Department of Computer Science

LEARNING-BASED COMPOSITIONAL VERIFICATION FOR SYNCHRONOUS PROBABILISTIC SYSTEMS

Lu Feng
Tingting Han
Marta Kwiatkowska
David Parker

RR-11-05

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford OX1 3QD

Learning-based Compositional Verification for Synchronous Probabilistic Systems

Lu Feng, Tingting Han, Marta Kwiatkowska, and David Parker

Department of Computer Science, University of Oxford, Oxford, OX1 3QD, UK

Abstract. We present novel, fully-automated techniques for compositional verification of synchronous probabilistic systems. First, we give an assume-guarantee framework for verifying probabilistic safety properties of systems modelled as discrete-time Markov chains. Assumptions about system components are represented as probabilistic finite automata (PFAs) and the relationship between components and assumptions is captured by weak language inclusion. Then, we develop a fully-automated implementation of this framework, which includes a semi-algorithm to check language inclusion for PFAs and a new active learning method for PFAs, which we use to automatically generate assumptions. We present experimental results from a prototypical implementation of our approach.

1 Introduction

Many real-life systems exhibit stochastic behaviour, for example due to the possibility of failures, uncertainty with regards to timing or the use of randomisation. In recent years, there has been a great deal of interest in *formal verification* techniques that perform a rigorous and fully-automated analysis of quantitative properties of stochastic systems. *Probabilistic model checking* is one such technique, which has been successfully applied in many application domains, from randomised communication protocols to biological systems. It is based on the exhaustive construction of a probabilistic model, such as a Markov chain or Markov decision process, and its analysis against properties specified in probabilistic temporal logics. These logics can express, for example, "the probability of an airbag failing to deploy within 0.02 seconds is at most 0.0001".

Like most automated formal verification techniques, one of the principal challenges in probabilistic model checking is scalability. Models capturing the behaviour of complex systems comprising many interacting components are often huge. This motivates the development of *compositional* verification methods, which decompose the task of analysing a large system model into smaller sub-tasks that analyse each of its components separately. We focus on the *assume-guarantee* paradigm, in which each system component is analysed under an *assumption* about the other component(s) it is composed with. After checking that the assumption is indeed satisfied, proof rules are used to deduce properties of the overall system.

Several assume-guarantee frameworks for probabilistic systems have been proposed, mainly for probabilistic automata, which model both probabilistic and nondeterministic behaviour [1,18,11]. The key challenge when developing such a framework is formulating an appropriate notion of *assumptions*. Fundamentally, of course, these need to support compositional reasoning, but our goal is to develop assume-guarantee techniques that are practical, efficient and fully-automated. This means that assumptions should ideally: (i) be expressive enough for practical applications; (ii) allow efficient, fully-automated verification; (iii) be amenable to automatic generation.

One promising direction is the framework of [18] (and its extensions in [11,10]). In [18], assumptions are *probabilistic safety properties* (e.g. "event A always occurs before event B with probability at least 0.98") and [11] generalises this to boolean combinations of ω-regular and reward properties. In both cases, this yields efficiently checkable assumptions and the approaches were successfully implemented and applied to some large case studies. Furthermore, [10] showed how to *automatically* generate probabilistic safety property assumptions [18] using *learning* techniques based on L*.

In this work, we continue to develop probabilistic assume-guarantee techniques in which assumptions can be automatically generated via learning. In particular, our focus is on using a more expressive class of assumptions. Probabilistic safety properties [18] can only capture a limited amount of information about a system component, restricting the cases where assume-guarantee reasoning can be applied. The framework of [18] is *incomplete* in the sense that, if the property being verified is true, there does not necessarily exists an assumption that can be used to perform the verification compositionally.

This paper proposes novel techniques for compositional probabilistic verification in which assumptions are *probabilistic finite automata* (PFAs). Unlike [18,11], our approach *is* complete. Furthermore, in a similar fashion to [11], we are able to use learning to automatically generate PFAs representing assumptions. PFAs represent weighted languages, mapping finite words to probabilities. An assumption A about a system component M is represented by a PFA that gives bounds on the probabilities of traces being observed in M. This is an inherently *linear-time* relation, which is well-known to be difficult to adapt to compositional techniques for systems that exhibit both probabilistic and nondeterministic behaviour [22]. So, in the present work, we restrict our attention to *fully probabilistic* systems. We target systems in which several components, each exhibiting probabilistic behaviour, are composed in a *synchronous* fashion, resulting in a fully probabilistic model that can be captured as a discrete-time Markov chain (DTMC).

We model components as *probabilistic I/O systems* (PIOSs), which extend DTMCs with output actions and (nondeterministic) input actions. The relation between a PIOS M a PFA A representing an assumption about M is captured by *weak language inclusion*, which abstracts internal behaviour to produce smaller assumptions. Based on this, we propose an assume-guarantee framework for verifying probabilistic safety properties on DTMCs composed of PIOSs. We give an asymmetric proof rule for two-component systems and show how this allows verification to be performed compositionally using two separate tasks.

In order to implement our framework, we give an algorithm to check weak language inclusion, reducing it to the existing notion of (strong) *language inclusion* for PFAs. Although checking PFA language *equivalence* (that each word maps to the same probability) is decidable in polynomial time [24,8,15], language *inclusion* is undecidable [5,19]. We propose a semi-algorithm, inspired by [24], to check language inclusion; in the case where the check fails, a minimal counterexample is produced.

We also present techniques, based on *algorithmic learning*, to automatically generate assumptions for our framework. This direction of work is influenced by the successful learning-based techniques for (non-probabilistic) compositional model checking, as pioneered by Pasareanu, Giannakopoulou et al. [20]. They use the L* algorithm [2], which learns an unknown regular language, to generate assumptions. In the probabilistic setting, these ideas were then adapted in [10] to generate probabilistic safety property assumptions for the framework of [18].

In this paper, we develop a novel technique for learning PFAs, in the style of L*. It uses an *active learning* approach, posing *queries* in an interactive fashion to gather information about the PFA representing a probabilistic assumption that is to be learnt. This is based on so-called *white-box* learning, where all information about the system being learnt (in this case, the component for which an assumption is being generated) is fully available. We do not consider *black-box* techniques, for example based on statistical techniques, because they yield only approximate results; for compositional verification, assume-guarantee proof rules can only be applied if we have guaranteed results about components. Finally, we incorporate the learning-based assumption generation into a complete implementation of our assume-guarantee framework and illustrate its applicability to a selection of benchmark case studies.

In summary, the contributions of this paper are as follows:

(i) a fully-automated assume-guarantee framework for DTMCs, including automatic generation of probabilistic assumptions represented as PFAs;
(ii) a semi-algorithm for checking language inclusion on PFAs;
(iii) a new L*-style learning method for PFAs.

This paper is an extended version of [9], including additional details and examples, as well as proofs for the results stated in the paper. All proofs can be found in the appendix.

Related work. In addition to the probabilistic assume-guarantee techniques mentioned above [1,18,11,10], compositional techniques have also been proposed for model checking DTMC models of hardware systems, based on conditional probabilities [16]. On the topic of applying learning to compositional verification, we are only aware of our prior work [10] in the probabilistic setting, but there are several successful approaches for non-probabilistic assumption generation [20,6].

Regarding techniques for PFAs, we are not aware of any existing techniques to check language inclusion, but, as stated above, there are algorithms to check language equivalence [24,8,15]. There are several existing active learning techniques for PFAs, e.g. [13,23,4]. The technique of [13] only applies to PFAs representing stochastic languages (probability distributions over finite words), which cannot represent PIOSs due to the presence of nondeterminism over input actions. It also only learns PDFAs, a restricted class of PFAs where the model's underlying structure is a DFA. The approach of [23] learns a more general class of PFAs, but uses queries that require the size of the learnt PFA to be known in advance. In [4], the learnt model is in fact a *multiplicity automaton*, not a PFA, and the transition weights may have negative values.

2 Preliminaries

We will use $SDist(S)$ to denote the set of all discrete probability *sub-distributions* over a set S, η_s for the point distribution on $s \in S$, and $\mu_1 \times \mu_2 \in SDist(S_1 \times S_2)$ for the product distribution of $\mu_1 \in SDist(S_1)$ and $\mu_2 \in SDist(S_2)$.

2.1 Probabilistic Finite Automata

In this paper, we use *probabilistic finite automata* (PFAs), as proposed by Rabin [21] to define (non-probabilistic) languages. They are also known as *probabilistic automata*; however, we avoid this term to prevent confusion with the identically named model of Segala [22], used for modelling and verification of concurrent probabilistic systems.

Definition 1 (PFA). *A* probabilistic finite automaton *(PFA) is a tuple $A = (S, \overline{s}, \alpha, \mathbf{P})$, where S is a finite set of states, $\overline{s} \in S$ is an initial state, α is an alphabet and $\mathbf{P} : \alpha \to (S \times S \to [0,1])$ is a function mapping actions to transition probability matrices. For each $a \in \alpha$ and $s \in S$, $\sum_{s' \in S} \mathbf{P}(a)[s,s'] \in [0,1]$.*

A PFA A defines a mapping $Pr^A : \alpha^* \to [0,1]$ from finite words to probabilities. Here, and elsewhere in the paper, we assume that all probabilities are rational values. In some formulations of PFAs, the probabilities for all finite words sum to one, defining a *stochastic language*. In other cases, including our use of PFAs in this paper, this is not necessarily true. Intuitively, the probability $Pr^A(w)$ for a word $w = a_1 \cdots a_n \in \alpha^*$ is determined by tracing path(s) through A that correspond to w, with $\mathbf{P}(a)[s,s']$ giving the probability to move from state s to s' when reading symbol a. More precisely, we let: $\boldsymbol{\iota}$ be a 0-1 row vector indexed by states S with $\boldsymbol{\iota}[s]$ equal to 1 if and only if $s = \overline{s}$; $\boldsymbol{\kappa}$ be a column vector over S containing all 1s; and $\mathbf{P}(w) = \mathbf{P}(a_1)\mathbf{P}(a_2)\cdots\mathbf{P}(a_n)$. Then, we define $Pr^A(w) = \boldsymbol{\iota}\mathbf{P}(w)\boldsymbol{\kappa}$ as the probability that A accepts w.

Our definition of PFAs is slightly non-standard in several respects. Firstly, the classical definition [21] assumes a set of *accepting states*; we omit this, effectively making all states accepting (which is why the vector $\boldsymbol{\kappa}$ contains all 1s). We adopt this definition because we restrict our attention to PFAs that correspond to executions of probabilistic models. Secondly, we allow rows of the matrices $\mathbf{P}(a)$ to sum to less than 1. We can always translate a PFA defined in the manner above to the classical definition by adding a (non-accepting) sink state and additional incoming transitions.

We will require the following relations between PFAs.

Definition 2 (Language inclusion/equivalence). *Given two PFAs A_1 and A_2 with the same alphabet α, we say A_1 and A_2 are related by (strong)* language inclusion *(resp.* language equivalence*), denoted $A_1 \sqsubseteq A_2$ (resp. $A_1 \equiv A_2$), if for every word $w \in \alpha^*$, $Pr^{A_1}(w) \leqslant Pr^{A_2}(w)$ (resp. $Pr^{A_1}(w) = Pr^{A_2}(w)$).*

2.2 Discrete-time Markov Chains

We focus on fully probabilistic systems, modelled as *discrete-time Markov chains*.

Definition 3 (DTMC). *A* discrete-time Markov chain *(DTMC) is a tuple $D{=}(S, \overline{s}, \alpha, \delta)$, where S is a finite set of states, $\overline{s} \in S$ is an initial state, α is an alphabet of* action labels *and $\delta : S \times (\alpha \cup \{\tau\}) \to SDist(S)$ is a (partial)* probabilistic transition function*, such that, for any s, $\delta(s,a)$ is defined for at most one $a \in \alpha \cup \{\tau\}$.*

The behaviour of a DTMC D in each state s is represented by the function δ. If $\delta(s,a) = \mu$, then the DTMC makes a transition, labelled with action a, and moves to each state s' with probability $\mu(s')$. We write $s \xrightarrow{a} \mu$ to indicate the existence of such a transition. Note that μ can be a sub-distribution, in which case the DTMC deadlocks with probability $1-\sum_{s\in S}\mu(s)$. The DTMC also deadlocks in the case where $\delta(s,a)$ is not defined for any a, which we denote by $s \not\rightarrow$. Following the common convention, action label τ denotes a "silent' (or "internal") transition.

A (finite or infinite) path through D, which represents one possible execution of the system that it models, is a sequence of transitions $\theta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ where $s_0 = \overline{s}$ and $\delta(s_i, a_i)(s_{i+1}) > 0$ for all $i \geq 0$. We use $tr(\theta) = a_0 a_1 \cdots$ to denote the word (or trace) of path θ and $Paths^D$ to denote the set of all D's (infinite) paths.[1] In standard fashion [14], we can define a probability space Pr^D over the set of paths $Paths^D$, which allows us to reason about the probability of certain events occurring.

The class of properties that we will verify on DTMCs in this paper are *probabilistic safety properties*. These can express, for example "the probability that no error occurs is at least 0.99" or "the probability that event A always precedes event B is at least 0.75". More formally, a probabilistic safety property takes the form $\langle G\rangle_{\geqslant p}$, where G is a *regular safety property* [3], defining a set of "good" executions, and $p \in [0,1]$ is a probability bound. We write $Pr^D(G)$ for the probability of a "good" execution occurring in D and say that the property is satisfied, denoted $D \models \langle G\rangle_{\geqslant p}$, if $Pr^D(G) \geqslant p$. In practice, we represent G by an *error automaton* G^{err}, a deterministic finite automaton (DFA) storing all prefixes of executions that do *not* satisfy G. Model checking $\langle G\rangle_{\geqslant p}$, i.e. computing $Pr^D(G)$, reduces to building the product $D\otimes G^{err}$ and solving a linear equation system [3].

3 Assume-Guarantee for Synchronous Probabilistic Systems

3.1 Probabilistic I/O Systems

We now define a compositional verification framework for synchronous probabilistic systems. Since these systems are fully probabilistic, they will be modelled as discrete-time Markov chains (DTMCs). The individual components, however, when considered in isolation, are nondeterministic in nature since they can respond to inputs from other components. We will model these components as *probabilistic I/O systems* (PIOSs).

Definition 4 (PIOS). *A* probabilistic I/O system (PIOS) *is a tuple* $M = (S, \overline{s}, \alpha, \delta)$, *where* S *and* $\overline{s}$ *are as for DTMCs, and the alphabet* α *and transition function* $\delta : S \times (\alpha \cup \{\tau\}) \to SDist(S)$ *satisfy the following two conditions:*

- α *is partitioned into three disjoint sets of* input, output *and* hidden *actions, which we denote* α^I, α^O *and* α^H*, respectively; input actions* α^I *are further partitioned into* m *disjoint* bundles $\alpha^{I,i}$ *(*$1 \leqslant i \leqslant m$*) for some* m*;*
- *the set* $enab(s) \subseteq \alpha \cup \{\tau\}$ *of* enabled *actions for each state* s *(i.e. the actions* a *for which* $\delta(s,a)$ *is defined) satisfies either* $|enab(s)| = 1$ *if* $enab(s) \in \alpha^O \cup \alpha^H \cup \{\tau\}$ *or* $enab(s) = \alpha^{I,i}$ *for some input action bundle* $\alpha^{I,i}$.

[1] This includes "infinite" paths which reach a deadlock and remain there forever.

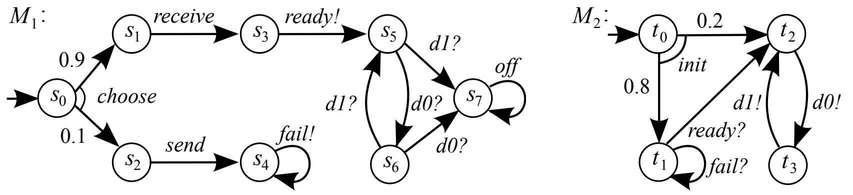


Fig. 1. Running example: two PIOSs M_1 and M_2.

PIOSs generalise DTMCs by distinguishing between different types of actions and by allowing multiple input actions to be enabled from the same state. From any state s of a PIOS, there is either a single transition labelled with an output, hidden or τ action, or k transitions, each labelled with one action from a particular bundle $\alpha^{I,i}$ comprising k input actions. In practice, a bundle $\alpha^{I,i}$ will typically represent the set of of possible values that can be communicated from one component to another. A PIOS with only hidden or τ actions can be considered to be a DTMC. As we will see below, the intention is that the parallel composition of PIOSs results in a DTMC.

Transitions and paths in a PIOS are defined as for DTMCs. However, we will sometimes write transitions labelled with action a as $s \xrightarrow{a!} \mu$ or $s \xrightarrow{a?} \mu$, to indicate that a is an output or input action, respectively. Typically, to reason about the probability of events in models that exhibit both probabilistic and nondeterministic behaviour, we need a notion of *schedulers* (or *adversaries*) that resolve nondeterminism. For PIOSs, however, we only consider the (maximum) probability $Pr^M(w)$ of a particular sequence of actions $w \in (\alpha \cup \{\tau\})^*$ being observed, which simplifies this.[2] The probability of a finite path $\theta = s_0 \xrightarrow{a_0} s_1 \cdots \xrightarrow{a_{n-1}} s_n$ in M is given by $Pr^M(\theta) = \prod_{i=0}^{n-1} \delta(s_i, a_i)(s_{i+1})$. Since PIOSs only have nondeterminism on input actions, the probability for a word $w \in (\alpha \cup \{\tau\})^*$ is well defined: letting $wd(\theta)$ denote the word $a_0 \ldots a_{n-1}$ of actions from path θ, we have $Pr^M(w) = \sum_{wd(\theta)=w} Pr^M(\theta)$. Then, letting $st : (\alpha \cup \{\tau\})^*\alpha \to \alpha^*$ be the function that removes all τs, we define the probability $Pr^M_\tau(w')$ for a τ-free word $w' \in \alpha^*$ as $Pr^M_\tau(w) = \sum_{w=st(w')} Pr^M(w')$.

Example 1. Fig. 1 depicts two PIOSs M_1 and M_2. M_1 is a data communicator which chooses (probabilistically) to either send or receive data. This simple example only models receiving; choosing to send results in a failure. M_1 tells M_2, a data generator, that it is ready to receive using action *ready*. M_2 should then send a sequence of packets, modelled by the alternating actions *d0* and *d1*. If M_1 has failed, it sends a message *fail*. M_2 also has an initialisation step (*init*), which can fail. With probability 0.8, it is ready to receive signals from M_1; otherwise, it just tries to send packets anyway. Input/output actions for M_1, M_2 are labelled with ?/! in the figure; all other actions are hidden. Each PIOS has a single input action bundle: $\alpha_1^{I,1} = \{d0, d1\}$, $\alpha_2^{I,1} = \{ready, fail\}$.

Parallel composition. Next, we define parallel composition of PIOSs. For simplicity, we only present a binary parallel composition operator that results in a DTMC (since this matches the proof rule that we will later define). However, this can be adapted to more flexible schemes composing multiple PIOSs.

[2] In fact, following the sequence of actions w dictates a unique scheduler.

Given two PIOSs M_1, M_2 with alphabets α_1, α_2, we say that M_1 and M_2 are *composable* if $\alpha_1^I = \alpha_2^O$, $\alpha_1^O = \alpha_2^I$ and $\alpha_1^H \cap \alpha_2^H = \varnothing$; and we call $\alpha_1^I \cup \alpha_1^O$ the *external actions*. We distinguish the following cases: (1) if both actions are external actions and they are matching, then they will be performed simultaneously; (2) if both actions are hidden or τ actions, say $b_1 \in \alpha_1^H \cup \{\tau\}, b_2 \in \alpha_2^H \cup \{\tau\}$, then they will be carried out by a combined action $b_1 * b_2$ simultaneously; (3) if a is an external action (or no actions are enabled) and b is an hidden action, then a will wait (or equivalently perform an idle action $\perp$) and a combined action $\perp * b$ will be executed. Formally:

Definition 5 (Parallel composition). *The* parallel composition *of two composable PIOSs $M_i = (S_i, \overline{s}_i, \alpha_i, \delta_i)$ for i=1, 2 is given by the PIOS $M_1||M_2 = (S_1 \times S_2, (\overline{s}_1, \overline{s}_2), \alpha, \delta)$, where $\alpha = \alpha^H = \alpha_1^I \cup \alpha_1^O \cup \big((\alpha_1^H \cup \{\perp\}) * (\alpha_2^H \cup \{\perp\})\big)$ and, for $b_1 \in \alpha_1^H \cup \tau$, $b_2 \in \alpha_2^H \cup \tau$ and $a \in \alpha_1^I \cup \alpha_1^O$, δ is defined such that $(s_1, s_2) \xrightarrow{\gamma} \mu_1 \times \mu_2$ iff one of the following holds:*

- $s_1 \xrightarrow{a} \mu_1$, $s_2 \xrightarrow{a} \mu_2$, *and* $\gamma = a$
- $s_1 \xrightarrow{b_1} \mu_1$, $s_2 \xrightarrow{b_2} \mu_2$ *and* $\gamma = b_1 * b_2$
- $s_1 \xrightarrow{b_1} \mu_1$, $s_2 \xrightarrow{a}$ *(or $s_2 \not\rightarrow$)*, $\mu_2 = \eta_{s_2}$, *and* $\gamma = b_1 * \perp$
- $s_1 \xrightarrow{a}$ *(or $s_1 \not\rightarrow$)*, $s_2 \xrightarrow{b_2} \mu_2$, $\mu_1 = \eta_{s_1}$, *and* $\gamma = \perp * b_2$

Note that $(\alpha_1^H \cup \{\perp\}) * (\alpha_2^H \cup \{\perp\})$ is defined in a point-wise fashion of $*$ operator. It is straightforward to check that PIOS $M_1 \| M_2$ is well-defined and a DTMC.

3.2 Assumptions for PIOSs

We will use PIOSs to develop an assume-guarantee framework for compositional verification. First, we need to formulate a notion of *assumptions* about PIOSs. For this, we will use a specific class of PFAs that satisfy certain additional conditions.

Definition 6 (Assumption). *Let M be a PIOS with alphabet $\alpha = \alpha^I \uplus \alpha^O \uplus \alpha^H$ and input action bundles $\alpha^I = \biguplus_{i=1}^m \alpha^{I,i}$. An* assumption *$A$ about M is a PFA $A = (S, \overline{s}, \alpha, \mathbf{P})$ satisfying, for each state $s \in S$: (i) either all or none of the actions in a bundle $\alpha^{I,i}$ $(1 \leqslant i \leqslant m)$ are enabled in s; (ii) $p^{\max}(s) \in [0, 1]$, where:*

$$p^{\max}(s) \overset{\text{def}}{=} \sum_{a \in \alpha^O \cup \alpha^H} \sum_{s' \in S} \mathbf{P}(a)[s, s'] + \sum_{i=1}^{m} p_i^{\max}(s) \text{ and } p_i^{\max}(s) \overset{\text{def}}{=} \max_{a \in \alpha^{I,i}} \sum_{s' \in S} \mathbf{P}(a)[s, s']$$

Intuitively, an assumption is a PFA in which, for any state, the sum of all its outgoing transitions by output or hidden actions and the *maximum* outgoing probability sums among all the actions in each bundle is at most 1. For simplicity, in this presentation, we assume that assumption A contains *all* actions (except τ) from PIOS M. As we will describe below, we first rename all actions in α^H to τ, meaning that A in fact only refers to actions from $\alpha^I \cup \alpha^O$.

To capture the fact that a PFA represents a *valid* assumption about a PIOS M, we introduce the notion of *weak language inclusion*, which relaxes the definition of language inclusion for PFAs introduced earlier by ignoring any occurrences of τ actions. We also define *weak language equivalence*.

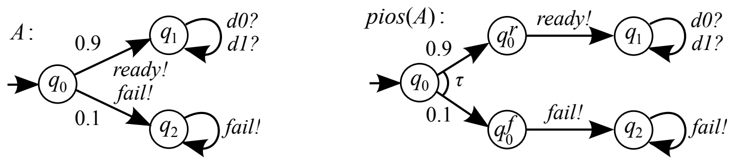


Fig. 2. Assumption A and its PIOS conversion $pios(A)$.

Definition 7 (Weak language inclusion/equivalence). *For PIOS M with alphabet α and an assumption A about M, we say that M and A are related by* weak language inclusion (resp. equivalence)*, denoted $M \sqsubseteq_w A$ (resp. $M \equiv_w A$), if for every word $w \in \alpha^*$, $Pr_\tau^M(w) \leqslant Pr^A(w)$ (resp. $Pr_\tau^M(w) = Pr^A(w)$).*

A valid assumption A for M is one that satisfies $M \sqsubseteq_w A$. We can reduce the problem of checking whether this is true to the problem of checking (strong) language inclusion between two PFAs, which we discuss in Section 4. This reduction is formalised by the following proposition.

Proposition 1. *Let $M = (S, \overline{s}, \alpha, \delta)$ be a PIOS and A be an assumption about M. We denote by $pfa(M)$ the (straightforward) translation of M to a PFA, defined as $(S, \overline{s}, \alpha \cup \{\tau\}, \mathbf{P})$ where $\mathbf{P}(a)[s, s'] = \delta(s, a)(s')$ for $a \in \alpha \cup \{\tau\}$. Then, letting A^τ denote the PFA derived from A by adding τ to its alphabet and a probability 1 τ-loop to every state, we have that: $M \sqsubseteq_w A \ \Leftrightarrow \ pfa(M) \sqsubseteq A^\tau$.*

We can also perform a conversion in the opposite direction, translating an assumption PFA A into a (weak language) equivalent PIOS, which we denote $pios(A)$. We will use this technique in the next section when performing compositional verification.

Definition 8 (Assumption-to-PIOS conversion). *Given assumption $A = (S, \overline{s}, \alpha, \mathbf{P})$, and action partition $\alpha = (\biguplus_{i=1}^m \alpha^{I,i}) \uplus \alpha^O \uplus \alpha^H$, its conversion to a PIOS is defined as $pios(A) = (S', \overline{s}, \alpha, \delta)$, where $S' = S \uplus \{s^a | s \in S, a \in \alpha^H \cup \alpha^O\} \uplus \{s^i | s \in S, 1 {\leq} i {\leq} m\}$ and δ is constructed as follows. For any transition $s \xrightarrow{a} s'$, let p denote $\mathbf{P}(a)[s, s']$ and $p^{\max}(s)$ and $p_i^{\max}(s)$ be as defined in Definition 6. Then:*

- *if $a \in \alpha^O \cup \alpha^H$, then $\delta(s, \tau)(s^a) = \frac{p}{p^{\max}(s)}$ and $\delta(s^a, a)(s') = p^{\max}(s)$;*
- *if $a \in \alpha^{I,i}$ (for $1 {\leqslant} i {\leqslant} m$), then $\delta(s, \tau)(s^i) = \frac{p_i^{\max}(s)}{p^{\max}(s)}$ and $\delta(s^i, a)(s') = p \cdot \frac{p^{\max}(s)}{p_i^{\max}(s)}$.*

Proposition 2. *For assumption $A = (S, \overline{s}, \alpha, \mathbf{P})$, $pios(A)$ is a well defined PFA.*

Example 2. Consider PIOS M_1 from Example 1. Fig. 2 shows a valid assumption A for M (i.e. $M_1 \sqsubseteq_w A$) and the corresponding PIOS $pios(A)$. In A, state q_0 has two output actions leading to respective sub-distributions. Thus A is not a PIOS. In $pios(A)$, a τ transition and the states q_0^{ready} and q_0^{fail} (abbreviated to q_0^r and q_0^f) are added.

3.3 Assume-Guarantee Verification for PIOSs

Now, we are in a position to describe how to perform compositional verification using our framework. We focus on verification of a probabilistic safety property $\langle G \rangle_{\geqslant p}$ on a DTMC constructed as the parallel composition $M_1 \| M_2$ of a pair of PIOSs. For simplicity, we will assume that the property refers only to input/output actions of M_1 and

M_2. However, it is possible to adapt our approach to allow G to also refer to hidden actions of *either* M_1 *or* M_2. Furthermore, we will assume that all hidden actions of M_1 and M_2 have been renamed as τ actions. This affects neither the parallel composition $M_1 \| M_2$ nor the probability of satisfying G.

First, we introduce the idea of an *assume-guarantee triple* $\langle A \rangle\, M\, \langle G \rangle_{\geqslant p}$, with the intuitive meaning "whenever component M is part of a system satisfying the assumption A, then the system is guaranteed to satisfy property $\langle G \rangle_{\geqslant p}$".

Definition 9 (Assume-guarantee triple). *If M is a PIOS with alphabet α, A is an assumption about M and $\langle G \rangle_{\geqslant p}$ is a probabilistic safety property, then $\langle A \rangle\, M\, \langle G \rangle_{\geqslant p}$ is an* assume-guarantee triple*, with the following meaning:*

$$\langle A \rangle\, M\, \langle G \rangle_{\geqslant p} \;\Leftrightarrow\; \forall M'.\, (M' \sqsubseteq_w A \implies M' \| M \models \langle G \rangle_{\geqslant p}).$$

Using the translation $pios(A)$ from PFA to PIOS described above, checking whether a triple is true reduces to standard probabilistic model checking (see Section 2.2).

Proposition 3. *For A, M and $\langle G \rangle_{\geqslant p}$ as given in Definition 9, the assume-guarantee triple $\langle A \rangle\, M\, \langle G \rangle_{\geqslant p}$ holds if and only if $pios(A) \| M \models \langle G \rangle_{\geqslant p}$.*

We focus on an assume-guarantee proof rule for $M_1 \| M_2$ that is *asymmetric*, in that it only uses an assumption about one of the two components (M_1). Despite their simplicity, rules of this form have proved to be widely applicable in both non-probabilistic and probabilistic contexts [20,18,11].

Theorem 1. *Let M_1, M_2 be PIOSs, and A be an assumption for M_1. For a probabilistic safety property $\langle G \rangle_{\geqslant p}$, the following proof rule holds:*

$$\frac{M_1 \sqsubseteq_w A \;\; and \;\; \langle A \rangle\, M_2\, \langle G \rangle_{\geqslant p}}{M_1 \| M_2 \models \langle G \rangle_{\geqslant p}} \quad (\text{ASYM})$$

Theorem 1 means that, given an appropriate assumption A about M_1, we can decompose the verification of $M_1 \| M_2$ into two sub-problems: checking weak language inclusion between M_1 and A; and checking that $\langle A \rangle\, M_2\, \langle G \rangle_{\geqslant p}$. The former, as shown in Proposition 1, reduces to (strong) language inclusion on PFAs, which we discuss in the next section. The latter, as shown in Proposition 3, requires construction of the DTMC $pios(A) \| M_2$ and then application of standard probabilistic model checking techniques.

Example 3. Consider probabilistic safety property $\langle G \rangle_{\geqslant 0.9}$, where G means "*fail* never occurs". We can check this on running example $M_1 \| M_2$ using assumption A from Example 2. Since $M_1 \sqsubseteq_w A$, we just need to check that $pios(A) || M_2 \models \langle G \rangle_{\geqslant 0.9}$. As $pios(A) || M_2$ has a single path $(q_0 t_0) \xrightarrow{\tau * init, 0.08} (q_2 t_1) \xrightarrow{fail, 1} (q_4 t_1) \cdots$ containing *fail* with probability 0.08, $\langle G \rangle_{\geqslant 0.9}$ is satisfied (since $1 - 0.08 \geqslant 0.9$) and we are done.

We conclude this section by pointing out that our verification framework is *complete* in the following sense: if $M_1 \| M_2 \models \langle G \rangle_{\geqslant p}$ is true, we can always find an assumption A to verify it using rule (ASYM). This is because we can simply take A to be $pfa(M_1)$.

Algorithm 1 Semi-algorithm of deciding language inclusion for PFAs

```
Input: PFAs $A_1$ and $A_2$ over the same alphabet $\alpha$.
Output: true if $A_1 \sqsubseteq A_2$; or false and a counterexample $w' \in \alpha^*$ otherwise.
1: queue := $\{(\boldsymbol{\iota}_1, \boldsymbol{\iota}_2, \varepsilon)\}$, $V := \{(\boldsymbol{\iota}_1, \boldsymbol{\iota}_2, \varepsilon)\}$
2: while queue $\neq \varnothing$ do
3:     remove $(\boldsymbol{v}_1, \boldsymbol{v}_2, w)$ from the head of queue
4:     for all $a \in \alpha$ do
5:         $\boldsymbol{v}_1' := \boldsymbol{v}_1 \mathbf{P}_1(a)$; $\boldsymbol{v}_2' := \boldsymbol{v}_2 \mathbf{P}_2(a)$; $w' := wa$
6:         if $\boldsymbol{v}_1' \boldsymbol{\kappa}_1 > \boldsymbol{v}_2' \boldsymbol{\kappa}_2$ then return false and counterexample $w'$
7:         else if $(\boldsymbol{v}_1', \boldsymbol{v}_2', w')$ does not satisfy (C1) or (C2) then
8:             add $(\boldsymbol{v}_1', \boldsymbol{v}_2', w')$ to the tail of queue, $V := V \cup \{(\boldsymbol{v}_1', \boldsymbol{v}_2', w')\}$
9: return true
```

4 Deciding Language Inclusion for PFAs

As discussed above, verifying whether a component satisfies an assumption in our framework reduces to checking language inclusion between PFAs, i.e. deciding whether two PFAs A_1 and A_2 over the same alphabet α satisfy $A_1 \sqsubseteq A_2$. In this section, we propose a *semi*-algorithm for performing this check. If $A_1 \sqsubseteq A_2$ does *not* hold, then the algorithm is guaranteed to terminate and return a lexicographically minimal word as a counterexample; but if $A_1 \sqsubseteq A_2$ *does* hold, then the algorithm may not terminate. The latter case is unavoidable since, as mentioned previously, the problem is undecidable.[3] However, as we will demonstrate in Section 7, the method often works well in practice.

Algorithm 1 shows the semi-algorithm for deciding if $A_1 \sqsubseteq A_2$, where $A_i = (S_i, \overline{s}_i, \alpha, \mathbf{P}_i)$ for $i = 1, 2$. We also define $\boldsymbol{\iota}_i$ and $\boldsymbol{\kappa}_i$ as in Section 2.1. Inspired by the language equivalence decision algorithm in [24], our method proceeds by expanding a tree. Each node of the tree is of the form $(\boldsymbol{v}_1, \boldsymbol{v}_2, w)$, where w is a word and $\boldsymbol{v}_i = \boldsymbol{\iota}_i \mathbf{P}_i(w)$ (for $i = 1, 2$) is the vector of probabilities of reaching each state via word w in A_i . Note that $\boldsymbol{v}_i \boldsymbol{\kappa}_i$ is the probability of PFA A_i accepting the word w. The root of the tree is $(\boldsymbol{\iota}_1, \boldsymbol{\iota}_2, \varepsilon)$, where ε is the empty word. As shown in Algorithm 1, during the tree expansion we maintain a *queue* of tree nodes, which helps us to expand the tree in breadth-first order. In addition, we maintain a set V of non-leaf nodes, which initially only contains the root. The major contribution of our method, compared to the work in [24], is that we adopt different criteria to decide when to add a node to the non-leaf set V. In [24], the set V is maintained by calculating the span of vector space. However, for the language inclusion check, we cannot simply use the same criteria.

In each iteration, we remove a node $(\boldsymbol{v}_1, \boldsymbol{v}_2, w)$ from the head of *queue*. We then expand the tree by appending a set of its child nodes $(\boldsymbol{v}_1', \boldsymbol{v}_2', w')$, where $\boldsymbol{v}_1' := \boldsymbol{v}_1 \mathbf{P}_1(a)$, $\boldsymbol{v}_2' := \boldsymbol{v}_2 \mathbf{P}_2(a)$, and $w' := wa$ for all actions $a \in \alpha$. If there is a node $(\boldsymbol{v}_1', \boldsymbol{v}_2', w')$ such that $Pr_1(w') = \boldsymbol{v}_1' \boldsymbol{\kappa}_1 > \boldsymbol{v}_2' \boldsymbol{\kappa}_2 = Pr_2(w')$, then the algorithm terminates and returns w' as a counterexample for $A_1 \sqsubseteq A_2$. Otherwise, we check if we can *prune* each child node $(\boldsymbol{v}_1', \boldsymbol{v}_2', w')$ (i.e. make it a leaf node) by seeing if it satisfies either of the following two criteria:

[3] In fact, existing undecidability proofs [5,19] assume both accepting and non-accepting states, but it is straightforward to establish a similar result for PFAs with all states accepting.

(C1) $\boldsymbol{v}'_1\boldsymbol{\kappa}_1 = 0$.
(C2) *There exist* $|V|$ *non-negative rational numbers* ρ^i *such that, for all* $(\boldsymbol{v}^i_1, \boldsymbol{v}^i_2, w^i) \in V$, $\boldsymbol{v}'_1 \leq \sum_{0\leq i<|V|} \rho^i \boldsymbol{v}^i_1$ *and* $\boldsymbol{v}'_2 \geq \sum_{0\leq i<|V|} \rho^i \boldsymbol{v}^i_2$*, where* $\leq$ *and* $\geq$ *denote point-wise comparisons between vectors.*

Criterion (C1) is included because it is never possible to find a counterexample word with accepting probability less than $\boldsymbol{v}'_1\boldsymbol{\kappa}_1 = 0$. Criterion (C2) is included because any node satisfying it would guarantee $\boldsymbol{v}'_1\boldsymbol{\kappa}_1 \leq \boldsymbol{v}'_2\boldsymbol{\kappa}_2$; moreover, if the algorithm terminates and a node satisfies (C2), all of its descendants also satisfy (C2). We can thus make it a leaf node. Formal proofs of the above argument are in the appendix. In practice, (C2) can easily be checked using an SMT solver. If a node cannot be pruned, we add it to the tail of *queue* and to the non-leaf set *V*. The algorithm terminates when *queue* is empty, concluding that $A_1 \sqsubseteq A_2$.

Correctness. An intuitive explanation of the correctness of the semi-algorithm is as follows. If $A_1 \sqsubseteq A_2$ does not hold, then there must exist at least one counterexample w' such that $Pr_{A_1}(w') > Pr_{A_2}(w')$. By expanding the tree in a breadth-first fashion, we eventually reach the node corresponding to such a word w' and terminate the algorithm (see Line 7 of Algorithm 1). Furthermore, due to the order of traversal, the counterexample returned will be the lexicographically minimum one. On the other hand, if $A_1 \sqsubseteq A_2$ does hold, the algorithm may not terminate due to the undecidability of the underlying problem, Algorithm 1 could potentially keep finding nodes that do not satisfy either (C1) or (C2), and adding them to the *queue*. A formal proof of correctness is included in the appendix.

5 L*-Style Learning for PFAs

In this section, we propose a novel learning method that aims to learn a PFA for a target weighted language generated by an unknown PFA. Our technique operates in a similar style to the well-known L* algorithm [2], which infers a minimal DFA for an unknown regular language, in the sense that it constructs a similar *observation table* which now contains the accepting probabilities of words. Both L* and our method are *active learning* approaches, in which *queries* are posed to a *teacher* in an interactive fashion during learning. There are two types of queries. The first are *membership queries*, which ask the probability of accepting a particular word in the target PFA. The second are *equivalence queries*, which ask whether a hypothesised PFA accepts exactly the target language. Our method is also inspired by [4], where an active learning algorithm was proposed for learning multiplicity automata.

Given a target weighted language generated by an unknown PFA A with alphabet α, our learning algorithm incrementally builds an observation table (P, E, T), where P is a non-empty finite prefix-closed set of words, E is a non-empty finite suffix-closed set of words, and $T : ((P \cup P{\cdot}\alpha) \cdot E) \to [0, 1]$ maps each word to its accepting probability in A. Note that $\cdot$ is the concatenation operator over sets (the concatenation operator $\cdot$ over two words will sometimes be omitted). The rows of table (P, E, T) are labelled by elements in the prefix set $P \cup P{\cdot}\alpha$ and the columns are labelled by elements in the suffix set E. Any entry at row u and column e represents a word $u \cdot e$, and $T(u{\cdot}e)$ is

Algorithm 2 L*-style learning for PFAs

```
Input: The alphabet $\alpha$ of a target weighted language generated by an unknown PFA.
Output: A PFA accepting the target language.
 1: initialise the observation table $(P, E, T)$, letting $P = E = \{\varepsilon\}$, where $\varepsilon$ is the empty word
 2: fill $T$ by asking membership queries for $\varepsilon$ and each action $a \in \alpha$
 3: while $(P, E, T)$ is not closed or not consistent do
 4:     if $(P, E, T)$ is not closed then find $u \in P, a \in \alpha$ that make $(P, E, T)$ not closed
 5:         add $u \cdot a$ to $P$, and extend $T$ to $(P \cup P \cdot \alpha) \cdot E$ using membership queries
 6:     if $(P, E, T)$ is not consistent then find $a \in \alpha, e \in E$ that make $(P, E, T)$ not consistent
 7:         add $a \cdot e$ to $E$, and extend $T$ to $(P \cup P \cdot \alpha) \cdot E$ using membership queries
 8: construct a hypothesised PFA $A$ and ask an equivalence query
 9: if answer = no, with a counterexample $c$ then add $c$ and all its prefixes to $P$
10:     extend $T$ to $(P \cup P \cdot \alpha) \cdot E$ using membership queries, goto Line 4
11: else return PFA $A$
```

the value of the entry (the probability of the word $u{\cdot}e$). We use $row(u)$ to represent the $|E|$-dimensional row vector in the table (P, E, T) labelled by the prefix $u \in (P \cup P{\cdot}\alpha)$.

Inspired by [4], we define the notions of *closed* and *consistent* observation tables by establishing linear dependencies between row vectors as follows.

Definition 10. *Observation table (P, E, T) is* closed *if, for all $u \in P$ and $a \in \alpha$, there exist non-negative rational coefficients ϕ_i such that $row(u \cdot a) = \sum_{u_i \in P} \phi_i row(u_i)$.*

Definition 11. *Observation table (P, E, T) is* consistent *if, for any rational coefficients ψ_i, $\forall e \in E.\ \sum_{u_i \in P} \psi_i T(u_i{\cdot}e) = 0$ implies $\forall a \in \alpha, e \in E.\ \sum_{u_i \in P} \psi_i T(u_i{\cdot}a{\cdot}e) = 0$.*

We require the linear coefficients to be non-negative to make the observation table closed (for calculating transition probabilities); this is a stronger condition than in [4]. Note that L* also keeps a closed and consistent observation table, whose definitions can be considered as a special case of ours, when the table is filled with 1s and 0s.

Algorithm 2 summarises the learning algorithm, which maintains the observation table (P, E, T). Initially $P = E = \{\varepsilon\}$ (where ε is the empty word). The algorithm then asks membership queries for ε and each action $a \in \alpha$ to fill T. The main part of the algorithm is a **while** loop, in which the algorithm checks if the current observation table (P, E, T) is closed and consistent. If (P, E, T) is not closed (resp. consistent), then the algorithm finds and adds to the table such $u \in P$ and $a \in \alpha$ (resp. $a \in \alpha, e \in E$) that make it not closed (resp. consistent) as in Definition 10 (resp. Definition 11). The table is updated accordingly by new membership queries. Note that when (P, E, T) is not closed (resp. consistent), such a pair of u and a (resp. a and e) must exist. And since u (resp. e) is already in P (resp. E), adding $u \cdot a$ (resp. $a \cdot e$) keeps P (resp. E) a prefix-closed (resp. suffix-closed) set.

When (P, E, T) is closed and consistent, the learning algorithm builds a hypothesis PFA A and asks an equivalence query. If the *teacher* answers "no", a counterexample word c should be provided. The algorithm adds c and all its prefixes to P, updates the observation table and continues to check if the table is closed and consistent. If the *teacher* answers "yes", the algorithm terminates and returns A.

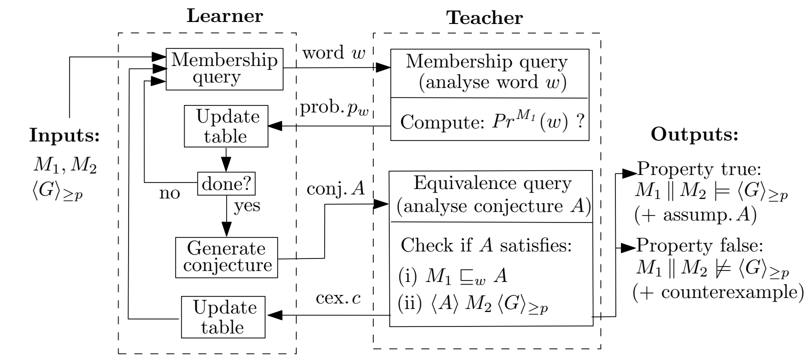

Learner
Teacher
Membership query
word w
Membership query (analyse word w)
prob. p_w
Compute: $Pr^{M_I}(w)$?
Update table
Inputs:
M_1, M_2
$\langle G \rangle_{\geq p}$
done?
no
yes
conj. A
Equivalence query (analyse conjecture A)
Generate conjecture
Check if A satisfies:
(i) $M_1 \sqsubseteq_w A$
(ii) $\langle A \rangle\, M_2\, \langle G \rangle_{\geq p}$
cex. c
Update table
Outputs:
Property true: $M_1 \parallel M_2 \models \langle G \rangle_{\geq p}$ (+ assump. A)
Property false: $M_1 \parallel M_2 \not\models \langle G \rangle_{\geq p}$ (+ counterexample)

Fig. 3. L*-style PFA learning loop for probabilistic assumption generation.

Given a closed and consistent observation table (P, E, T), the learning algorithm builds a PFA $A = (S, \overline{s}, \alpha, \mathbf{P})$ as follows. Firstly, it computes a subset of P, denoted $con(P)$, such that every element of $\{row(u) | u \in P\}$ can be represented as a *conical combination* of elements in $\{row(v) | v \in con(P)\}$, i.e. there exists a set of non-negative rational coefficients λ_i that for all $u \in P$, $row(u) = \sum_{v_i \in con(P)} \lambda_i row(v_i)$. Then, it defines the set of states $S = \{s_0, \ldots, s_{n-1}\}$ such that each state s_i corresponds to a row vector in $\{row(v) | v \in con(P)\}$, and the initial state $\overline{s}$ corresponds to $row(\varepsilon)$ (note that $row(\varepsilon) \in con(P)$). To obtain $\mathbf{P}(a)$ (for $a \in \alpha$), a set of rational coefficients γ_j is computed such that $row(s_i \cdot a) = \sum_{s_j \in S} \gamma_j row(s_j)$, where $s_i \in S$; and then $\mathbf{P}(a)[i,j] := \gamma_j \cdot (T(s_j \cdot \varepsilon) / T(s_i \cdot \varepsilon))$. We show in the appendix that A is a well-defined PFA and that it conforms with the table (P, E, T), i.e. , $\forall u \in (P \cup P \cdot \alpha)$, $\forall e \in E$, $Pr^A(u \cdot e) = T(u \cdot e)$.

Correctness. When the learning algorithm terminates, the learnt PFA A accepts the target language. This is guaranteed by the result of the equivalence query. Unfortunately, we cannot prove the termination of our method. For L*, the basis for the corresponding proof is that, for any regular language, there exists a unique minimal accepting DFA. However, an analogous property does not exist for weighted languages and PFAs. According to [4], the smallest multiplicity automaton *can* be learnt given a weighted language. However, as shown in [7], converting a multiplicity automaton to a PFA (even for the subclass that define stochastic languages) is not always possible.

6 Learning Assumptions for Compositional Verification

Finally, we build upon the the techniques introduced in Sections 4 and 5 to produce a fully-automated implementation of the assume-guarantee framework proposed in Section 3. In particular, we use PFA learning to automatically generate assumptions to perform compositional verification.

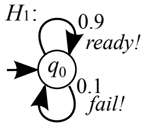
Fig. 3 summarises the overall structure of our approach, which aims to verify (or refute) $M_1 \parallel M_2 \models \langle G \rangle_{\geqslant p}$ for two PIOSs M_1, M_2 and a probabilistic safety property $\langle G \rangle_{\geqslant p}$. This is done using proof rule (ASYM) from Section 3, with the required assumption PFA A about component M_1 being generated through learning. The left-hand side of the figure shows the learning algorithm of Section 5, which drives the whole process; the right-hand side shows the teacher.

The teacher answers *membership queries* (about word w) by computing the probability $Pr^{M_1}(w)$ that M_1 accepts word w. It answers *equivalence queries* (about conjectured PFA A) by checking if A satisfies both premises of rule (ASYM): (i) $M_1 \sqsubseteq_w A$, and (ii) $\langle A \rangle\, M_2\, \langle G \rangle_{\geqslant p}$. The first is done using Proposition 1 and the algorithm in Section 4. The second is done using Proposition 3, which reduces to probabilistic model checking of the DTMC $pios(A) \parallel M_2$.

If both premises are true, we can conclude that $M_1 \parallel M_2 \models \langle G \rangle_{\geqslant p}$ holds. Otherwise, the teacher needs to provide a counterexample c for the learning algorithm to update the observation table and proceed. If premise (i) failed, then c is taken as the word showing the violation of (weak) language inclusion. If premise (ii) failed, we try to extract c from the results of model checking. We extract a *probabilistic counterexample* [12] C: a set of paths showing $pios(A)||M_2 \not\models \langle G \rangle_{\geqslant p}$. Following the same approach as [10], we transform C into a (small) fragment of M_1 (denoted M_1^C) and check whether $M_1^C||M_2 \not\models \langle G \rangle_{\geqslant p}$. If so, we stop the learning loop, concluding that $M_1||M_2 \not\models \langle G \rangle_{\geqslant p}$. If, on the other hand, C is a *spurious counterexample*, we can always extract, from C a counterexample (word) c such that the learning algorithm can update its observation table (full details can be found in the appendix).

From the arguments above, we can show that, when the learning loop terminates, it always yields a correct result. However, since the loop is driven by the learning algorithm of Section 5, whose termination we cannot prove, we are also unable to guarantee that the loop finishes. However, as will show next, learning does indeed terminate successfully on a variety of real examples.

Example 4. Finally, we illustrate the learning loop described above through an example. Given PIOSs M_1 and M_2 as illustrated in Fig. 1, we show how to learn an assumption A about M_1 with the alphabet $\alpha = \{\mathit{fail}!, \mathit{ready}!, d0?, d1?\}$. Firstly, we initialise the observation table (P, E, T) of the learning algorithm (see Algorithm 2 in Section 5), where the prefix set $P = \{\varepsilon\}$ and the suffix set $E = \{\varepsilon\}$. A first membership query is made with the empty word ε; and the answer is 1, since the accepting probability $Pr^{M_1}(\varepsilon) = 1$. Then each action $a \in \alpha$ is added to the set $P \cdot \alpha$ and the table is filled with membership query answers of $Pr^{M_1}(a)$ The table is closed and consistent, so a hypothesis PFA H_1 is built as shown in Fig. 4.

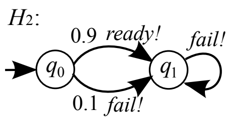


T		E: ε
P	ε	1
$P \cdot \alpha$	*fail!*	0.1
	ready!	0.9
	d0?	0
	d1?	0

Fig. 4. The first hypothesis PFA H_1 and its corresponding observation table.

Now, an equivalence query is made for H_1. A counterexample word $c_1 = fail! \cdot fail!$ is provided by the *teacher* since $Pr^{M_1}(c_1) = 0.1 > Pr^{H_1}(c_1) = 0.01$, indicating that $M_1 \not\sqsubseteq_w H_1$. The observation table (P, E, T) is updated by adding c_1 and all its prefixes to P and extending T to $(P \cup P \cdot \alpha) \cdot E$ using membership queries as shown in Fig. 5. Again, this table is closed and consistent. Now a second hypothesis PFA H_2 (Fig. 5) is constructed and an equivalence query is made for it. The *teacher* provides a counterexample $c_2 = ready! \cdot d0?$, because $Pr^{M_1}(c_2) = 0.9 > Pr^{H_1}(c_2) = 0$. The table is updated again using c_2 as shown in Fig. 6.

Since this table is also closed and consistent, a hypothesis PFA is built as the A in Fig. 2 and an equivalence query is asked for it. As described in Example 2, A is a good assumption about M_1 to verify the safety property. Therefore, the learning loop terminates.



T		E: ε
P	ε	1
	$fail!$	0.1
	$fail! \cdot fail!$	0.1
$P \cdot \alpha$	$fail!$	0.1
	$ready!$	0.9
	$d0?$	0
	$d1?$	0
	$fail! \cdot fail!$	0.1
	$fail! \cdot ready!$	0
	$fail! \cdot d0?$	0
	$fail! \cdot d1?$	0
	$fail! \cdot fail! \cdot fail!$	0.1
	$fail! \cdot fail! \cdot ready!$	0
	$fail! \cdot fail! \cdot d0?$	0
	$fail! \cdot fail! \cdot d1!$	0

Fig. 5. The second hypothesis PFA H_2 and its corresponding observation table.

7 Experimental Results

We have built a prototype tool implementing the compositional verification framework described in this paper. Our prototype uses PRISM [17] to answer queries from the assumption learning procedure and the SMT solver Yices 2 to solve the linear arithmetic problems in the language inclusion check (Section 4) and in PFA learning (Section 5). Experiments were run on a 2.80GHz PC with 32GB RAM running 64-bit Fedora.

We applied our implementation to several benchmark case studies. The first is the contract signing protocol of Even/Goldreich/Lempel (*egl*), where two parties are exchanging N pairs of secrets and each secret contains L bits. The second and third are variants of the bounded retransmission (*brp*) protocol which sends a file in N chunks,

	T	E: ε
P	ε	1
	$fail!$	0.1
	$fail! \cdot fail!$	0.1
	$ready!$	0.9
	$ready! \cdot d0?$	0.9
$P \cdot \alpha$	$fail!$	0.1
	$ready!$	0.9
	$d0?$	0
	$d1?$	0
	$fail! \cdot fail!$	0.1
	$fail! \cdot ready!$	0
	$fail! \cdot d0?$	0
	$fail! \cdot d1?$	0
	$fail! \cdot fail! \cdot fail!$	0.1
	$fail! \cdot fail! \cdot ready!$	0
	$fail! \cdot fail! \cdot d0?$	0
	$fail! \cdot fail! \cdot d1!$	0
	$ready! \cdot fail!$	0
	$ready! \cdot ready!$	0
	$ready! \cdot d0?$	0.9
	$ready! \cdot d1?$	0.9
	$ready! \cdot d0? \cdot fail!$	0
	$ready! \cdot d0? \cdot ready!$	0
	$ready! \cdot d0? \cdot d0?$	0.9
	$ready! \cdot d0? \cdot d1!$	0.9

Fig. 6. The third closed and consistent observation table (see its corresponding PFA A in Fig. 2).

Case study [parameters]		Component sizes		Compositional					Non-compositional	
		$\lvert M_2 \otimes G^{err} \rvert$	$\lvert M_1 \rvert$	$\lvert A \rvert$	$\lvert pios(A) \rvert$	$\lvert EQ \rvert$	Result	Time (s)	Result	Time (s)
egl [N L]	6 8	8,265,625	433	**24**	37	5	0.514648	39.8	0.514648	5.39
	7 8	48,399,849	505	**28**	43	5	0.015380	58.6	0.015380	5.58
	8 2	31,573,161	193	**32**	49	5	0.503845	67.6	0.503845	4.24
brp-1 [N Max]	16 5	642	191	**2**	3	2	1.121E-8	2.3	1.121E-8	0.50
	32 5	1,282	191	**2**	3	2	2.241E-8	3.8	2.241E-8	0.92
	64 5	2,562	191	**2**	3	2	4.482E-8	4.1	4.482E-8	1.79
brp-2 [N Max]	2 2	764	50	**19**	25	9	1.792E-2	55.5	1.792E-2	0.04
	2 3	764	63	**39**	57	7	3.740E-3	172.2	3.740E-3	0.05
	4 4	764	146	-	-	-	-	-	1.556E-3	0.17
client-server [N]	2	72	274	**17**	17	10	0.079999	61.1	0.079999	0.019
	3	372	416	**31**	31	14	0.079999	325.9	0.079999	0.118
	4	1,728	562	**49**	49	37	0.079999	3,155.5	0.079999	0.247

Fig. 7. Performance of the learning-based compositional verification framework.

but allows only a bounded (Max) number of retransmissions of each chunk (*brp-1* and *brp-2* differ only with regards to which component is M_1). The fourth is a synchronous variant of the *client-server* model from [10], with N clients requesting resources from a server. Models, properties and references are available online [25].

Fig. 7 shows experimental results for these case studies. For each model, we first report the state space of components M_1 and M_2 (the latter together with the safety property's DFA G^{err}). Then, we show the state space of the learnt assumption A and its conversion to a PIOS $pios(A)$. To illustrate the performance of the learning loop (Fig. 3), we also show the number of requested equivalence queries, $|EQ|$, which corresponds to the number of iterations in the loop. Finally, we report the compositional verification result[4] and the total run-time. For comparison, we also include the result from non-compositional verification using PRISM. A comparison with our earlier work to learn probabilistic safety properties [10] is not applicable since that framework is only meaningful for Markov decision processes, not Markov chains.

Our primary concern in these experiments is to investigate the feasibility of automatically generating valid and compact assumptions using our approach. Indeed, we see that, in all but one case, we successfully learn assumptions that permit compositional verification to be performed; furthermore, these assumptions are usually considerably smaller than the components that they represent (even after the PFA-to-PIOS conversion) and the results obtained are exact. One model ($brp2$) performs less well, including an instance where the loop does not terminate. Investigations show that this is due to round-off errors in the numerical computation performed by PRISM being converted to rationals for the SMT solver. We plan to investigate the use of arbitrary precision arithmetic to alleviate this problem. Currently our implementation is a prototype and (unsurprisingly) results in slower run-times than non-compositional verification using (highly-optimised) PRISM.

8 Conclusion

In this paper, we have presented a novel assume-guarantee framework where multi-component systems are modelled by discrete-time Markov chains, components are probabilistic I/O systems and assumptions are probabilistic finite automata. Based on new techniques for checking PFA language inclusion and active learning of PFAs, we have built a fully-automated implementation of the framework that generates assumptions and then performs compositional verification. Future work in this area will include exploring ways to extend our techniques to models with nondeterminism and investigating termination conditions for the language inclusion check and PFA learning.

Acknowledgments. The authors are supported by ERC Advanced Grant VERIWARE, EU FP7 project CONNECT and EPSRC grant EP/F001096/1. We also thank Taolue Chen, Stefan Kiefer, Björn Wachter and James Worrell for insightful discussions.

References

1. de Alfaro, L., Henzinger, T., Jhala, R.: Compositional methods for probabilistic systems. In: Proc. CONCUR'01. LNCS, vol. 2154. Springer (2001)
2. Angluin, D.: Learning regular sets from queries and counterexamples. Information and Computation 75(2), 87–106 (1987)
3. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
4. Bergadano, F., Varricchio, S.: Learning behaviors of automata from multiplicity and equivalence queries. SIAM J. Comput. 25(6), 1268–1280 (1996)
5. Blondel, V., Canterini, V.: Undecidable problems for probabilistic automata of fixed dimension. Theory of Computing Systems 36, 231–245 (2001)

[4] In fact, the table actually shows the maximum probability of *violating* the safety property.

6. Chen, Y.F., Clarke, E., Farzan, A., Tsai, M.H., Tsay, Y.K., Wang, B.Y.: Automated assume-guarantee reasoning through implicit learning. In: Proc. CAV'10 (2010)
7. Denis, F., Esposito, Y.: Learning classes of probabilistic automata. In: COLT (2004)
8. Doyen, L., Henzinger, T.A., Raskin, J.F.: Equivalence of labeled Markov chains. Int. J. Found. Comput. Sci. 19(3), 549–563 (2008)
9. Feng, L., Han, T., Kwiatkowska, M., Parker, D.: Learning-based compositional verification for synchronous probabilistic systems. In: Proc. ATVA'11. LNCS, Springer (2011)
10. Feng, L., Kwiatkowska, M., Parker, D.: Compositional verification of probabilistic systems using learning. In: Proc. QEST'10. pp. 133–142. IEEE CS Press (2010)
11. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: Proc. TACAS'11 (2011)
12. Han, T., Katoen, J.P., Damman, B.: Counterexample generation in probabilistic model checking. IEEE Trans. Software Eng. 35(2), 241–257 (2009)
13. de la Higuera, C., Oncina, J.: Learning stochastic finite automata. In: ICGI (2004)
14. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer-Verlag (1976)
15. Kiefer, S., Murawski, A.S., Ouaknine, J., Wachter, B., Worrell, J.: Language equivalence for probabilistic automata. In: Proc. CAV'11 (2011), to appear
16. Kumar, J.A., Vasudevan, S.: Automatic compositional reasoning for probabilistic model checking of hardware designs. In: Proc. QEST'10. IEEE CS Press (2010)
17. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Proc. CAV'11. LNCS, Springer (2011), to appear
18. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: Proc. TACAS'10 (2010)
19. Murawski, A., Ouaknine, J.: On probabilistic program equivalence and refinement. In: Proc. CONCUR'05, volume 3653 of LNCS. LNCS, vol. 3653. Springer (2005)
20. Pasareanu, C., Giannakopoulou, D., Bobaru, M., Cobleigh, J., Barringer, H.: Learning to divide and conquer: Applying the L* algorithm to automate assume-guarantee reasoning. Formal Methods in System Design 32(3), 175–205 (2008)
21. Rabin, M.: Probabilistic automata. Information and Control 6, 230–245 (1963)
22. Segala, R.: Modelling and Verification of Randomized Distributed Real Time Systems. Ph.D. thesis, Massachusetts Institute of Technology (1995)
23. Tzeng, W.G.: Learning probabilistic automata and Markov chains via queries. Mach. Learn. 8, 151–166 (1992)
24. Tzeng, W.G.: A polynomial-time algorithm for the equivalence of probabilistic automata. SIAM J. Comput. 21(2), 216–227 (1992)
25. `http://www.prismmodelchecker.org/files/atva11/`

APPENDIX

This appendix contains proofs for the results stated in the main paper.

A Proofs for Section 3

Proposition 1. Let $M = (S, \overline{s}, \alpha, \delta)$ be a PIOS and A be an assumption about M. We denote by $\mathit{pfa}(M)$ the (straightforward) translation of M to a PFA, defined as $(S, \overline{s}, \alpha \cup \{\tau\}, \mathbf{P})$ where $\mathbf{P}(a)[s, s'] = \delta(s, a)(s')$ for $a \in \alpha \cup \{\tau\}$. Then, letting A^τ denote the PFA derived from A by adding τ to its alphabet and a probability 1 τ-loop to every state, we have that: $M \sqsubseteq_w A \;\Leftrightarrow\; \mathit{pfa}(M) \sqsubseteq A^\tau$.

Proof. For each word w in M, the following holds:

$$Pr^M(w) \overset{(1)}{\leqslant} Pr^M(st(w)) \overset{(2)}{\leqslant} Pr^A(st(w)) \overset{(3)}{=} Pr^{A^\tau}(st(w)) \overset{(4)}{=} Pr^{A^\tau}(w),$$

where (1) holds because $st(w)$ represents a set of words $T = \{w' \mid st(w') = st(w)\}$, and $w \in T$. (2) is due to the definition of $M \sqsubseteq_w A$. (3) is true because all the τ actions in A^τ have self-loops with probability 1. (4) is true because for each word w in A^τ, $Pr^{A^\tau}(w){=}Pr^{A^\tau}(\bar{w})$, where $\bar{w}$ is the τ-free version of w. This is because every τ is a self-loop with probability 1. It is also clear that $Pr^M(w) = Pr^{\mathit{pfa}(A)}(w)$. The above (in)equalities hold from both directions, thus $M \sqsubseteq_w A$ iff $\mathit{pfa}(M) \sqsubseteq A^\tau$. $\square$

Proposition 2. Given assumption $A = (S, \overline{s}, \alpha, \mathbf{P})$, and action partition $\alpha = (\biguplus_{i=1}^m \alpha^{I,i}) \uplus \alpha^O \uplus \alpha^H$, $\mathit{pios}(A) = (S', \overline{s}, \alpha, \delta)$ is well-defined.

Proof. Since the action partition is given, it is to prove that 1) $|enab(s)| = 1$ if $enab(s) \in \alpha^O \cup \alpha^H \cup \{\tau\}$ 2) $enab(s) = \alpha^{I,i}$ for some input action bundle $\alpha^{I,i}$.

For any state $s \in S$, due to the construction, s has only one action τ, which satisfies 1). Since there can be a unique action with name $a \in \alpha$ from any state s in A, for any state s and action $a \in \alpha^I \cup \alpha^H$, the newly created state s^a is unique. Thus, the transition $\delta(s, \tau)(s^a)$ is well-defined. Since $p \in [0, 1]$ and $p^{\max}(s) \in [0, 1]$ and $p \le p^{\max}(s)$, thus $\frac{p}{p^{\max}(s)}$ is a probability. For any state $s^a \in S' \setminus S$, the transition to s' with action a is unique, thus $\delta(s^a, a)(s')$ is well-defined. $p^{\max}(s) \in [0, 1]$ is a probability. The probability from s to s' is $\mathbf{P}(a)[s, s'] = p$ in A, while in $\mathit{pios}(A)$ the probability is $\delta(s, \tau)(s^a) \cdot \delta(s^a, a)(s') = \frac{p}{p^{\max}(s)} \cdot p^{\max}(s) = p$.

Likewise, there is at most one input action bundle $\alpha^{I,i}$ from any state $s \in S$, the newly created state s^i is unique. Thus, the transition $\delta(s, \tau)(s^i)$ is well-defined and $\frac{p_i^{\max}(s)}{p^{\max}(s)}$ is a probability. For any state $s^i \in S \setminus S$, the transition to s' with action a is unique, thus $\delta(s^i, a)(s')$ is well-defined and $p{\cdot}\frac{p^{\max}(s)}{p_i^{\max}(s)}$ is a probability. Furthermore, the probability from s to s' is p in A, which equals that in $\mathit{pios}(A)$. Besides, due to Def. 6 the input actions in a bundle will be enabled in the all-or-none fashion, it is thus guaranteed that $enab(s) = \alpha^{I,i}$ for some input action bundle $\alpha^{I,i}$. $\square$

Proposition 3. For A, M and $\langle G\rangle_{\geqslant p}$ as given in Definition 9, the assume-guarantee triple $\langle A\rangle\ M\ \langle G\rangle_{\geqslant p}$ holds if and only if $pios(A)\|M \models \langle G\rangle_{\geqslant p}$.

To prove Proposition 3, we require the following two lemmas.

Lemma 1. *If $M_1 \sqsubseteq_w A$, then $M_1||M_2 \sqsubseteq_w pios(A)||M_2$.*

Proof. Given any word w in $M_1||M_2$, w can be uniquely mapped back onto M_1 and M_2, as $w_1 = w{\downharpoonright_{M_1}}$ and $w_2 = w{\downharpoonright_{M_2}}$, by sequentially taking the synchronous actions and their respective hidden actions (if not $\perp$). We write $w = w_1||w_2$ for simplicity. For a word in a composed model, since only input/output actions are visible and all the other actions are τ actions in M_1, M_2 and $pios(A)$, it is easy to see that given a τ-free word $\bar{w} \in \alpha^*$, $\bar{w} = \bar{w}_1||\bar{w}_2$.

Given a path θ in $M_1||M_2$ with $tr(\theta) = w$, θ can be decomposed into θ_1 in M_1 with $tr(\theta_1) = w_1$ and $\theta_2 \in M_2$ with $tr(\theta_2) = w_2$. We write $\theta = \theta_1||\theta_2$ for simplicity. The probability of the composed path is

$$Pr^{M_1||M_2}(\theta_1||\theta_2) = Pr^{M_1}(\theta_1)\cdot Pr^{M_2}(\theta_2). \tag{1}$$

Since $M_1 \sqsubseteq_w A$, we have that $Pr^{M_1}(st(w_1)) \leqslant Pr^{A}(st(w_1))$, which means

$$\sum_{tr(\theta_1)=st(w_1)} Pr^{M_1}(\theta_1) \leqslant \sum_{tr(\theta_1')=st(w_1)} Pr^{A}(\theta_1'). \tag{2}$$

We have that:

$$\begin{aligned}
& Pr^{M_1||M_2}(\bar{w})\\
= \ & Pr^{M_1||M_2}(\bar{w}_1||\bar{w}_2)\\
= \ & \sum_{st(w')=\bar{w}_1||\bar{w}_2}\ \sum_{tr(\theta_1||\theta_2)=w'} Pr^{M_1||M_2}(\theta_1||\theta_2)\\
= \ & \sum_{st(w')=\bar{w}_1||\bar{w}_2}\ \sum_{tr(\theta_1||\theta_2)=w'} Pr^{M_1}(\theta_1)\cdot Pr^{M_2}(\theta_2)\\
= \ & \sum_{st(w')=\bar{w}_1||\bar{w}_2, w'=\theta_1||\theta_2}\ \sum_{tr(\theta_2)=\bar{w}_2}\ \sum_{tr(\theta_1)=\bar{w}_1} Pr^{M_1}(\theta_1)\cdot Pr^{M_2}(\theta_2)\\
= \ & \sum_{st(w')=\bar{w}_1||\bar{w}_2, w'=\theta_1||\theta_2}\ \sum_{tr(\theta_2)=\bar{w}_2} Pr^{M_1}(\bar{w}_1)\cdot Pr^{M_2}(\theta_2)\\
\overset{(2)}{\leqslant} \ & \sum_{st(w')=\bar{w}_1||\bar{w}_2, w'=\theta_1||\theta_2}\ \sum_{tr(\theta_2)=\bar{w}_2} Pr^{pios(A)}(\bar{w}_1)\cdot Pr^{M_2}(\theta_2)\\
= \ & \sum_{st(w')=\bar{w}_1||\bar{w}_2, w'=\theta_1||\theta_2}\ \sum_{tr(\theta_2)=st(w_2)}\ \sum_{tr(\theta_1)=st(w_1)} Pr^{pios(A)}(\theta_1)\cdot Pr^{M_2}(\theta_2)\\
= \ & Pr^{pios(A)||M_2}(\bar{w}_1||\bar{w}_2)\\
= \ & Pr^{pios(A)||M_2}(\bar{w})
\end{aligned}$$

□

Lemma 2. *If $M_1||M_2 \sqsubseteq_w pios(A)||M_2$ and $pios(A)||M_2 \models \langle G\rangle_{\geqslant p}$, then $M_1||M_2 \models \langle G\rangle_{\geqslant p}$.*

Proof. Given a DTMC D and DFA G^{err} for G, we can build a product $D \otimes G^{err}$ in standard fashion [3] such that D satisfies $\langle G\rangle_{\geq p}$ iff:

$$Pr^{D\otimes G^{err}}\{\theta \in Paths^{D\otimes G^{err}} \mid \theta \text{ is accepting in } D \otimes G^{err}\} \leqslant 1 - p. \qquad (3)$$

Let $D_1{=}M_1||M_2$ and $D_2{=}pios(A)||M_2$. Both are DTMCs and $D_2 \models \langle G\rangle_{\geqslant p}$ so, by (3), we have that $Pr^{D_2\otimes G}(\diamondsuit\, err) \leqslant 1 - p$ where $\diamondsuit\, err$ is a shorthand for the set of accepting paths in $D \otimes G^{err}$. Since $D_1 \sqsubseteq_w D_2$, and there is no probability in G, then $D_1 \otimes G \sqsubseteq_w D_2 \otimes G$. Thus

$$Pr^{D_1\otimes G}(\diamondsuit\, err') \leqslant Pr^{D_2\otimes G}(\diamondsuit\, err) \leqslant 1 - p.$$

Therefore, $M_1||M_2 \models \langle G\rangle_{\geqslant p}$. □

Proof (of Proposition 3). The result follows directly from Lemma 1 and Lemma 2.

Theorem 1. Let M_1, M_2 be PIOSs, and A be an assumption for M_1. For a probabilistic safety property $\langle G\rangle_{\geqslant p}$, the following proof rule holds:

$$\frac{M_1 \sqsubseteq_w A \;\text{ and }\; \langle A\rangle\, M_2\, \langle G\rangle_{\geqslant p}}{M_1\|M_2 \models \langle G\rangle_{\geqslant p}} \quad (\text{ASYM})$$

Proof. The result follows directly from Definition 9.

Lemma 3. *Given PIOSs M_1 and M_2, $M_1||M_2$ is a DTMC.*

Proof. We distinguish three cases for the states in $M_1||M_2$. Note that $\mu_1 \times \mu_2$ is a distribution.

1. $(s_1, s_2) \xrightarrow{a}$: Since only one of the states s_1 and s_2 has more than one input action and the other state has one output action, the resulting state (s_1, s_2) will have only one matching action enabled.
2. $(s_1, s_2) \xrightarrow{b_1 * b_2}$: Since each states s_1 or s_2 has exactly one hidden action, there is only one combined action enabled.
3. $(s_1, s_2) \xrightarrow{b_1 * \bot}$ or $(s_1, s_2) \xrightarrow{\bot * b_2}$: If one state has a hidden action enabled, then no matter whether the other state has one or more external states or is an endpoint, there will be one combined action enabled.

In each case, there is only one action enabled, therefore $M_1||M_2$ is a DTMC. □

Lemma 4 (Hiding). *If $M_1^\tau||M_2 \models \langle G\rangle_{\geqslant p}$, then $M_1||M_2 \models \langle G\rangle_{\geqslant p}$.*

Proof. Since renaming changes neither the structure nor the probabilities on the transitions in M_1, the structure and transition probabilities in $M_1^\tau||M_2$ and $M_1||M_2$ stays the same. Since model checking MC $M_1||M_2$ and $\langle G\rangle_{\geqslant p}$ boils down to computing the reachability of certain action-related states in their product and the renamed actions are not in the DFA G, the reachability probability won't change because of renaming.

□

B Proofs for Section 4

To prove the correctness of Algorithm 1, we first prove the following two lemmas.

Lemma 5. *If a node* $(\boldsymbol{v}'_1, \boldsymbol{v}'_2, w')$ *satisfies* $(C2)$*, then* $Pr_{A_1}(w') \leq Pr_{A_2}(w')$.

Proof. Based on the definitions of $Pr_{A_1}(w')$ and $Pr_{A_2}(w')$, we have

$$Pr_{A_1}(w') = \boldsymbol{v}'_1\boldsymbol{\kappa}_1 \leq \sum_{0\leq i<|V|} \rho^i \boldsymbol{v}^i_1\boldsymbol{\kappa}_1 = \sum_{0\leq i<|V|} \rho^i Pr_{A_1}(w^i),$$

and

$$Pr_{A_2}(w') = \boldsymbol{v}'_2\boldsymbol{\kappa}_2 \geq \sum_{0\leq i<|V|} \rho^i \boldsymbol{v}^i_2\boldsymbol{\kappa}_2 = \sum_{0\leq i<|V|} \rho^i Pr_{A_2}(w^i).$$

According to Line 10 of Algorithm 1, for every node $(\boldsymbol{v}^i_1, \boldsymbol{v}^i_2, w^i) \in V$, we have checked that $Pr_{A_1}(w^i) = \boldsymbol{v}^i_1\boldsymbol{\kappa}_1 \leq \boldsymbol{v}^i_2\boldsymbol{\kappa}_2 = Pr_{A_2}(w^i)$. Thus, we have

$$Pr_{A_1}(w') \leq \sum_{0\leq i<|V|} \rho^i Pr_{A_1}(w^i) \leq \sum_{0\leq i<|V|} \rho^i Pr_{A_2}(w^i) \leq Pr_{A_2}(w').$$

□

Lemma 6. *Given a node* $(\boldsymbol{v}_1, \boldsymbol{v}_2, w) \in V$*, if the node* $(\boldsymbol{v}_1\mathbf{P}_1(a), \boldsymbol{v}_2\mathbf{P}_2(a), wa)$ *satisfies* $(C2)$ *for any action* $a \in \alpha$*, then the node* $(\boldsymbol{v}_1\mathbf{P}_1(u), \boldsymbol{v}_2\mathbf{P}_2(u), wu)$ *also satisfies* $(C2)$ *for any finite words* $u \in \alpha^*$.

Proof. Recall that a node $(\boldsymbol{v}_1\mathbf{P}_1(a), \boldsymbol{v}_2\mathbf{P}_2(a), wa)$ satisfies $(C2)$ iff there exists a set of non-negative rational numbers $\hat{\rho}^i$ such that

$$\begin{cases} \boldsymbol{v}_1\mathbf{P}_1(a) \leq \sum_{0\leq i<|V|} \hat{\rho}^i \boldsymbol{v}^i_1, \\ \boldsymbol{v}_2\mathbf{P}_2(a) \geq \sum_{0\leq i<|V|} \hat{\rho}^i \boldsymbol{v}^i_2, \end{cases} \tag{4}$$

And a node $(\boldsymbol{v}_1\mathbf{P}_1(u), \boldsymbol{v}_2\mathbf{P}_2(u), wu)$ satisfies $(C2)$ iff there exists a set of non-negative rational numbers ρ^i such that

$$\begin{cases} v_1\mathbf{P}_1(u) \leq \sum_{0\leq i<|V|} \rho^i \boldsymbol{v}^i_1, \\ v_2\mathbf{P}_2(u) \geq \sum_{0\leq i<|V|} \rho^i \boldsymbol{v}^i_2, \end{cases} \tag{5}$$

We prove this lemma by induction on the length of $|u|$. Suppose that $(\boldsymbol{v}_1\mathbf{P}_1(u), \boldsymbol{v}_2\mathbf{P}_2(u), wu)$ satisfying (C2) for some finite word u. We have

$$\begin{aligned}
\boldsymbol{v}_1\mathbf{P}_1(ua) &= \boldsymbol{v}_1\mathbf{P}_1(u)\mathbf{P}_1(a) \\
&\overset{I.H.(5)}{\leq} \sum_{0\leq i<|V|} \rho^i \boldsymbol{v}_1^i \mathbf{P}_1(a) \\
&\overset{(4)}{\leq} \sum_{0\leq i<|V|} \rho^i \sum_{0\leq j<|V|} \hat{\rho}^{ij} \boldsymbol{v}_1^j \\
&= \sum_{0\leq j<|V|} \Big(\sum_{0\leq i<|V|} \rho^i \hat{\rho}^{ij} \Big) \boldsymbol{v}_1^j
\end{aligned}$$

Let $\lambda^j = \sum_{0\leq i<|V|} \rho^i \hat{\rho}^{ij}$, we get

$$\boldsymbol{v}_1\mathbf{P}_1(ua) \leq \sum_{0\leq i<|V|} \lambda^j \boldsymbol{v}_1^j$$

where λ^j is a set of nonnegative rational numbers. Symmetrically, we can deduce that

$$\boldsymbol{v}_2\mathbf{P}_2(ua) \geq \sum_{0\leq i<|V|} \lambda^j \boldsymbol{v}_2^j$$

Therefore, for all $a \in \alpha$, $(\boldsymbol{v}_1\mathbf{P}_1(ua), \boldsymbol{v}_2\mathbf{P}_2(ua), wua)$ also satisfies (C2). $\square$

Theorem 2 (Correctness). *Algorithm 1 is correct when it terminates.*

Proof. If the algorithm terminates in Line 7, then a word w' is found such that

$$Pr_{A_1}(w') = \boldsymbol{\iota}_1\mathbf{P}_1(w')\boldsymbol{\kappa}_1 = \boldsymbol{v}'_1\boldsymbol{\kappa}_1 > \boldsymbol{v}'_2\boldsymbol{\kappa}_2 = \boldsymbol{\iota}_2\mathbf{P}_2(w')\boldsymbol{\kappa}_2 = Pr_{A_2}(w').$$

Therefore, $A_1 \not\sqsubseteq A_2$.

If the algorithm terminates in Line 11, we prove that $A_1 \sqsubseteq A_2$ is true, i.e., for any finite word $w \in \alpha^*$, $Pr_{A_1}(w) \leq Pr_{A_2}(w)$. We distinguish three different cases on w:

(1) The node $(\boldsymbol{v}_1, \boldsymbol{v}_2, w)$ is in the non-leaf node set V. Then, it is guaranteed by Line 11 of Algorithm 1 that $Pr_{A_1}(w) = \boldsymbol{v}_1\boldsymbol{\kappa}_1 \leq \boldsymbol{v}_2\boldsymbol{\kappa}_2 = Pr_{A_2}(w)$.

(2) (Note that a node is made as a leaf node because it satisfies either (C1) or (C2).)The word w has a prefix $\bar{w}$ such that the node $(\bar{\boldsymbol{v}}_1, \bar{\boldsymbol{v}}_1, \bar{w})$ is a leaf node and satisfies (C1). We have $Pr_{A_1}(\bar{w}) = \bar{\boldsymbol{v}}_1\boldsymbol{\kappa}_1 = 0$, which implies that $Pr_{A_1}(w) = 0$ since $\bar{w}$ is a prefix of w. Therefore, $Pr_{A_1}(w) = 0 \leq Pr_{A_2}(w)$ is true for any probability value of $Pr_{A_2}(w)$.

(3) The word w has a prefix $\bar{w}$ such that the node $(\bar{\boldsymbol{v}}_1, \bar{\boldsymbol{v}}_1, \bar{w})$ is a leaf node and satisfies (C2). Let $(\hat{\boldsymbol{v}}_1, \hat{\boldsymbol{v}}_2, \hat{w})$ be the parent node of $(\bar{\boldsymbol{v}}_1, \bar{\boldsymbol{v}}_2, \bar{w})$. Following Line 6 of Algorithm 1, we have $\bar{\boldsymbol{v}}_1 = \hat{\boldsymbol{v}}_1\mathbf{P}_1(a)$ and $\bar{\boldsymbol{v}}_2 = \hat{\boldsymbol{v}}_2\mathbf{P}_2(a)$, where $a \in \alpha$. Based on Lemma 5 and Lemma 6, appending any finite word suffix u after $\hat{w}$ yields $Pr_{A_1}(\hat{w}u) \leq Pr_{A_2}(\hat{w}u)$. Since the word w can be represented as the word $\hat{w}$ concatenating with a finite suffix u, we have $Pr_{A_1}(w) \leq Pr_{A_2}(w)$.

Thus, when Algorithm 1 terminates in Line 11, $Pr_{A_1}(w) \leq Pr_{A_2}(w)$ for any finite word $w \in \alpha^*$, i.e. , $A_1 \sqsubseteq A_2$ holds. $\square$

C Proofs for Section 5

Lemma 7. *Given (P, E, T) a closed and consistent observation table, the hypothesis PFA $A = (S, \overline{s}, \alpha, \mathbf{P})$ is well-defined.*

Proof. First, the initial state $\overline{s}$ is well-defined because $row(\varepsilon)$ is always included the set $con(P)$, which contains the rows corresponding to the states set S. Because one element of $row(\varepsilon)$, $T(\varepsilon \cdot \varepsilon) = 1$, so that it is an extreme point in the conical hull and will always be in $con(P)$.

Now we show that the transition matrices $\mathbf{P}$ are well-defined, i.e. , $\sum_j \mathbf{P}(a)[i,j] \in [0,1]$ for all $a \in \alpha$. The row vector of any PFA should satisfy the following: for any $a \in \alpha$, $\dfrac{row(s_i.a)(\varepsilon)}{row(s_i)(\varepsilon)} \in [0,1]$, where $row(u)(\varepsilon)$ is the probability of word u (here we abuse s_i as its corresponding prefix word). This is true since there is a (sub)distribution performing action a from state s_i.

We are to show that $\sum_j \mathbf{P}(a)[i,j] \in [0,1]$. We have that for any j,

$$T(s_j \cdot \varepsilon) = row(s_j)(\varepsilon) \tag{6}$$

and that

$$row(s_i \cdot a) = \sum_j \gamma_j row(s_j) \quad \Longrightarrow \quad row(s_i \cdot a)(\varepsilon) = \sum_j \gamma_j row(s_j)(\varepsilon) \tag{7}$$

Therefore, we have

$$\begin{aligned}\sum_j \mathbf{P}(a)[i,j] &\overset{(\dagger)}{=} \sum_j \gamma_j \cdot \frac{T(s_j \cdot \varepsilon)}{T(s_i \cdot \varepsilon)} \overset{(6)}{=} \sum_j \gamma_j \cdot \frac{row(s_j \cdot \varepsilon)}{row(s_i \cdot \varepsilon)} \\ &\overset{(7)}{=} \sum_j \gamma_j \cdot \frac{row(s_j \cdot \varepsilon)(\varepsilon)}{row(s_i \cdot \varepsilon)(\varepsilon)} \overset{(\ddagger)}{=} \frac{row(s_i.a)(\varepsilon)}{row(s_i)(\varepsilon)} \in [0,1]\end{aligned}$$

$(\dagger)$ is because $\mathbf{P}(a)[i,j] = \gamma_j \cdot \frac{T(s_j \cdot \varepsilon)}{T(s_i \cdot \varepsilon)}$ and $(\ddagger)$ is because $row(s_i \cdot a) = \sum_{s_j \in S} \gamma_j row(s_j)$. □

Lemma 8. *Given (P, E, T) a closed and consistent observation table, the hypothesis PFA $A = (S, \overline{s}, \alpha, \mathbf{P})$ conforms with the function T, i.e. , $\forall u \in (P \cup P \cdot \alpha)$, $\forall e \in E$, $Pr^A(u \cdot e) = T(u \cdot e)$.*

Proof. For simplicity, assume that the index of initial state $\overline{s}$ is 0, so that the initial row vector $\boldsymbol{\iota} = [1, 0, \cdots, 0]$. Also recall that $\boldsymbol{\kappa}$ is defined as a column vector of 1s. For all

$u \in (P \cup P \cdot \alpha)$ and for all $e \in E$, we have

$$\begin{aligned}
Pr^A(u \cdot e) &= \boldsymbol{\iota}\mathbf{P}(u \cdot e)\boldsymbol{\kappa} \\
&= \sum_j \mathbf{P}(u \cdot e)[0, j] \\
&= \sum_j \sum_k \mathbf{P}(u)[0, k]\mathbf{P}(e)[k, j] \\
&\overset{(\dagger 1)}{=} \sum_j \sum_k \gamma^u_{0,k} \frac{T(s_k \cdot \varepsilon)}{T(s_0 \cdot \varepsilon)} \gamma^e_{k,j} \frac{T(s_j \cdot \varepsilon)}{T(s_k \cdot \varepsilon)} \\
&\overset{(\dagger 2)}{=} \sum_k \gamma^u_{0,k} \big(\sum_j \gamma^e_{k,j} T(s_j \cdot \varepsilon) \big) \\
&\overset{(\dagger 3)}{=} \sum_k \gamma^u_{0,k} T(s_k \cdot e) \\
&\overset{(\dagger 4)}{=} T(s_0 \cdot ue) \\
&= T(u \cdot e).
\end{aligned}$$

$(\dagger 1)$ is by the definition of $\mathbf{P}$. $(\dagger 2)$ is because $T(s_0 \cdot \varepsilon) = T(\varepsilon) = 1$. $(\dagger 3)$ and $(\dagger 4)$ are due to $row(s_k \cdot e) = \sum_j \gamma^e_{k,j} row(s_j)$, and $row(s_0 \cdot u) = \sum_k \gamma^u_{0,k} row(s_k)$, respectively. $\square$

D Proofs for Section 6

Lemma 9. *A learner can always update the observation table if the teacher returns a counterexample.*

Proof. We first define counterexamples in these settings.

Definition 12 (Counterexamples for $M_1 \sqsubseteq_w A$). *A counterexample for $M_1 \sqsubseteq_w A$ is a word $c \in \alpha^*$ such that $Pr^{M_1}(c) \geqslant Pr^A(c)$.*

This word c will for sure update the learner's observation table since it violates language inclusion.

Definition 13 (Counterexamples for $pios(A)||M_2 \models \langle G \rangle_{\geqslant p}$). *A counterexample for $pios(A)||M_2 \models \langle G \rangle_{\geqslant p}$ is a set C of paths in $pios(A)||M_2$ such that $Pr^{pios(A)||M_2}(C) \geqslant p$, where $Pr^{pios(A)||M_2}(C) = \sum_{\theta \in C} Pr^{pios(A)||M_2}(\theta)$.*

Note that for any path θ in $pios(A)||M_2$, we can map it onto a path in $pios(A)$, denoted $\theta \restriction_{pios(A)}$. Given a counterexample C, we may derive a set C^A and by $tr(\theta \restriction_{pios(A)}$) we obtain a set of words W^A. And then we check whether each word $tr(\theta \restriction_{pios(A)})$ in W^A can be mapped back onto M_1 as $\bar{w}$. If yes, then a word $\bar{w}$ corresponds to a set of words (regarding τs) in M_1. We denote this set of words as W^{M_1} and the corresponding set of paths as C^{M_1}. We finally form a fragment of M_1 from C^{M_1}, denoted as M_1^C. The whole process can be illustrated as follows, where the first line is on the level of paths or words, and the second line is on the level of their set notations.

$$\begin{array}{ccccccccc} \theta & \rightarrow & \theta \restriction_{pios(A)} & \rightarrow & tr(\theta \restriction_{pios(A)}) & \Rightarrow & \bar{w} & \rightarrow & \pi \\ C & \rightarrow & C^A & \rightarrow & W^A & \supseteq & W^{M_1} & \rightarrow & C^{M_1} \end{array}$$

If C is a real counterexample, then $M_1^C||M_2 \not\models \langle G \rangle_{\geqslant p}$. We stop the learning loop and claim that $M_1||M_2 \not\models \langle G \rangle_{\geqslant p}$. C can also be a spurious counterexample.

Definition 14 (Spurious counterexamples for $pios(A)||M_2 \models \langle G \rangle_{\geqslant p}$). *$C$ is a spurious counterexample if $Pr^{pios(A)||M_2}(C) > 1 - p$, but $M_1^C||M_2 \not\models \langle G \rangle_{\geqslant p}$.*

If this is the case, then M_1^C is not enough to show the violation. The reason is that some of the words in $pios(A)$ cannot be mapped back to M_1, i.e., $W^A \supset W^{M_1}$. For those $w \in W^A \setminus W^{M_1}$, we return w to the learner. Since this word is not in M_1 but currently in A, we can for sure update the observation table. □