsf{S1}} \,.\ \mathsf{send}_{\mathsf{R1}} \,.\ \mathsf{ack}_{\mathsf{R1}} \,.\, \mathsf{announce}(*).\ \mathsf{ack}_{\mathsf{R0}} \,.\ \mathsf{ack}_{\mathsf{S0}} \,.\, \mathsf{reveal}(*).\ \mathsf{ack}_{\mathsf{S1}} \,.$$

It is interesting to note that Adv learns the secret s with probability 1 under γ_1, but with probability $\frac{1}{2}$ under γ_2. This is because, if $b = 1$, $\mathsf{reveal}(*)$ is not enabled in the state resulting from the performance of $\mathsf{ack}_{\mathsf{S0}}$, so Adv can only guess what b is. Nonetheless, Adv is considered to have high advantage in learning s since we define the behavior of automata and the "security" of a protocol that they represent by quantifying over all possible schedulers (see Sections 3.4 and 3.7).

Note that task schedules can also be viewed as resolving nondeterminism due to unknown input behavior, albeit in a non-adaptive way. If an action a is an input of A and an output of B, and a task schedule dictates the performance of a by B then the task schedule also resolves the nondeterminism in A due to unknown input behavior. However, since task schedulers are oblivious, this kind of nondeterminism resolution is not adaptive on its own. Our modeling approach allows the incorporation of adaptiveness into scheduling by using adversarial components such as Adv above. Note also that since components that do adversarial scheduling can be any task-PIOA, our approach gives us the flexibility to model a wide range of adversarial power.

Adv

Signature

Input:
send_{S0}, send_{S1}, ack_{R0}, ack_{R1}
$\mathsf{announce}(x)$, $x \in \{0,1\}$
$\mathsf{reveal}(x)$, $x \in \{0,1\}$
Output:
send_{R0}, send_{R1}, ack_{S0}, ack_{S1}
Internal:
None

Tasks

$\{\mathsf{send}_{R0}\}$, $\{\mathsf{send}_{R1}\}$, $\{\mathsf{ack}_{S0}\}$, $\{\mathsf{ack}_{S1}\}$

States

$b, s \in \{0, 1, \perp\}$, initially $\perp$
$c_0, c_1 \in \{0, 1, 2, 3\}$, initially 0

Transitions

$\mathsf{announce}(x)$
Effect:
$b := x$

send_{Si}
Effect:
if $c_i = 0$ then $c_i := 1$

send_{Ri}
Precondition:
$c_i = 1$
Effect:
None

ack_{Ri}
Effect:
$c_i := 2$

ack_{Si}
Precondition:
$b \neq \perp \wedge c_i = 2 \wedge (b = i \vee c_{1-i} = 3)$
Effect:
$c_0 := 3$

$\mathsf{reveal}(x)$
Effect:
$s := x$

Figure 3: Code for Task-PIOA Adv

3.3 Properties of the apply Function

In this section we prove several properties of the apply function, and in particular we show that it is a well defined function and that its outcome is a probabilistic execution fragment. Our first lemma states a simple property of functions p_1 and p_2 in the definition of apply, that is, the measure ϵ of a finite execution fragment α is distributed by $p_1(\cdot, T)$ and $p_2(\cdot, T)$ onto α and its immediate successors.

Lemma 3.4 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let ϵ be a probability measure on finite execution fragments of $\mathcal{P}$, and let T be a task. Then:*

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

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

Proof. Item (1) follows trivially from the definition of $p_1(q, T)$. For Item (2), we distinguish two cases.

- If T is not enabled from $\mathsf{lstate}(\alpha)$, then, by definition of p_2, $\epsilon(\alpha) = p_2(\alpha, T)$. Furthermore, for each action a and each state q such that αaq is an execution fragment, we claim that $p_1(\alpha aq, T) = 0$. Indeed, if $a \notin T$, then the first case of the definition of $p_1(\alpha, T)$ trivially does not apply; if $a \in T$, then, since T is not enabled from $\mathsf{lstate}(\alpha)$, there is no μ such that $(\mathsf{lstate}(\alpha), a, \mu) \in D$, and thus, again, the first case of the definition of $p_1(\alpha, T)$ does not apply.

Rec_i

Signature

Input:
$\mathsf{send}_{\mathsf{Ri}}$
Output:
$\mathsf{ack}_{\mathsf{Ri}}$
Internal:
None

Tasks

$\{\mathsf{send}_{\mathsf{Ri}}\}$, $\{\mathsf{ack}_{\mathsf{Ri}}\}$

States

$c \in \{0, 1\}$, initially 0

Transitions

$\mathsf{send}_{\mathsf{Ri}}$
Effect:
$c := 1$

$\mathsf{ack}_{\mathsf{Ri}}$
Precondition:
$c \neq 0$
Effect:
None

Figure 4: Code for Task-PIOA Rec_i

- If T is enabled from $\mathsf{lstate}(\alpha)$, then trivially $p_2(\alpha, T) = 0$. Furthermore, we claim that $\epsilon(\alpha) = \sum_{(a,q)} p_1(\alpha aq, T)$. By action determinism, only one action $b \in T$ is enabled from $\mathsf{lstate}(\alpha)$. By definition of p_1, $p_1(\alpha aq, T) = 0$ if $a \neq b$ (either $a \notin T$ or a is not enabled from $\mathsf{lstate}(\alpha)$). Thus,

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

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

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

The next lemmas shows that indeed apply induces a probability measure on finite execution fragments when applied to finite task schedules.

Lemma 3.5 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let ϵ be a probability measure on finite execution fragments of $\mathcal{P}$, and γ be a finite task schedule. Then* $\mathsf{apply}\,(\epsilon, \gamma)$ *is a probability measure on finite execution fragments.*

Proof. The proof is by induction, where we consider the three cases of Definition 3.3.

- If $\gamma = \lambda$, then $\mathsf{apply}\,(\epsilon, \gamma) = \epsilon$, and the result follows trivially.
- If $\gamma = T$ for some task T, then let ϵ' be $\mathsf{apply}\,(\epsilon, T)$. We show that $\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = 1$. By Item (2) of Definition 3.3,

$$\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = \sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} (p_1(\alpha, T) + p_2(\alpha, T)).$$

  Rearranging terms, we obtain

$$\sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \epsilon'(\alpha) = \sum_{q} p_1(q, T) + \sum_{\alpha \in \mathsf{Frags}^*(\mathcal{P})} \left(p_2(\alpha, T) + \sum_{(a,q):\alpha aq \in \mathsf{Frags}^*(\mathcal{P})} p_1(\alpha aq, T) \right).$$

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

- If $\gamma = \gamma' T$, then, by Item (2) of Definition 3.3, $\mathsf{apply}\,(\epsilon, \gamma) = \mathsf{apply}\,(\mathsf{apply}\,(\epsilon, \gamma), T)$. By induction, $\mathsf{apply}\,(\epsilon, \gamma)$ is a probability measure on finite execution fragments. Then the result follows by the previous case.

□

The next lemma compares the probabilities of cones before and after the application of a single task. An immediate consequence, stated in the two subsequent lemmas, is that the starting probability measure on execution fragments is always a prefix of the outcome of apply, at least for finite task schedules.

Lemma 3.6 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let ϵ be a probability measure on finite execution fragments of $\mathcal{P}$, and T be a task in R. Let ϵ' be $\mathsf{apply}\,(\epsilon, T)$. Then, for each finite execution fragment α:*

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

Proof. By Lemma 3.5, ϵ' is a discrete probability measure. By definition of a cone and of ϵ',

$$\epsilon'(C_\alpha) = \sum_{\alpha' | \alpha \leq \alpha'} \epsilon'(\alpha') = \sum_{\alpha' | \alpha \leq \alpha'} (p_1(\alpha', T) + p_2(\alpha', T)).$$

By definition of a cone and Lemma 3.4,

$$\begin{aligned}
\epsilon(C_\alpha) &= \sum_{\alpha' | \alpha \leq \alpha'} \epsilon(\alpha') \\
&= \sum_{\alpha' | \alpha \leq \alpha'} \left(p_2(\alpha', T) + \sum_{(a,q):\alpha' a q \in \mathsf{Frags}^*(\mathcal{P})} p_1(\alpha' a q, T) \right) \\
&= \sum_{\alpha' | \alpha \leq \alpha'} (p_1(\alpha', T) + p_2(\alpha', T)) - p_1(\alpha, T),
\end{aligned}$$

where the last step is just an algebraic manipulation. Thus, $\epsilon'(C_\alpha) = \epsilon(C_\alpha) + p_1(\alpha, T)$. We distinguish the three cases. If α consists of a single state, then $p_1(\alpha, T) = 0$ by Lemma 3.4, yielding $\epsilon'(C_\alpha) = \epsilon(C_\alpha)$. If $\alpha = \tilde{\alpha} a q$ and $a \notin T$, then $p_1(\alpha, T) = 0$ by definition, yielding $\epsilon'(C_\alpha) = \epsilon(C_\alpha)$. Finally, if $\alpha = \tilde{\alpha} a q$ and $a \in T$, then $p_1(\alpha, T) = \epsilon(\tilde{\alpha})\mu_{\tilde{\alpha}a}(q)$ by definition, yielding $\epsilon'(C_\alpha) = \epsilon(C_\alpha) + \epsilon(\tilde{\alpha})\mu_{\tilde{\alpha}a}(q)$. □

Lemma 3.7 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let ϵ be a probability measure on finite execution fragments of $\mathcal{P}$, and T be a task. Then $\epsilon \leq \mathsf{apply}\,(\epsilon, T)$.*

Proof. Follows directly by Lemma 3.6. □

Lemma 3.8 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let ϵ be a probability measure on finite execution fragments of $\mathcal{P}$, and let γ_1 and γ_2 be two finite task schedules such that γ_1 is a prefix of γ_2. Then $\mathsf{apply}\,(\epsilon, \gamma_1) \leq \mathsf{apply}\,(\epsilon, \gamma_2)$.*

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

Another important consequence of Lemma 3.7 is that Item (4) of Definition 3.3 is well defined, in the sense that the measures $\epsilon_1, \epsilon_2, \ldots$ used in the definition form a chain, and thus the limit is well defined. We prove below even a stronger result, which implies our claim above if all γ_i's are of length 1.

Lemma 3.9 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let $\gamma_1, \gamma_2, \cdots$ be a finite or infinite sequence of finite task schedules, and let $\gamma = \gamma_1\gamma_2\cdots$ (where juxtaposition denotes concatenation of finite sequences). Let I be the length of the sequence. Let ϵ be a probability measure on finite execution fragments of $\mathcal{P}$. For each integer i, $1 \leq i \leq I$, let $\epsilon_i = \mathsf{apply}\,(\epsilon, \gamma_1\gamma_2\cdots\gamma_i)$. Let $\epsilon = \mathsf{apply}\,(\epsilon, \gamma)$. Then the ϵ_i's form a, possibly finite, chain and if $I = \infty$, $\epsilon = \lim_{i\to\infty} \epsilon_i$.*

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

At this stage we know that apply is well defined and that it produces a probability measure on execution fragments. What is left to show is that the outcome of apply is indeed a probabilistic execution fragment (that is, induced by some scheduler from a state) whenever we start from a finite probabilistic execution fragment. We prove first two other properties of apply. The first property states that apply does not increase the probabilities of cones of states; the second property states distributivity of apply with respect to combinations of probability measures.

Lemma 3.10 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let ϵ be a probability measure on finite execution fragments of $\mathcal{P}$, γ a task schedule for $\mathcal{T}$, and q a state of $\mathcal{T}$. Then $\mathsf{apply}\,(\epsilon, \gamma)\,(C_q) = \epsilon(C_q)$.*

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

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

To prove that $\mathsf{apply}\,(\cdot, \gamma)$ distributes over convex combinations of probability measures we consider first the case of a single task.

Lemma 3.11 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let $\{\epsilon_i\}_{i\in I}$ be a countable family of probability measures on finite execution fragments of $\mathcal{P}$, and let $\{p_i\}_{i\in I}$ be a countable family of probabilities such that $\sum_{i\in I} p_i = 1$. Let T be a task. Then $\mathsf{apply}\left(\sum_{i\in I} p_i\epsilon_i, T\right) = \sum_{i\in I} p_i\,\mathsf{apply}\,(\epsilon_i, T)$.*

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

$$\begin{aligned}
\mathsf{apply}\left(\sum_{i\in I} p_i\epsilon_i, T\right)(\alpha) &= p_1(\alpha, T) + p_2(\alpha, T) && \text{definition of } \mathsf{apply}\left(\textstyle\sum_{i\in I} p_i\epsilon_i, T\right)\\
&= \sum_{i\in I} p_i p_1^i(\alpha, T) + \sum_{i\in I} p_i p_2^i(\alpha, T) && \text{claims proved below}\\
&= \sum_{i\in I} p_i(p_1^i(\alpha, T) + p_2^i(\alpha, T)) && \text{algebraic manipulation}\\
&= \sum_{i\in I} p_i\,\mathsf{apply}\,(\epsilon_i, T)\,(\alpha) && \text{definition of } \mathsf{apply}\,(\epsilon_i, T).
\end{aligned}$$

To prove our claim about p_1 we distinguish two cases. If α can be written as $\tilde{\alpha}\ a\ q$, where $\tilde{\alpha} \in \mathsf{supp}(\epsilon), a \in T$, and $(\mathsf{lstate}(\tilde{\alpha}), a, \epsilon) \in D_{\mathcal{P}}$, then, by Definition 3.3, $p_1(\alpha, T) = (\sum_{i\in I} p_i\epsilon_i)(\tilde{\alpha})\mu(q)$,

and, for each i, $p_1^i(\alpha) = \epsilon_i(\tilde{\alpha})\mu(q)$. Thus, $p_1(\alpha, T) = \sum_{i \in I} p_i p_1^i(\alpha, T)$ by definition of combination of measures. Otherwise, again by Definition 3.3, $p_1(\alpha, T) = 0$, and, for each i, $p_1^i(\alpha, T) = 0$. Thus, $p_1(\alpha, T) = \sum_{i \in I} p_i p_1^i(\alpha, T)$ trivially.

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

Proposition 3.12 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let $\{\epsilon_i\}_{i \in I}$ be a countable family of probability measures on finite execution fragments of $\mathcal{P}$, and let $\{p_i\}_{i \in I}$ be a countable family of probabilities such that $\sum_{i \in I} p_i = 1$. Let γ be a finite schedule. Then,* $\mathsf{apply}\left(\sum_{i \in I} p_i \epsilon_i, \gamma\right) = \sum_{i \in I} p_i \, \mathsf{apply}\,(\epsilon_i, \gamma)$.

Proof. We proceed by induction on the length of γ. If $\gamma = \lambda$, 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 γ be $\gamma' T$. By Definition 3.3 and the induction hypothesis,

$$\mathsf{apply}\left(\sum_{i \in I} p_i \epsilon_i, \gamma' T\right) = \mathsf{apply}\left(\mathsf{apply}\left(\sum_{i \in I} p_i \epsilon_i, \gamma'\right), T\right) = \mathsf{apply}\left(\sum_{i \in I} p_i \, \mathsf{apply}\,(\epsilon_i, \gamma'), T\right).$$

By Lemma 3.5, each $\mathsf{apply}\,(\epsilon_i, \gamma')$ is a discrete probability measure on finite execution fragments. By Lemma 3.11, $\mathsf{apply}\left(\sum_{i \in I} p_i \, \mathsf{apply}\,(\epsilon_i, \gamma'), T\right) = \sum_{i \in I} p_i \, \mathsf{apply}\,(\mathsf{apply}\,(\epsilon_i, \gamma'), T)$, and by Definition 3.3, for each i, $\mathsf{apply}\,(\mathsf{apply}\,(\epsilon_i, \gamma'), T) = \mathsf{apply}\,(\epsilon_i, \gamma' T)$. Thus, $\mathsf{apply}\left(\sum_{i \in I} p_i \epsilon_i, \gamma' T\right) = \sum_{i \in I} p_i \, \mathsf{apply}\,(\epsilon_i, \gamma' T)$ as needed. □

Distributivity of apply holds for infinite task schedules as well; however, we refrain from proving the result here since it is not needed in the rest of the paper.

We now turn to the proof that apply returns probabilistic execution fragments when we start from finite probabilistic execution fragments. We first prove separately the result for the empty task schedule and for single tasks. Essentially the proof builds a scheduler that induces the outcome of apply.

Lemma 3.13 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let ϵ be a finite probabilistic execution fragment of $\mathcal{P}$. Then* $\mathsf{apply}\,(\epsilon, \lambda)$ *is a probabilistic execution fragment of $\mathcal{P}$.*

Proof. Follows directly from the definitions since $\mathsf{apply}\,(\epsilon, \lambda) = \epsilon$. □

Lemma 3.14 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. If ϵ is a finite probabilistic execution fragment of $\mathcal{P}$, and T is a task, then* $\mathsf{apply}\,(\epsilon, T)$ *is a finite probabilistic execution fragment of $\mathcal{P}$.*

Proof. Let s be $\mathsf{fstate}(\epsilon)$, ϵ' be $\mathsf{apply}\,(\epsilon, T)$, and σ be a scheduler such that $\epsilon = \epsilon_{\sigma, s}$. We define a new scheduler σ' and prove that $\epsilon' = \epsilon_{\sigma', s}$. For a finite execution fragment α and a transition $tr \in D$,

$$\sigma'(\alpha)(tr) := \begin{cases} 0 & \text{if } \epsilon'(C_\alpha) = 0 \\ \dfrac{\epsilon(C_\alpha)}{\epsilon'(C_\alpha)} \sigma(\alpha)(tr) & \text{if } \epsilon'(C_\alpha) \neq 0 \text{ and } a_{tr} \notin T \\ \dfrac{\epsilon(C_\alpha)}{\epsilon'(C_\alpha)} (\sigma(\alpha)(tr) + \sigma(\alpha)(\bot)) & \text{if } \epsilon'(C_\alpha) \neq 0 \text{ and } a_{tr} \in T \end{cases}$$

We give an informal explanation of the definition of σ'. Recall that $\epsilon(C_\alpha)$ is the probability of ever arriving at α. If we never arrive at α, then there is no need to schedule anything (first case). If we arrive at α, then we should do whatever σ was doing and in addition schedule a transition of task T if it is enabled. The additional probability to give to the transition of task T is the probability with which σ stopped at α while inducing ϵ.

Yet, scheduling T may increase the probability of arriving at α. Our scheduler should act only on the fraction of the ϵ'-probability of arriving at α that was contributed already by ϵ. This is the motivation for the factor $\epsilon(C_\alpha)/\epsilon'(C_\alpha)$.

We first prove that σ' is a scheduler, that is, that $\sigma'(\alpha)$ is a sub-probability measure for each finite execution fragment α. If the first clause applies, then $\sigma'(\alpha)$ is 0 everywhere, hence is a sub-probability measure. Assume otherwise. By Lemma 3.7, $\epsilon \leq \epsilon'$. Thus, $\epsilon(C_\alpha) \leq \epsilon'(C_\alpha)$, and in particular $\epsilon(C_\alpha)/\epsilon'(C_\alpha) \leq 1$. Since σ is a scheduler, then, by definition of σ', $\sigma'(\alpha)(tr) \leq 1$ for each tr. By action- and transition-determinism, there is at most one transition $tr \in D(\alpha)$ with $a_{tr} \in T$. Let Y denote $\{tr\}$ if such tr exists and $\emptyset$ otherwise. Then we have the following.

$$\sum_{tr \notin Y} \sigma(\alpha)(tr) + \sum_{tr \in Y} \big(\sigma(\alpha)(tr) + \sigma(\alpha)(\bot)\big) \leq \left(\sum_{tr \in D} \sigma(\alpha)(tr)\right) + \sigma(\alpha)(\bot) = 1,$$

where the first step follows from the fact that Y has at most one element and the second step follows from the fact that σ is a scheduler. Putting the pieces together, we have

$$\sigma'(\alpha)(D) = \frac{\epsilon(C_\alpha)}{\epsilon'(C_\alpha)} \left(\sum_{tr \notin Y} \sigma(\alpha)(tr) + \sum_{tr \in Y} (\sigma(\alpha)(tr) + \sigma(\alpha)(\bot))\right) \leq 1.$$

Next, we prove by induction on the length of a finite execution fragment α that $\epsilon_{\sigma',s}(C_\alpha) = \epsilon'(C_\alpha)$. For the base case, let $\alpha = q$. Then $\epsilon_{\sigma',s}(C_q) = \epsilon'(C_q)$ trivially since, by Equation (1), the measure of a cone of a state does not depend on the scheduler. For the inductive step, let $\alpha = \tilde{\alpha}aq$. By Proposition 2.8

$$\epsilon_{\sigma',s}(C_\alpha) = \epsilon_{\sigma',s}(C_{\tilde{\alpha}}) \cdot \sigma'(\tilde{\alpha})(tr_{\tilde{\alpha}a}) \cdot \mu_{\tilde{\alpha}a}(q). \tag{9}$$

where we recall that $tr_{\tilde{\alpha}a}$ is the only transition of $D(\tilde{\alpha}, a)$ and $\mu_{\tilde{\alpha}a}$ is the target measure of $tr_{\tilde{\alpha}a}$. By induction, $\epsilon_{\sigma',s}(C_{\tilde{\alpha}}) = \epsilon'(C_{\tilde{\alpha}})$. Thus,

$$\epsilon_{\sigma',s}(C_\alpha) = \epsilon'(C_{\tilde{\alpha}}) \cdot \sigma'(\tilde{\alpha})(tr_{\tilde{\alpha}a}) \cdot \mu_{\tilde{\alpha}a}(q). \tag{10}$$

We distinguish three cases.

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

   By induction, $\epsilon_{\sigma',s}(C_{\tilde{\alpha}}) = \epsilon'(C_{\tilde{\alpha}}) = 0$. Then $\epsilon_{\sigma',s}(C_\alpha) = 0 = \epsilon'(C_\alpha)$ trivially since $C_{\tilde{\alpha}} \subseteq C_\alpha$.

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

   By replacing σ' with its definition in Equation (10), and simplifying the factors $\epsilon'(C_{\tilde{\alpha}})$,

   $$\epsilon_{\sigma',s}(C_\alpha) = \epsilon(C_{\tilde{\alpha}})\sigma(\tilde{\alpha})(tr_{\tilde{\alpha}a}) \cdot \mu_{\tilde{\alpha}a}(q). \tag{11}$$

   Recall that $\epsilon = \epsilon_{\sigma,s}$. By Proposition 2.8 the right-hand side of the equation above is $\epsilon_{\sigma,s}(C_\alpha)$. Thus, $\epsilon_{\sigma',s}(C_\alpha) = \epsilon(C_\alpha)$. By Lemma 3.6, Item (3), $\epsilon(C_\alpha) = \epsilon'(C_\alpha)$. Hence, $\epsilon_{\sigma',s}(C_\alpha) = \epsilon'(C_\alpha)$.

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

   By replacing σ' with its definition in Equation (10), and simplifying the factors $\epsilon'(C_{\tilde{\alpha}})$,

   $$\epsilon_{\sigma',s}(C_\alpha) = \epsilon(C_{\tilde{\alpha}})(\sigma(\tilde{\alpha}) \cdot \big(tr_{\tilde{\alpha}a}) + \sigma(\tilde{\alpha})(\bot)\big) \cdot \mu_{\tilde{\alpha}a}(q). \tag{12}$$

   Recall that $\epsilon = \epsilon_{\sigma,s}$. By Proposition 2.8, the equation above can be rewritten as

   $$\epsilon_{\sigma',s}(C_\alpha) = \epsilon(C_\alpha) + \epsilon(C_{\tilde{\alpha}}) \cdot \sigma(\tilde{\alpha})(\bot) \cdot \mu_{\tilde{\alpha}a}(q). \tag{13}$$

   Observe that $\epsilon(C_{\tilde{\alpha}}) \cdot \sigma(\tilde{\alpha})(\bot) = \epsilon(\tilde{\alpha})$. By Lemma 3.6, Item (3), $\epsilon(C_\alpha) + \epsilon(\tilde{\alpha})\mu_{\tilde{\alpha}a}(q) = \epsilon'(C_\alpha)$. Hence, $\epsilon_{\sigma',s}(C_\alpha) = \epsilon'(C_\alpha)$.

□

By combining the previous two lemmas we get that the outcome of any finite schedule is a probabilistic execution fragment.

Lemma 3.15 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let ϵ be a finite probabilistic execution fragment of $\mathcal{P}$, and γ be a finite schedule. Then,* $\mathsf{apply}\,(\epsilon, \gamma)$ *is a finite probabilistic execution fragment of $\mathcal{P}$.*

Proof. By induction on the length of γ. If $\gamma = \lambda$, then the result follows by Lemma 3.13. If $\gamma = \gamma' T$, then, by definition, $\mathsf{apply}\,(\epsilon, \gamma) = \mathsf{apply}\,(\mathsf{apply}\,(\epsilon, \gamma'), T)$. By induction, $\mathsf{apply}\,(\epsilon, \gamma')$ is a finite probabilistic execution fragment of $\mathcal{P}$. By Lemma 3.14, $\mathsf{apply}\,(\mathsf{apply}\,(\epsilon, \gamma'), T)$ is a finite probabilistic execution fragment of $\mathcal{P}$. □

Finally, we deal with infinite task schedules.

Lemma 3.16 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let ϵ be a finite probabilistic execution fragment of $\mathcal{P}$, and γ be an infinite schedule. Then,* $\mathsf{apply}\,(\epsilon, \gamma)$ *is a probabilistic execution fragment of $\mathcal{P}$.*

Proof. For each $i \geq 0$, let γ_i denote the length-i prefix of γ and let ϵ_i be $\mathsf{apply}\,(\epsilon, \gamma_i)$. By Lemmas 3.15 and 3.8, the sequence $\epsilon_0, \epsilon_1, \ldots$ is a chain of probabilistic execution fragments of $\mathcal{P}$. By Proposition 2.12, $\lim_{i \to \infty} \epsilon_i$ is a probabilistic execution fragment of $\mathcal{P}$. This suffices, since $\mathsf{apply}\,(\epsilon, \gamma)$ is $\lim_{i \to \infty} \epsilon_i$ by definition. □

Theorem 3.17 *Let $\mathcal{T} = (\mathcal{P}, R)$ be an action-deterministic task-PIOA. Let ϵ be a finite probabilistic execution fragment of $\mathcal{P}$, and γ be a schedule. Then,* $\mathsf{apply}\,(\epsilon, \gamma)$ *is a probabilistic execution fragment of $\mathcal{P}$.*

Proof. Follows by Lemma 3.15 if γ is finite and by Lemma 3.16 if γ is infinite □

3.4 Trace Distributions

We write $\mathsf{tdist}(\epsilon, \gamma)$ as shorthand for $\mathsf{tdist}(\mathsf{apply}\,(\epsilon, \gamma))$, the trace distribution obtained by applying task schedule γ starting from the measure ϵ on execution fragments. We write $\mathsf{tdist}(\gamma)$ for $\mathsf{tdist}(\mathsf{apply}\,(\delta(\bar{q}), \gamma))$ the trace distribution obtained by applying γ from the unique start state. (Recall that $\delta(\bar{q})$ denotes the Dirac measure on $\bar{q}$.) A *trace distribution* of $\mathcal{T}$ is any $\mathsf{tdist}(\gamma)$. We use $\mathsf{tdists}(\mathcal{T})$ to denote the set $\{\mathsf{tdist}(\gamma) : \gamma \text{ is a task schedule for } \mathcal{T}\}$.

3.5 Composition

We define composition of task-PIOAs:

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

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

3.6 Hiding

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

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

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

3.7 Implementation

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

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

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

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

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

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

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

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

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

The subscript 0 in the relation symbol $\leq_0$ refers to the requirement that every trace distribution in $\mathsf{tdists}(\mathcal{T}_1||\mathcal{E})$ must have an identical match in $\mathsf{tdists}(\mathcal{T}_2||\mathcal{E})$. For security analysis, we also define another relation $\leq_{neg,pt}$, which allows "negligible" discrepancies between matching trace distributions [CCK+05b, CCK+06c].

3.8 Compositionality

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

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

Proof. Let $\mathcal{T}_4 = (\mathcal{P}_4, R_4)$ be any environment (action-deterministic) task-PIOA for both $\mathcal{T}_1\|\mathcal{T}_3$ and $\mathcal{T}_2\|\mathcal{T}_3$. Fix any task schedule γ_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 γ_1. It suffices to show that τ is also generated by some task schedule γ_2 for $(\mathcal{T}_2\|\mathcal{T}_3)\|\mathcal{T}_4$.

Note that γ_1 is also a task schedule for $\mathcal{T}_1\|(\mathcal{T}_3\|\mathcal{T}_4)$, and that γ_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 γ_2 for $\mathcal{T}_2\|(\mathcal{T}_3\|\mathcal{T}_4)$ such that γ_2 generates trace distribution τ in the task-PIOA $\mathcal{T}_2\|(\mathcal{T}_3\|\mathcal{T}_4)$. Since γ_2 is also a task schedule for $(\mathcal{T}_2\|\mathcal{T}_3)\|\mathcal{T}_4$ and γ_2 generates τ, this suffices. □

4 Simulation Relations

Simulation relations provide sufficient conditions for proving that one automaton implements another, according to some precise notion of implementation such as $\leq_0$ from Definition 3.23. Typically, simulation relations reduce the proof obligations for implementation into conditions relating the start states and conditions relating individual algorithm steps. Checking these individual conditions is generally much easier, and more systematic, than reasoning about entire executions.

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

4.1 Simulation relation definition

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

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

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

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

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

5points/measures and states/executions. Overall we end up with four simulation relation. However, language inclusion is not implied.

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

1. Start condition: $\delta(\bar{q}_1)\ R\ \delta(\bar{q}_2)$.
2. Step condition: *If $\epsilon_1\ R\ \epsilon_2$, $\gamma_1 \in {R_1}^*$, ϵ_1 is consistent with γ_1, ϵ_2 is consistent with $\mathsf{full}(\mathsf{c})(\gamma_1)$, and $T \in R_1$, then $\epsilon_1'\ \mathcal{E}(R)\ \epsilon_2'$ where $\epsilon_1' = \mathsf{apply}\,(\epsilon_1, T)$ and $\epsilon_2' = \mathsf{apply}\,(\epsilon_2, \mathsf{c}(\gamma_1, T))$.*

Intuitively, $\epsilon_1\ R\ \epsilon_2$ means that it is possible to simulate from ϵ_2 anything that can happen from ϵ_1. Furthermore, $\epsilon'_1\ \mathcal{E}(R)\ \epsilon'_2$ means that we can decompose ϵ'_1 and ϵ'_2 into pieces that can simulate each other, and so we can also say that it is possible to simulate from ϵ'_2 anything that can happen from ϵ'_1. This rough intuition is at the base of the proof of our soundness result, Theorem 4.7.

Recall from Proposition 2.6 that the expansion of a relation can be characterized in other operational ways that may simplify proofs in practical applications. Since in the examples of this paper we take advantage of the characterization given by Proposition 2.6 (cf. Example 4.3), we provide below an alternative definition of simulation relation that combines all pieces together.

Lemma 4.4 *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 $\mathsf{Disc}(\mathsf{Execs}^*(\mathcal{P}_1))$ to $\mathsf{Disc}(\mathsf{Execs}^*(\mathcal{P}_2))$, such that, if $\epsilon_1\ R\ \epsilon_2$, then $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$. Let $\mathsf{c} : ({R_1}^* \times R_1) \to {R_2}^*$. Suppose further that the following conditions hold:*

1. Start condition: $\delta(\bar{q}_1)\ R\ \delta(\bar{q}_2)$.
2. Step condition: *If $\epsilon_1\ R\ \epsilon_2$, $\gamma_1 \in {R_1}^*$, ϵ_1 is consistent with γ_1, ϵ_2 is consistent with $\mathsf{full}(\mathsf{c})(\gamma_1)$, 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 executions of $\mathcal{P}_1$, and*
   - *probability measures ϵ'_{2j}, $j \in I$, on finite executions 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*
   - *$\sum_{j \in I} p(j)(\epsilon'_{2j}) = \mathsf{apply}\,(\epsilon_2, \mathsf{c}(\gamma_1, T))$.*

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

Proof. By Proposition 2.6, the probability measure p and the probability measures ϵ'_{1j} and ϵ'_{2j}, $j \in I$, exist iff $\mathsf{apply}\,(\epsilon_1, T)\ \mathcal{E}(R)\ \mathsf{apply}\,(\epsilon_2, \mathsf{c}(\gamma_1, T))$. □

4.2 Soundness

In this section, we state and prove two soundness results. The first result, Theorem 4.7, says that, for closed task-PIOAs, the existence of a simulation relation implies inclusion of sets of trace distributions. The second soundness result, Corollary 4.8, asserts soundness for (not necessarily closed) task-PIOAs, with respect to the $\leq_0$ relation.

The proof is based on two lemmas. Recall that the definition of simulation relations requires that any two R-related probability measures on executions must have the same trace distribution. Lemma 4.5 shows that the same continues to hold for any two $\mathcal{E}(R)$ probability measures on executions. For the proof, the only property of simulation relations we need is that any two R-related probability measures on executions have the same trace distribution.

Lemma 4.5 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be comparable closed action-deterministic task-PIOAs and let R be a simulation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Let ϵ_1 and ϵ_2 be discrete probability measures on finite executions of $\mathcal{T}_1$ and $\mathcal{T}_2$, respectively, such that $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$. Then $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$.*

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

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

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

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

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

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

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

The second lemma provides the inductive step needed in the proof of soundness (Theorem 4.7).

Lemma 4.6 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable action-deterministic closed task-PIOAs and let R be a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$. Furthermore, let c be a mapping witnessing the fact that R is a simulation relation. Let a finite task schedule γ_1 of $\mathcal{T}_1$ be given and set $\gamma_2 = \mathsf{full}(\mathsf{c})(\gamma_1)$. (Then γ_2 is a finite task schedule of $\mathcal{T}_2$.) Let ϵ_1 denote $\mathsf{apply}\,(\delta(\bar{q}_1), \gamma_1)$ and let ϵ_2 denote $\mathsf{apply}\,(\delta(\bar{q}_2), \gamma_2)$. Suppose that $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$.*

Now let T be a task of $\mathcal{T}_1$. Let $\epsilon_1' = \mathsf{apply}\,(\delta(\bar{q}_1), \gamma_1 T)$ and let $\epsilon_2' = \mathsf{apply}\,(\delta(\bar{q}_2), \gamma_2\, \mathsf{c}(\gamma_1, T))$. 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 $\epsilon_2' = \mathsf{apply}\,(\epsilon_2, \mathsf{c}(\gamma_1, T))$.

We apply Lemma 2.7: define the function f on discrete distributions on finite executions of $\mathcal{T}_1$ by $f(\epsilon) = \mathsf{apply}\,(\epsilon, T)$, and the function g on discrete distributions on finite executions of $\mathcal{T}_2$ by $g(\epsilon) = \mathsf{apply}\,(\epsilon, \mathsf{c}(\gamma_1, T))$. We show that the hypothesis of Lemma 2.7 is satisfied, so we can invoke Lemma 2.7 to conclude that $\epsilon_1'\ \mathcal{E}(R)\ \epsilon_2'$.

Distributivity of f and g follows directly by Proposition 3.12. Let μ_1, μ_2 be two measures such that $w(\mu_1, \mu_2) > 0$. We must show that $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$. Since w is a weighting function for $\epsilon_1\ \mathcal{E}(R)\ \epsilon_2$, $\mu_1\ R\ \mu_2$. Observe that $\mathsf{supp}(\mu_1) \subseteq \mathsf{supp}(\epsilon_1)$ and $\mathsf{supp}(\mu_2) \subseteq \mathsf{supp}(\epsilon_2)$; thus, μ_1 is consistent with γ_1 and μ_2 is consistent with γ_2. By the step condition for R, $\mathsf{apply}\,(\mu_1, T)\ \mathcal{E}(R)\ \mathsf{apply}\,(\mu_2, \mathsf{c}(\gamma_1, T))$. Observe that $\mathsf{apply}\,(\mu_1, T) = f(\mu_1)$ and that $\mathsf{apply}\,(\mu_2, \mathsf{c}(\gamma_1, T)) = g(\mu_2)$. Thus, $f(\mu_1)\ \mathcal{E}(R)\ g(\mu_2)$, as needed. □

The following theorem is the main soundness result. The proof simply puts the pieces together, using Lemma 3.9 (which says that the probabilistic execution generated by an infinite task scheduler can be seen as the limit of the probabilistic executions generated by some of the finite prefixes of the task scheduler), Lemma 4.6 (the step condition), Lemma 4.5 (related probabilistic executions have the same trace distribution), and Lemma 2.13 (limit commutes with $tdist$).

Theorem 4.7 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be comparable task-PIOAs that are closed and action-deterministic. If there exists a simulation relation from $\mathcal{T}_1$ to $\mathcal{T}_2$, then $\mathsf{tdists}(\mathcal{T}_1) \subseteq \mathsf{tdists}(\mathcal{T}_2)$.*

Proof. 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 γ_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 $\gamma_1\gamma_2\cdots$. It is sufficient to prove $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$.

For each $j \geq 0$, let $\epsilon_{1,j} = \mathsf{apply}\,(\bar{q}_1, T_1 \cdots T_j)$, and $\epsilon_{2,j} = \mathsf{apply}\,(\bar{q}_2, \gamma_1 \cdots \gamma_j)$. Then by Lemma 3.9, for each $j \geq 0$, $\epsilon_{1,j} \leq \epsilon_{1,j+1}$ and $\epsilon_{2,j} \leq \epsilon_{2,j+1}$; moreover, $\lim_{j\to\infty} \epsilon_{1,j} = \epsilon_1$ and $\lim_{j\to\infty} \epsilon_{2,j} = \epsilon_2$. Also, for every $j \geq 0$, $\mathsf{apply}\,(\epsilon_{1,j}, T_{j+1}) = \epsilon_{1,j+1}$ and $\mathsf{apply}\,(\epsilon_{2,j}, \gamma_{j+1}) = \epsilon_{2,j+1}$.

Observe that $\epsilon_{1,0} = \delta(\bar{q}_1)$ and $\epsilon_{2,0} = \delta(\bar{q}_2)$. The start condition for a simulation relation and a trivial expansion imply that $\epsilon_{1,0}\ \mathcal{E}(R)\ \epsilon_{2,0}$. Then by induction, using Lemma 4.6 for the definition of a simulation relation in proving the inductive step, for each $j \geq 0$, $\epsilon_{1,j}\ \mathcal{E}(R)\ \epsilon_{2,j}$. Then, by Lemma 4.5, for each $j \geq 0$, $\mathsf{tdist}(\epsilon_{1,j}) = \mathsf{tdist}(\epsilon_{2,j})$.

By Lemma 2.13, $\mathsf{tdist}(\epsilon_1) = \lim_{j\to\infty} \mathsf{tdist}(\epsilon_{1,j})$, and $\mathsf{tdist}(\epsilon_2) = \lim_{j\to\infty} \mathsf{tdist}(\epsilon_{2,j})$. Since for each $j \geq 0$, $\mathsf{tdist}(\epsilon_{1,j}) = \mathsf{tdist}(\epsilon_{2,j})$, we conclude that $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$, as needed. □

Corollary 4.8 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two comparable action-deterministic task-PIOAs. Suppose that, for every environment $\mathcal{E}$ for both $\mathcal{T}_1$ and $\mathcal{T}_2$, there exists a simulation relation R from $\mathcal{T}_1\|\mathcal{E}$ to $\mathcal{T}_2\|\mathcal{E}$. Then $\mathcal{T}_1 \leq_0 \mathcal{T}_2$.*

Proof. Immediate by Theorem 4.7 and the definition of $\leq_0$. □

4.3 Example: Trapdoor vs. Rand

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

More precisely, neither Rand nor Trapdoor has any input actions. Rand has output actions $\mathsf{report}(k)$, $k \in [n] = \{1, \ldots, n\}$ and an internal action choose. It has tasks $\mathsf{Report} = \{\mathsf{report}(k) : k \in [n]\}$, and $\mathsf{Choose} = \{\mathsf{choose}\}$. Its state contains one variable $zval$, which assumes values in $[n] \cup \{\perp\}$, initially $\perp$. The choose action is enabled when $zval = \perp$, and has the effect of setting $zval$ to a number in $[n]$, chosen uniformly at random. The $\mathsf{report}(k)$ action is enabled when $zval = k$, and has no effect on the state (so it may happen repeatedly). Precondition/effect code for Rand appears in Figure 5, and a diagram appears in Figure 6.

Rand

Signature

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

Tasks

$\mathsf{Report} = \{\mathsf{report}(k) : k \in \{1, \ldots, n\}\}$
$\mathsf{Choose} = \{\mathsf{choose}\}$

States

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

Transitions

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

$\mathsf{report}(k)$
Precondition:
$zval = k$
Effect:
None

Figure 5: Code for Task-PIOA Rand

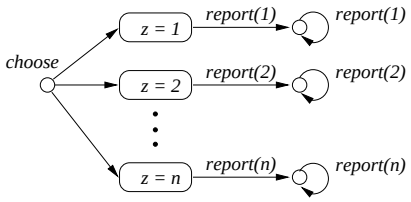

Figure 6: Task-PIOA Rand

Trapdoor has the same actions as Rand, plus internal action compute. It has the same tasks as Rand, plus the task Compute $= \{\text{compute}\}$. Trapdoor's state contains two variables, y and z, each of which takes on values in $[n] \cup \{\perp\}$, initially $\perp$. The choose action is enabled when $y = \perp$, and sets y to a number in $[n]$, chosen uniformly at random. The compute action is enabled when $y \neq \perp$ and $z = \perp$, and sets $z := f(y)$. The report(k) action behaves exactly as in Rand. Precondition/effect code for Trapdoor appears in Figure 7, and a diagram appears in Figure 8.

Trapdoor

Signature

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

Tasks

Report $= \{\text{report}(k) : k \in \{1, \ldots, n\}\}$
Choose $= \{\text{choose}\}$, Compute $= \{\text{compute}\}$

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:
$yval := \text{random}(\text{unif}(\{1, \ldots, n\}))$

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

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

Figure 7: Code for Task-PIOA Trapdoor

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

Simulation relation: We are able to express this correspondence using a simulation relation in the sense of Definition 4.3. Let $\epsilon_1 \in \text{Disc}(\text{Execs}^*(\text{Trapdoor}))$ and $\epsilon_2 \in \text{Disc}(\text{Execs}^*(\text{Rand}))$ such that $\text{tdist}(\epsilon_1) =$

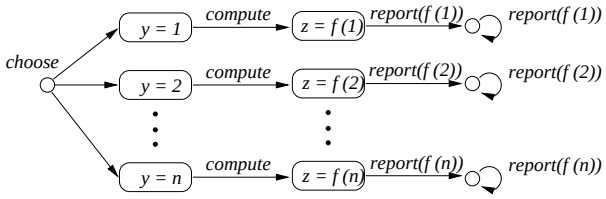

Figure 8: Task-PIOA Trapdoor

$\mathsf{tdist}(\epsilon_2)$.[4] Then we say that ϵ_1 and ϵ_2 are related by R whenever the following conditions hold:

1. For every $s \in \mathsf{supp}(\mathsf{lstate}(\epsilon_1))$ and $u \in \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$, $s.zval = u.zval$.
2. For every $u \in \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$, if $u.zval = \perp$ then either
   (a) $\mathsf{lstate}(\epsilon_1).yval$ is the Dirac distribution on $\perp$, or else
   (b) $\mathsf{lstate}(\epsilon_1).yval$ is the uniform distribution on $[n]$.

We define the task correspondence mapping c as follows.[5]

- $\mathsf{c}(\gamma, \mathsf{Choose}) = \lambda$.
- If γ contains Choose, then $\mathsf{c}(\gamma, \mathsf{Compute}) = \mathsf{Choose}$; otherwise, $\mathsf{c}(\gamma, \mathsf{Compute}) = \lambda$.
- $\mathsf{c}(\gamma, \mathsf{Report}) = \mathsf{Report}$.

Proving the simulation relation: We now prove that the relation R defined above is a simulation relation in the sense of Definition 4.3. We do this by showing that R satisfies the two conditions, namely the start condition and the step condition, in Lemma 4.4. Note that Property 1 of R requires that for any ϵ_1 and ϵ_2 such that $\epsilon_1\ R\ \epsilon_2$, the last states of executions in $\mathsf{supp}(\epsilon_1) \cup \mathsf{supp}(\epsilon_2)$ have the same value for $zval$. This property is used mainly to argue about the uniform enabling or disabling of tasks in the last states of executions in $\mathsf{supp}(\epsilon_1) \cup \mathsf{supp}(\epsilon_2)$. The proof case where $T = \mathsf{Compute}$ and $\mathsf{Choose} \in \gamma_1$ is an interesting case; the probability measure p that we use to show the step condition (cf. Lemma 4.4) is not simply a Dirac measure on a singleton I. We need to use an index set I which is not a singleton, and define ϵ'_{1j} and ϵ'_{2j} for $j \in I$ such that each j corresponds to a particular value in $[n]$. This ensures that we can preserve R in the step condition.

The following two invariant properties are used in the simulation proof. They both follow from easy inductive arguments.

Lemma 4.9 *In every reachable state s of* Trapdoor*, if $s.zval \neq \perp$ then $s.yval \neq \perp$.*

Lemma 4.10 *Suppose γ is a finite task schedule for* Trapdoor*, and ϵ is a discrete distribution on finite executions of* Trapdoor *that is consistent with γ. Let $s \in \mathsf{supp}(\mathsf{lstate}(\epsilon))$ be given. Suppose further that* Choose $\in \gamma$*, that is, the task* Choose *occurs in the schedule γ. Then $s.yval \neq \perp$. If* Choose $\notin \gamma$*, then $s.yval = \perp$.*

Lemma 4.11 *The relation R is a simulation relation from* Trapdoor *to* Rand *using the mapping*

$$\mathsf{c} : R^*_{Trapdoor} \times R_{Trapdoor} \rightarrow R^*_{Rand}$$

defined above.

[4] In the extended abstract [CCK+06b], this condition is missing.

[5] In the extended abstract [CCK+06b], the definition of c contains a small error. Namely, in the second clause, $\mathsf{c}(\gamma, \mathsf{Compute})$ is set to Choose even if γ does not contain Choose.

Proof. We show that R satisfies the two conditions in Lemma 4.4.
Start condition: The Dirac measures on executions consisting of the unique start states s and u of, respectively, Trapdoor and Rand are R-related. Properties 1 and 2 of R hold since for every $s \in \mathsf{supp}\,(\mathsf{lstate}(\epsilon_1))$ and $u \in \mathsf{supp}\,(\mathsf{lstate}(\epsilon_2))$, $s.yval = s.zval = u.zval = \perp$.
Step condition: Suppose $\epsilon_1\ R\ \epsilon_2$, $\gamma_1 \in {R_1}^*$, ϵ_1 is consistent with γ_1, ϵ_2 is consistent with $full(\mathsf{c})(\gamma_1)$. Let $\epsilon'_1 = \mathsf{apply}\,(\epsilon_1, T)$ and $\epsilon'_2 = \mathsf{apply}\,(\epsilon_1, \mathsf{c}(\gamma_1, T))$. We now successively consider each task T in R_1.

1. $T = \mathsf{Choose}$.

   Fix any pair of states $s \in \mathsf{supp}(\mathsf{lstate}(\epsilon_1))$ and $u \in \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$. We consider two cases, according to the value of $s.zval$.

   (a) $s.zval \neq \perp$. By Property 1 of R, for every $v \in \mathsf{supp}(\mathsf{lstate}(\epsilon_1))$, we have that $v.zval \neq \perp$. By the invariant expressed in Lemma 4.9, for every $v \in \mathsf{supp}(\mathsf{lstate}(\epsilon_1))$, we have that $v.yval \neq \perp$. As a result, T is disabled in every state in $\mathsf{supp}(\mathsf{lstate}(\epsilon_1))$.
   Let I be the singleton index $\{1\}$, let p be the Dirac measure on 1 and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. By Definition 3.3 we have $\epsilon'_1 = \epsilon_1$, $\epsilon'_2 = \epsilon_2$. Since $\epsilon_1\ R\ \epsilon_2$, we have $\epsilon'_{11}\ R\ \epsilon'_{21}$, as needed. The trace distribution equivalence condition $\mathsf{tdist}(\epsilon'_1) = \mathsf{tdist}(\epsilon'_2)$ also holds since $\mathsf{tdist}(\epsilon_1) = \mathsf{tdist}(\epsilon_2)$. The summations in the step condition of Lemma 4.4 clearly hold.

   (b) $s.zval = \perp$. By Property 1 of R, for every $v \in \mathsf{supp}(\mathsf{lstate}(\epsilon_1)) \cup \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$, we have that $v.zval = \perp$. Following Property 2 of R, we distinguish two cases.

      i. $\mathsf{lstate}(\epsilon_1).yval$ is the Dirac distribution on $\perp$. In this case, T is enabled in every state in $\mathsf{supp}(\mathsf{lstate}(\epsilon_1))$.
      Let I be the singleton index $\{1\}$, let p be the Dirac measure on 1 and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2 = \epsilon_2$. We now show that $(\epsilon'_1, \epsilon_2) \in R$. By Definition 3.3 and by the effect of the unique action in T, we observe that $\mathsf{supp}(\mathsf{lstate}(\epsilon'_1))$ is equal to $\mathsf{supp}(\mathsf{lstate}(\epsilon_1))$, except that $\mathsf{supp}(\mathsf{lstate}(\epsilon'_1))$ projected on $yval$ is now the uniform distribution on $[n]$. As a result, Property 1 of R holds since the $zval$ component is not modified by the application of T to ϵ_1 and λ to ϵ_2, and Property 2b of R holds since the effect of the $choose$ action in Trapdoor is to select $yval$ according to a uniform distribution. The summations in the step condition clearly hold.

      ii. $\mathsf{lstate}(\epsilon_1).yval$ is uniformly distributed on $[n]$. In this case, T is disabled in every state in $\mathsf{supp}(\mathsf{lstate}(\epsilon_1))$, and the treatment is similar to that of Case 1a.

   In all cases, the only action that may take place is the choose action, and it is an internal action. So, the trace condition of R is trivially satisfied.

2. $T = \mathsf{Compute}$ and $\mathsf{Choose} \in \gamma_1$.

   Fix any pair of states $s \in \mathsf{supp}(\mathsf{lstate}(\epsilon_1))$ and $u \in \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$. Lemma 4.10 guarantees that $s.yval \neq \perp$. We consider two cases, according to the value of $s.zval$.

   (a) $s.zval \neq \perp$. Property 1 of R guarantees that $v.zval \neq \perp$ for every v in $\mathsf{supp}(\mathsf{lstate}(\epsilon_1)) \cup \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$. As a result T is disabled in every state in $\mathsf{supp}(\mathsf{lstate}(\epsilon_1))$, and $\mathsf{c}(\gamma_1, T)$ is disabled in every state $\mathsf{supp}(\mathsf{lstate}(\epsilon_2))$. The treatment of this case is then the same as the one of Case 1a of the Choose task.

   (b) $s.zval = \perp$. By the same observation as in the previous paragraph, we obtain that T is enabled in every state in $\mathsf{supp}(\mathsf{lstate}(\epsilon_1))$, and $\mathsf{c}(\gamma, T)$ is enabled every state in $\mathsf{supp}(\mathsf{lstate}(\epsilon_2))$. Also, using Property 2 of R, we obtain that the projection of $\mathsf{lstate}(\epsilon_1)$ on $yval$ is the uniform distribution on $[n]$.
   We define the probability measures needed to show the step correspondence. Let p be the uniform probability measure on the index set $I = \{1 \cdots n\}$. That is, $p(j) = \frac{1}{n}$ for each $j \in I$. For each j, we define the probability measure ϵ'_{1j} as follows. The support $\mathsf{supp}(\epsilon'_{1j})$ is the set of executions

$\alpha \in \mathsf{supp}(\epsilon'_1)$ such that $\mathsf{lstate}(\alpha).zval = j$. For each $\alpha \in \mathsf{supp}(\epsilon'_{1j})$ of the form $\alpha' \,\mathsf{compute}\, q$, let $\epsilon'_{1j}(\alpha) = \epsilon_1(\alpha')$. We define ϵ'_{2j} analogously from ϵ'_2, except that we now consider $\alpha \in \mathsf{supp}(\epsilon'_{2j})$ of the form $\alpha' \,\mathsf{choose}\, q$.

Now fix $j \in I$; we show that $(\epsilon'_{1j}, \epsilon'_{2j}) \in R$. To do this, we establish the properties of R for ϵ'_{1j} and ϵ'_{2j}, and show trace distribution equivalence for ϵ'_{1j} and ϵ'_{2j}.

Consider any pair of states $s' \in \mathsf{supp}(\mathsf{lstate}(\epsilon'_{1j}))$ and $u' \in \mathsf{supp}(\mathsf{lstate}(\epsilon'_{2j}))$. By definition of ϵ'_{1j} and ϵ'_{1j}, we know that Property 1 of R holds. Also, since $u'.zval \neq \bot$, Property 2 of R holds too. Since choose and compute are internal actions, trace distribution equivalence holds.

Since the projection of $\mathsf{lstate}(\epsilon_1)$ on $yval$ is the uniform distribution on $[n]$, since f is a permutation, and since the effect of the choose action of Rand is to select $zval$ according to the uniform distribution on $[n]$, the summations of the step condition of Lemma 4.4 hold.

3. $T = \mathsf{Compute}$ and $\mathsf{Choose} \notin \gamma_1$.

   Fix any pair of states $s \in \mathsf{supp}(\mathsf{lstate}(\epsilon_1))$ and $u \in \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$. Lemma 4.10 guarantees that $s.yval = \bot$. As a result, T is disabled in all states in $\mathsf{supp}(\mathsf{lstate}(\epsilon_1))$. Since $\mathsf{c}(\gamma_1, T) = \lambda$, the treatment of this case is then the same as the one of Case 1a of the Choose task.

4. $T = \mathsf{Report}$. Fix any pair of states $s \in \mathsf{supp}(\mathsf{lstate}(\epsilon_1))$ and $u \in \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$. We consider two cases, according to the value of $s.zval$.

   (a) $s.zval \neq \bot$. In this case, Property 1 guarantees that the projection of $\mathsf{supp}(\mathsf{lstate}(\epsilon_1)) \cup \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$ on $zval$ is a unique value j. That is, for all $v \in \mathsf{supp}(\mathsf{lstate}(\epsilon_1)) \cup \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$ we have $v.zval = j$. As a result, there is a unique action $a = report(j) \in T \cup \mathsf{c}(\gamma_1, T)$ that is enabled in all states in $\mathsf{supp}(\mathsf{lstate}(\epsilon_1)) \cup \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$.

   Let I be the singleton index $\{1\}$, let p be the Dirac measure on 1 and let $\epsilon'_{11} = \epsilon'_1$ and $\epsilon'_{21} = \epsilon'_2$. We show that $(\epsilon'_1, \epsilon'_2) \in R$. First, since the action a has no effect, Properties 1 and 2 of R trivially hold for $(\epsilon'_1, \epsilon'_2)$. Then, since the same action a is executed in both systems, trace distribution equivalence for ϵ'_1 and ϵ'_2 holds.

   (b) $s.zval = \bot$. In this case, Property 1 guarantees that the projection of $\mathsf{supp}(\mathsf{lstate}(\epsilon_1)) \cup \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$ on $zval$ is equal to $\bot$. Then, T and $\mathsf{c}(\gamma_1, T)$ are disabled in every state in $\mathsf{supp}(\mathsf{lstate}(\epsilon_1)) \cup \mathsf{supp}(\mathsf{lstate}(\epsilon_2))$, and the treatment of this case is the same as that of Case 1a of the Choose task.

□

5 Application to Security Protocols

In [CCK+05b, CCK+06c], we use the task-PIOAs to model and analyze the Oblivious Transfer (OT) protocol of Goldreich et al. [GMW87]. Our analysis takes into account computational issues such as resource-bounded adversaries and computational indistinguishability assumptions. Therefore, it requires extensions to the task-PIOA theory presented in this paper. These extensions, and the full details of our analysis, are beyond the scope of this paper. Our purpose in this section is to discuss how task-PIOAs can be used in modeling and analysis of cryptographic protocols based on a concrete, nontrivial example.

Oblivious Transfer In the OT problem, two input bits (x_0, x_1) are submitted to a Transmitter Trans and a single input bit i to a Receiver Rec. After engaging in an OT protocol, Rec should output only the single bit x_i. Rec should not learn the other bit x_{1-i}, and Trans should not learn i; moreover, an eavesdropping adversary should not, by observing the protocol messages, be able to learn anything about the inputs or the progress of the protocol. OT has been shown to be "complete" for multi-party secure computation, in the

sense that, using OT as the only cryptographic primitive, one can construct protocols that securely realize any functionality.

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

Our Analysis Our analysis follows the *trusted party* paradigm of [GMW87], with a formalization of secure emulation that is close in spirit to [PW00, Can01]. Our modeling involves two systems of automata:

- The *real system (RS)* consists of two automata representing protocol parties (Trans and Rec) and one automaton representing an adversarial communication service (Adv), which has access to all messages sent during the execution of the protocol. In RS, typical tasks include "choose random (y_0, y_1)", "send round 1 message", and "deliver round 1 message", as well as arbitrary tasks of environment and adversary automata. (The environment and adversary automata are purposely under-specified, so that our results are as general as possible.) Note that these tasks do not specify exactly what transition occurs. For example, the "choose" task does not specify the chosen values of (y_0, y_1). And the "send" task does not specify the message contents—these are computed by Trans, based on its own internal state.
- The *ideal system (IS)* consists of an ideal *Oblivious Transfer functionality* automaton, and a *simulator* automaton, which interacts with the functionality and tries to mimic the behavior of the real system such as the messages between the protocol parties.

Then we prove that RS implements IS. The proof consists of four cases, depending on which parties are corrupted.[6] In the two cases where Trans is corrupted, we can show that RS implements IS perfectly, using $\leq_0$. In the cases where Trans is not corrupted, we can show implementation only in a "computational" sense, namely, (i) for resource-bounded adversaries, (ii) up to negligible difference in probabilities, and (iii) under computational indistinguishability assumptions. Modeling these aspects requires additions to the task-PIOA framework of this paper, namely, defining a *time-bounded* version of task-PIOAs, and defining a variation, $\leq_{neg,pt}$, of the $\leq_0$ relation, which describes approximate implementation with respect to polynomial-time environments. Similar relations were defined in [LMMS98, PW01]. Our simulation relations (extended with time bounds on schedule lengths) are also shown to be sound for proving $\leq_{neg,pt}$.

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

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

Each of our system models, at each level, includes an explicit adversary component automaton, which acts as a message delivery service that can eavesdrop on communications and control the order of message

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

delivery. The behavior of this adversary is arbitrary, subject to polynomial time constraints on its computational capabilities. In our models, the adversary is the same at all levels, so our simulation relations relate the adversary states at consecutive levels directly, using the identity function. This treatment allows us to consider arbitrary adversaries without examining their structure in detail (they can do anything, but must do the same thing at all levels).

The following patterns arise in our simulation relation proofs. They are the motivations for our new definition of simulation relations, which has the expansion capability and relates measures to measures.

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

2. The Trapdoor vs. Rand example described in Section 4 occurs in our OT proof. Here, the low-level system chooses a random y and then computes $z = f(y)$ using a trap-door permutation f. The higher level system simply chooses the value of z randomly, without using value y or permutation f. In order to correspond the steps that define z in both automata, we need to correspond the point between random choice of y and the computation of z in Trapdoor to the initial state of Rand. This correspondence relates measures to measures and uses expansion.

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

6 Local Schedulers

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

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

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

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

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

We next define composition for probabilistic systems. This definition requires the local scheduler of a composition to be of the form $S_1 \| S_2$, where S_1 and S_2 are local schedulers of its components. That is, we require a local scheduler for a composition to project onto local schedulers of the individual components.

Without this condition, a choice made in one component might depend on the state of the other component, and this would prevent us from proving compositionality properties such as Theorem 6.9. An example that demonstrates how compositionality fails in the absence of such a condition can be found in [LSV07].

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

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

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

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

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

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

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

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

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

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

Proof. Let $\mathcal{T}_4 = (\mathcal{P}_4, R_4)$ be any environment (action-deterministic) task-PIOA for both $\mathcal{M}_1 \| \mathcal{M}_3$ and $\mathcal{M}_2 \| \mathcal{M}_3$. Let $\mathcal{M}_4$ be the trivial probabilistic system $(\mathcal{T}_4, \{\mathcal{T}_4\})$. Fix any task schedule γ_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 γ_1 and $\mathcal{P}'_{13}$. It suffices to show that τ is also generated by some task schedule γ_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 γ_1 is also a task schedule for $\mathcal{T}_1 \| (\mathcal{T}_3 \| \mathcal{T}_4)$. Since $\mathcal{P}'_{13}$ is a local scheduler of $\mathcal{M}_1 \| \mathcal{M}_3$, it is (by definition) of the form $\mathcal{P}'_1 \| \mathcal{P}'_3$, where $\mathcal{P}'_1 \in \mathcal{S}_1$ and $\mathcal{P}'_3 \in \mathcal{S}_3$. Let $\mathcal{P}'_{34} = \mathcal{P}'_3 \| \mathcal{P}_4$. Then $\mathcal{P}'_{34}$ is a local scheduler of $\mathcal{M}_3 \| \mathcal{M}_4$. Then, γ_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 γ_2 for $\mathcal{T}_2 \| \mathcal{T}_5$ and a local scheduler $\mathcal{P}'_2$ for $\mathcal{P}_2$ such that γ_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 γ_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 γ_2, $\mathcal{P}'_{23}$ and $\mathcal{P}'_4$ also generate τ, which suffices to show the required implementation relationship. □

7 Related Work

There are numerous models in the literature that combine nondeterministic and probabilistic choices (see [SdV04] for a survey). However, very few tackle the issue of partial-information scheduling, as we do. Exceptions include [dA99], which uses partitions on the state space to obtain partial-information schedules, and [CH05], which models local-oblivious scheduling. The former is essentially within the framework of *partially observable MDPs (POMDPs)*, originally studied in the context of reinforcement learning [KLC98]. In these frameworks, basic units of scheduling are individual actions, as opposed to equivalent classes of actions. That allows the adversary some access to random choices generated during execution, therefore these frameworks are not suitable for cryptographic modeling.

Our general approach to cryptographic protocol verification is inspired by the Interactive Turing Machine (ITM) framework used in [Can01]. There, protocol participants are modeled as ITMs and messages as bit strings written on input and output tapes. ITMs are purely probabilistic and are activated sequentially via message deliveries. This framework is well suited for computational analysis: a typical argument reduces the correctness of a protocol to the computational hardness (or indistinguishability) assumptions associated with the underlying cryptographic primitives. However, a complete and rigorous analysis is often impractical, because it involves too many low-level machine details. Indeed, in the community of computational cryptography, protocols are typically described using an informal high-level language, and proof sketches are given in terms of these informal descriptions.

Thus, our motivation is to provide a framework in which protocols can be specified clearly and concisely, and computational proofs can be carried out formally and at a high level of abstraction. This motivation is certainly not new: several other research groups have taken conventional concurrency modeling frameworks and added features intended for computational cryptographic analysis. This includes the *Probabilistic Polynomial-time process Calculus (PPC)* framework of Mitchell et al. [LMMS98, MMS03, MRST06] and the *Reactive Simulatability (RSIM)* framework of Pfitzmann et al. [PW00, PW01, BPW07]. However, the semantic foundations of concurrent computation used in these frameworks differ from task-PIOAs in some fundamental ways. These differences arise in part because we aim to exploit the benefits of nondeterminism to a greater extent. Below we give a brief summary of these two frameworks and compare them against task-PIOAs.

PPC On a conceptual level, the PPC and task-PIOA frameworks handle concurrency and nondeterminism in a similar way. Processes with nondeterministic choices are definable in both frameworks, and a global scheduler is used to resolve nondeterminism.[7] In [MRST06], the scheduler is a probabilistic polynomial-time function that selects an action label from the set of enabled actions. Typically, action labels in PPC correspond to the types of protocol messages, as opposed to the messages themselves. This is similar to our distinction between tasks and actions.

Despite these similarities, a major difference exists between the operational semantics of PPC and our task-scheduling mechanism. Our task schedules are oblivious sequences of tasks. Since tasks may represent either internal computations or external communications, a task schedule resolves choices involving both types of activities with no priority restrictions. In PPC, however, internal computations are prioritized over secure communications, which are in turn prioritized over insecure communications. The probabilistic polynomial-time scheduler is invoked when no more internal computations are possible.

Like our task schedules, the scheduler function of PPC is an entity distinct from the adversary. To faithfully model adversarial scheduling, one must construct the PPC adversary contexts in such a way that message timing is controlled by the adversary and not by the scheduler function. This point is observed in [MMS03] and it corresponds to our proposal that high-level nondeterminism is resolved by an adversary automaton. Thus, the real difference between PPC's scheduling mechanism and our own lies in the handling of low-level nondeterminism. In our view, task-based scheduling is a simple and sufficient method for that purpose.

[7] The authors have also developed a sequential version of PPC [DKMR05], with a semantics akin to the ITM and RSIM frameworks.

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

In PPC, symmetric semantic relations are sufficient because process expressions typically do not contain low-level nondeterminism. For example, under the grammar of PPC, there is no natural way to express a process P that performs actions a and b in either order and then proceeds as process P'. This is because PPC allows nondeterministic choices only via parallel composition, but one cannot pretend $a \| b$ to a separate process P'. In contrast, this type of choices is very common in our models (cf. our OT model in [CCK$^+$05b]). Using a simulation relation, we are able to handle the two cases (i.e., whether a precedes b or vice versa) separately but without duplicating any arguments involving P'.

To sum up, the PPC and task-PIOAs frameworks share similar goals but have different features and strengths. Having a process algebraic foundation, PPC proofs are based on formal deduction, hence very amenable to mechanization. However, process expressions constructed from a formal syntax are less flexible compared to our automata-based specifications. These differences are inherent and, because of them, one framework may be more appropriate than the other for a given application.

RSIM Next we consider the RSIM framework of Pfitzmann et al. There a protocol is modeled by a network of interrupt-driven probabilistic state machines, with special "buffer" machines to capture message delays, and special "clock ports" to control the scheduling of message delivery. Each individual machine is purely probabilistic; that is, it is fully-specified up to inputs and/or random choices generated during execution. In particular, if a system of machines is closed (i.e., every input of every machine is provided by some machine in the system), it is fully specified up to coin tosses.

For a closed RSIM system, a sequential activation scheme is used to define a unique probabilistic run for each possible initial state of the system. Under this scheme, machines are switched one at a time, and the current active machine M_1 selects the next machine M_2 by scheduling a message intended for M_2. (This is possible only when M_1 "owns" the clock port to an input buffer of M_2.) If no machines are active, a special "master scheduler" machine is triggered by default.

Thus, in order to capture nondeterministic choices within the RSIM framework, one must associate explicit inputs to each schedulable event and then quantify over different machines that provide these scheduling inputs. That is, each scheduler machine corresponds to one particular schedule. In contrast, nondeterministic choices can be expressed in a task-PIOA without the use of explicit inputs, and we quantify over task schedules to capture the possible ways of resolving nondeterminism.

These technical differences are again the result of our divergent views on nondeterminism. The RSIM framework is designed with high-level nondeterminism in mind. Therefore, message buffering and delivery are made explicit, while internal computations are entirely encapsulated within single-step transitions. As in PPC, there is no natural way (aside from adding inputs and scheduler machines) to express nondeterministic choices that correspond to implementation freedom.

Aside from practical consequences, the different treatments of nondeterminism also affect the meaning of security definitions. For example, in the definition of reactive simulatability, the user and adversary are fixed *after* all other machines are determined. Essentially, this allows the worst possible adversary for every schedule of the system. In our security definitions [CCK$^+$06c, CCK$^+$05b], the environment[8] and adversary

[8]The role of "user" in RSIM is comparable to the role of "environment" in our definitions.

are fixed *before* the task schedules. Therefore, we consider instead the worst possible schedule for each given adversary. In light of the separation result in [CCLP07], we believe RSIM and task-PIOA definitions are *not* equivalent, because RSIM is based on sequential scheduling.

8 Conclusions

We have extended the traditional PIOA model with a task mechanism, which provides a systematic way of resolving nondeterministic scheduling choices without using dynamically generated information. In the resulting Task-PIOA framework, we develop basic machinery for verification, including a compositional trace-based semantics and a new kind of simulation relation that is sound for proving implementation. The utility of these tools are demonstrated in the Oblivious Transfer case study, which is outlined in this paper and presented in full in [CCK+05b]. We have also proposed a further extension, in which local nondeterminism are resolved by schedulers that use local information only.

Although our development is motivated by concerns in cryptographic protocol analysis, the notion of partial-information scheduling is interesting in its own right. For example, some distributed algorithms are designed using partial-information scheduling assumptions, because the problems they address are provably unsolvable under perfect-information scheduling [Cha96, Asp03]. Also, one may argue that partial-information scheduling is more realistic in models of large distributed systems, in which basic scheduling decisions are made locally, and not by any centralized mechanism.

There are many more interesting problems in our general project of cryptographic protocol verification. Below we discuss some of them.

Computational Assumptions We would like to express standard computational assumptions in terms of implementation relations between task-PIOAs. Judging from our results on hard-core predicates [CCK+05b], decisional assumptions can be formulated quite easily by comparing the accept probabilities of a distinguisher environment. To handle hardness assumptions, we need to formalize the usual "conversion" argument: any successful adversary can be converted into an efficient algorithm for solving a difficult problem.

Statistical Indistinguishability Another problem is to formalize the notion of statistical indistinguishability. This yields a third type of implementation relation, in addition to perfect implementation $\leq_0$ and the computational approximate implementation $\leq_{neg,pt}$. In [ML06], metric distances between trace distributions are used to define approximate implementations and approximate simulations for individual task-PIOAs. These definitions are extended to families of task-PIOAs, thereby obtaining a notion of statistical approximate implementation relation [CMP07].

Simulation-Based Security Definitions Our security definition falls under the category of simulation-based security. In [DKMR05], various simulation-based security notions, including universally composable security [Can01] and reactive simulatability [PW00, PW01], are translated into the sequential version of PPC. A semantic hierarchy of these notions is given, based on the placement of a master process and the ability to forward communication between processes. Interestingly, there is no obvious way to place our notion of security in this hierarchy. This is because there is no inherent notion of master process in our definitions. Moreover, our task schedules are determined *after* the adversary, simulator, and environment automata have been fixed. This additional quantification is not found in other security definitions. We investigate the role of scheduling in simulation-based security and show that different choices of scheduling mechanisms give rise to differences in the meanings of security definitions in [CCLP07].

Composition Theorem In [CCK+07], we prove that our notion of secure emulation is preserved under a polynomial number of protocol substitutions. While the result is standard, the actual proof in our framework requires a nontrivial adaptation of the usual hybrid argument. This reveals some interesting implications of

our definitions. In particular, we allow a time-bounded task-PIOA to continue its computation indefinitely, even though each individual transition is of bounded complexity. A task schedule thus takes on two roles: resolving nondeterminism and limiting the number of individual transitions. This separation between single step complexity and schedule length complexity is a design decision, because eventually we are interested in proving long-term correctness properties. As a result, our definition of secure emulation involves an additional quantification over schedule length bounds, which makes the hybrid argument more delicate than usual.

Mechanized Proofs The simulation proofs in our OT case study are quite similar in character to typical analysis of distributed algorithms. Namely, we establish correspondence between two specifications by examining their respective behaviors at comparable stages in the protocol. Invariant-style properties are often invoked to prove these correspondences. This type of reasoning is a good candidate for mechanization. In fact, we are quite confident that the (non-probabilistic) invariant properties in our proofs can be proven in a straightforward fashion using a theorem prover such as PVS. Since our language is based on I/O automata, translation into PVS (without probabilities) can be done systematically using the TEMPO Toolset [Too] and the TAME interface [Arc]. It will be very interesting to explore the possibility of mechanizing a greater portion of our simulation proofs. This may involve the development of new reasoning techniques to handle sets of traces (as opposed to individual traces or trace distributions).

Acknowledgments

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

Canetti's work on this project was supported by NSF CyberTrust Grant #0430450. Cheung was supported by DFG/NWO bilateral cooperation project 600.050.011.01 Validation of Stochastic Systems (VOSS) and by NSF Award #CCR-0326277. Part of this work was done when she was at Radboud University Nijmegen. Kaynar and Lynch were supported by DARPA/AFOSR MURI Award #F49620-02-1-0325, MURI AFOSR Award #SA2796PO 1-0000243658, NSF Awards #CCR-0326277 and #CCR-0121277, and USAF, AFRL Award #FA9550-04-1-0121, and Kaynar was supported by US Army Research Office grant #DAAD19-01-1-0485. Pereira was supported by the Belgian National Fund for Scientific Research (F.R.S.-FNRS). Segala was partially supported by the European Project CON4COORD (C4C; contract number 223844) and the Italian PRIN Project SOFT.

References

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

[Arc] M. Archer. TAME support for refinement proofs. Available at `http://chacs.nrl.navy.mil/personnel/archer.html`.

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

[Bea91] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

[BK98] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

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

[BPW07] M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, December 2007.

[Can95] R. Canetti. *Studies in Secure Multi-Party Computation and Applications*. PhD thesis, Weizmann Institute, Israel, 1995.

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

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

[CCK+05b] R. Canetti, L. Cheung, D. Kaynar, M.Liskov, N.Lynch, O. Pereira, and R. Segala. Using task-structured probabilistic I/O automata to analyze an oblivious transfer protocol. Cryptology ePrint Archive, Report 2005/452, 2005. Available at `http://eprint.iacr.org/`.

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

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

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

[CCK+07] R. Canetti, L. Cheung, D. Kaynar, N. Lynch, and Olivier Pereira. Compositional security for Task-PIOAs. In *Proceedings of the 20th IEEE Computer Security Foundations Symposium (CSF-20)*, pages 125–139. IEEE Computer Society Press, July 2007.

[CCLP07] R. Canetti, L. Cheung, N. Lynch, and O. Pereira. On the role of scheduling in simulation-based security. In R. Focardi, editor, *Proceedings of the 7th International Workshop on Issues in the Theory of Security (WITS '07)*, pages 22–37, March 2007. Available at `http://eprint.iacr.org/2007/102`.

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

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

[CLK+06] R. Canetti, L.Cheung, D. Kaynar, M. Liskov, N. Lynch, O. Pereira, and R. Segala. Using task-structured probabilistic I/O automata to analyze cryptographic protocol. In V. Cortier and S. Kremer, editors, *Proceedings of Workshop on Formal and Computational Cryptography (FCC '06)*, pages 34–39, 2006.

[CLSV06] L. Cheung, N. Lynch, R. Segala, and F. Vaandrager. Switched PIOA: Parallel composition via distributed scheduling. *Theoretical Computer Science*, 365(1–2):83–108, 2006.

[CMP07] Ling Cheung, Sayan Mitra, and Olivier Pereira. Verifying statistical zero knowledge with approximate implementations. Cryptology ePrint Archive, Report 2007/195, 2007. Available at `http://eprint.iacr.org/2007/195`.

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

[Der70] C. Derman. *Finite State Markovian Decision Processes*. Academic Press, 1970.

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

[GL90] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology - Crypto '90*, pages 77–93, Berlin, 1990. Springer-Verlag. Lecture Notes in Computer Science Volume 537.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing (STOC'85)*, pages 291–304, 1985.

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

[GPS06] Marc Girault, Guillaume Poupard, and Jacques Stern. On the fly authentication and signature schemes based on groups of unknown order. *Journal of Cryptology*, 19(4):463—487, 2006.

[JL91] B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the 6th IEEE Symposium on Logic in Computer Science*, pages 266–277, 1991.

[Kan58] L. Kantorovitch. On the translocation of masses. *Management Science*, 5(1):1–4, 1958.

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

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

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

[LSV03] N.A. Lynch, R. Segala, and F.W. Vaandrager. Compositionality for probabilistic automata. In *Proc. 14th International Conference on Concurrency Theory (CONCUR 2003)*, volume 2761 of *LNCS*, pages 208–221. Springer-Verlag, 2003.

[LSV07] N.A. Lynch, R. Segala, and F.W. Vaandrager. Observing branching structure through probabilistic contexts. *SIAM Journal on Computing*, 37:977–1013, 2007.

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

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

[ML07] Sayan Mitra and Nancy Lynch. Approximate implementation relations for probabilistic I/O automata. *Electr. Notes Theor. Comput. Sci.*, 174(8):71–93, 2007.

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

[MR91] S. Micali and P. Rogaway. Secure computation. In Joan Feigenbaum, editor, *Advances in Cryptology - Crypto '91*, pages 392–404, Berlin, 1991. Springer-Verlag. Lecture Notes in Computer Science Volume 576.

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

[NMO08] Waka Nagao, Yoshifumi Manabe, and Tatsuaki Okamoto. Relationship of three cryptographic channels in the uc framework. In *Provable Security, Second International Conference, ProvSec 2008*, volume 5324 of *LNCS*, pages 268–282. Springer, 2008.

[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.

[PW94] Birgit Pfitzmann and Michael Waidner. A general framework for formal notions of "secure" system. Technical report, Hildesheimer Informatik-Berichte 11/94, Institut fr Informatik, Universitt Hildesheim., April 1994.

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

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

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

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

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

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

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

[Too] TAME Tempo Toolset. Available at `http://www.veromodo.com/Veromodo/Tempo-Toolset.html`.