Learning-based Compositional Verification for Synchronous Probabilistic Systems

Lu Feng, Tingting Han, Marta Kwiatkowska, David Parker

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 techniques for automated 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. In order to implement this framework, we develop a semi-algorithm to check language inclusion for PFAs and a new active learning method for PFAs. The latter is then used to automatically generate assumptions for compositional verification.

1 Introduction

Probabilistic model checking is a formal verification technique for analysing quantitative properties of systems that exhibit stochastic behaviour. A key challenge in this area is scalability, motivating the development of *compositional* verification methods that decompose the analysis of a large system model into smaller sub-tasks. 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 satisfied, proof rules are used to deduce properties of the overall system.

Several assume-guarantee frameworks for verifying probabilistic systems have been proposed, mainly for models with both probabilistic and nondeterministic behaviour [1,13,10]. The main difficulty when developing such a framework is formulating an appropriate notion of *assumptions* that can support compositional reasoning. Our goal is to develop assume-guarantee techniques for probabilistic model checking 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; and (iii) be amenable to automatic generation.

One promising direction is the framework of [13] (and its extensions in [10,9]). In [13], assumptions are *probabilistic safety properties* (e.g. "event A always occurs before event B with probability at least 0.98") and [10] 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, [9] shows how to *automatically* generate probabilistic safety property assumptions [13] 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 property

assumptions [13] can only capture a limited amount of information about a component, restricting the cases where assume-guarantee reasoning can be applied. The framework of [13] 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 verify it compositionally.

This paper proposes novel techniques for compositional probabilistic verification in which assumptions are *probabilistic finite automata* (PFAs) [15]. Unlike [13,10], our approach *is* complete. Furthermore, as in [10], we use learning to automatically generate assumptions. PFAs represent weighted languages, mapping finite words to probabilities. In our framework, an assumption about a system component M is represented by a PFA that gives upper 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 [16]. So, in the present work, we restrict our attention to *fully probabilistic* systems. To do so, we model components as *probabilistic I/O systems* (PIOSs), which, when combined through *synchronous* parallel composition, result in a (fully probabilistic) discrete-time Markov chain (DTMC). The relation between a PIOS M and a PFA A representing an assumption about M is captured by *weak language inclusion*. Based on this, we give an asymmetric proof rule for verifying probabilistic safety properties on a DTMC composed of two PIOSs.

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 [18,7], checking language *inclusion* is undecidable [5]. We propose a semi-algorithm, inspired by [18], to check language inclusion; in the case where the check fails, a minimal counterexample is produced.

We also develop a novel technique for *learning* PFAs, which we use to automatically generate assumptions for our framework. Our algorithm, like L*, is based on *active learning*, posing queries in an interactive fashion about the PFA to be generated. Several active PFA learning algorithms exist [12,17,4] but are not suitable for our needs: [12] applies to a restricted class of PFAs, [17] needs to know the size of the PFA in advance, and [4] actually learns multiplicity automata, which may contain negative values.

Full version: For an extended version of this paper, including additional details, explanations and running examples, experimental results and proofs, see [8].

2 Preliminaries

We first briefly describe *probabilistic finite automata* and *discrete-time Markov chains*. We use $SDist(S)$ to denote the set of probability *sub-distributions* over set S, η_s for the point distribution on $s{\in}S$, and $\mu_1{\times}\mu_2$ for the product distribution of μ_1 and μ_2.

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 $Pr^A : \alpha^* \to [0,1]$ giving the probability of accepting each finite word $w \in \alpha^*$. Intuitively, the probability $Pr^A(w)$ for a word $w = a_1 \cdots a_n$ is

determined by tracing paths through A that correspond to w, with $\mathbf{P}(a)[s, s']$ giving the probability to move from s to s' on reading a. More precisely, we let $\boldsymbol{\iota}$ be an S-indexed 0-1 row vector with $\boldsymbol{\iota}[s] = 1$ if and only if $s = \overline{s}$, $\boldsymbol{\kappa}$ be an S-indexed column vector of 1s and $\mathbf{P}(w) = \mathbf{P}(a_1) \cdots \mathbf{P}(a_n)$. Then, we define $Pr^A(w) = \boldsymbol{\iota}\mathbf{P}(w)\boldsymbol{\kappa}$.

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. $Pr^{A_1}(w) = Pr^{A_2}(w)$).*

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 $\delta : S \times (\alpha \cup \{\tau\}) \to \mathit{SDist}(S)$ is a (partial)* probabilistic transition function*, such that, for any s, $\delta(s, a)$ is defined for at most one $a \in \alpha \cup \{\tau\}$.*

If $\delta(s, a) = \mu$, the DTMC can make a transition, labelled with action a, and move to state s' with probability $\mu(s')$. We denote such transitions by $s \xrightarrow{a} \mu$ (or $s \xrightarrow{a} s'$). The DTMC deadlocks when $\delta(s, a)$ is not defined for any a, which we denote by $s \not\to$. We use action label τ to denote a "silent' (or "internal") transition. A (finite or infinite) path through D is a sequence of transitions $\theta = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ with $s_0 = \overline{s}$.

In this paper, we consider *probabilistic safety properties* $\langle G \rangle_{\geqslant p}$, where G is a *regular safety property* [3], defining a set of "good" executions, and $p \in [0, 1]$ is a probability bound. Model checking $\langle G \rangle_{\geqslant p}$ reduces to solving a linear equation system [3].

3 Assume-Guarantee for Synchronous Probabilistic Systems

We now define a compositional verification framework for fully probabilistic systems. Components are modelled by *probabilistic I/O systems* (PIOSs). These exhibit (input) nondeterminism but, when composed *synchronously* in parallel, result in a DTMC.

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 $\delta : S \times (\alpha \cup \{\tau\}) \to \mathit{SDist}(S)$ satisfy the following two conditions: (i) 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; (ii) the set $\mathit{enab}(s) \subseteq \alpha \cup \{\tau\}$ of* enabled *actions for each state s (i.e. the actions a for which $\delta(s, a)$ is defined) satisfies either $|\mathit{enab}(s)| = 1$ if $\mathit{enab}(s) \in \alpha^O \cup \alpha^H \cup \{\tau\}$ or $\mathit{enab}(s) = \alpha^{I,i}$ for some input action bundle $\alpha^{I,i}$.*

From any state s of a PIOS M, there is either a single transition with an output, hidden or τ action, or k transitions, each with one action from a particular bundle $\alpha^{I,i}$ comprising k input actions. Transitions and paths in PIOSs are defined as for DTMCs. 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 $Pr^M_\tau(w) = \sum_{w=st(w')} Pr^M(w')$.

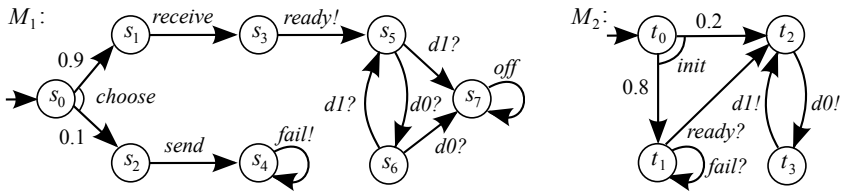

Fig. 1. Running example: two PIOSs M_1 and M_2.

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: $\alpha_1^{I,1} = \{d0, d1\}$, $\alpha_2^{I,1} = \{ready, fail\}$.

Given PIOSs M_1, M_2 with alphabets α_1, α_2, we say 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 define their parallel composition as follows.

Definition 5 (Parallel composition). *The* parallel composition *of 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 $\alpha = \alpha^H = \alpha_1^I \cup \alpha_1^O \cup \big((\alpha_1^H \cup \{\bot\}) * (\alpha_2^H \cup \{\bot\})\big)$ and, for $b_i \in \alpha_i^H \cup \tau$ 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: (i) $s_1 \xrightarrow{a} \mu_1$, $s_2 \xrightarrow{a} \mu_2$, $\gamma = a$; (ii) $s_1 \xrightarrow{b_1} \mu_1$, $s_2 \xrightarrow{b_2} \mu_2, \gamma = b_1 * b_2$; (iii) $s_1 \xrightarrow{b_1} \mu_1$, $s_2 \xrightarrow{a}$ (or $s_2 \not\to$), $\mu_2 = \eta_{s_2}$, $\gamma = b_1 * \bot$; (iv) $s_1 \xrightarrow{a}$ (or $s_1 \not\to$), $s_2 \xrightarrow{b_2} \mu_2$, $\mu_1 = \eta_{s_1}$, $\gamma = \bot * b_2$.*

Notice PIOS $M_1 \| M_2$ has only τ or hidden actions and can thus be considered a DTMC.

We next introduce our notion of *assumptions* about PIOSs, for which we use a specific class of PFAs and *weak language inclusion*, which relaxes the definition of language inclusion for PFAs introduced earlier by ignoring τ actions.

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:*

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

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 (see Section 4) by the following proposition.

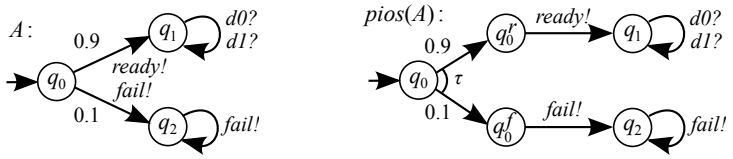

Fig. 2. Assumption A and its PIOS conversion $pios(A)$.

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

We will also need to perform a conversion in the opposite direction, translating an assumption PFA A into a (weak language) equivalent PIOS, which we denote $pios(A)$.

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 $S' = S \uplus \{s^a | s \in S, a \in \alpha^H \cup \alpha^O\} \uplus \{s^i | s \in S, 1 \leqslant i \leqslant m\}$ 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 $\delta(s^i, a)(s') = p \cdot \frac{p^{\max}(s)}{p_i^{\max}(s)}$.*

Example 2. Consider PIOS M_1 from Example 1. Fig. 2 shows a valid assumption A for M_1 (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.

Now, we describe how to perform compositional verification using our framework. We focus on verifying $\langle G \rangle_{\geqslant p}$ on a DTMC $M_1 \| M_2$ where M_i are PIOSs. For simplicity, we will assume that the property refers only to input/output actions of M_1 and M_2 and assume that all hidden actions of M_1 and M_2 have been renamed as τ actions, which affects neither the parallel composition $M_1 \| M_2$ nor the probability of satisfying G.

An *assume-guarantee triple* $\langle A \rangle\ M\ \langle G \rangle_{\geqslant p}$ means "whenever component M is part of a system satisfying the assumption A, the system is guaranteed to satisfy $\langle G \rangle_{\geqslant p}$".

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).

Proposition 2. *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 $pios(A) \| M \models \langle G \rangle_{\geqslant p}$.*

Finally, we give an asymmetric assume-guarantee proof rule (in the style of those from [14,13]) for verifying a system $M_1 \| M_2$ compositionally.

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

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

Thus, 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 2, 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.

Completeness. Our framework is *complete* in the sense that, if $M_1 \| M_2 \models \langle G \rangle_{\geqslant p}$, we can always find an assumption A to apply Theorem 1 by converting M_1 to a PFA.

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 the problem is undecidable (see [8]).

Input: PFAs A_1 and A_2 over the same alphabet α.
Output: **true** if $A_1 \sqsubseteq A_2$; or **false** and a cex $w' \in \alpha^*$.
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 cex w'
7: **else if** $(\boldsymbol{v}_1', \boldsymbol{v}_2', w')$ does not satisfy (C1), (C2) **then**
8: add $(\boldsymbol{v}_1', \boldsymbol{v}_2', w')$ to the tail of *queue*
9: $V := V \cup \{(\boldsymbol{v}_1', \boldsymbol{v}_2', w')\}$
10: **return true**

Fig. 3. Semi-algorithm for deciding PFA language inclusion

Fig. 3 shows the semi-algorithm to decide 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. Inspired by the language equivalence decision algorithm in [18], 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 $\boldsymbol{\upsilon}_i\boldsymbol{\kappa}_i$ 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 Fig. 3, we use a *queue* of tree nodes, which expands the tree in breadth-first order. In addition, we maintain a set V of non-leaf nodes, which initially only contains the root. The main difference between our method and [18] is that we adopt different criteria to decide when to add a node to the non-leaf set V. In [18], 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 $(\boldsymbol{\upsilon}_1, \boldsymbol{\upsilon}_2, w)$ from the head of *queue*. We then expand the tree by appending a set of its child nodes $(\boldsymbol{\upsilon}'_1, \boldsymbol{\upsilon}'_2, w')$, where $\boldsymbol{\upsilon}'_1 := \boldsymbol{\upsilon}_1\mathbf{P}_1(a)$, $\boldsymbol{\upsilon}'_2 := \boldsymbol{\upsilon}_2\mathbf{P}_2(a)$ and $w' := wa$ for all actions $a \in \alpha$. If there is a node $(\boldsymbol{\upsilon}'_1, \boldsymbol{\upsilon}'_2, w')$ such that $Pr_1(w') = \boldsymbol{\upsilon}'_1\boldsymbol{\kappa}_1 > \boldsymbol{\upsilon}'_2\boldsymbol{\kappa}_2 = Pr_2(w')$, 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 $(\boldsymbol{\upsilon}'_1, \boldsymbol{\upsilon}'_2, w')$ (i.e. make it a leaf node) by seeing if it satisfies either of the following two criteria: (C1) $\boldsymbol{\upsilon}'_1\boldsymbol{\kappa}_1 = 0$; (C2) There exist $|V|$ non-negative rational numbers ρ^i such that, for all $(\boldsymbol{\upsilon}^i_1, \boldsymbol{\upsilon}^i_2, w^i) \in V$, $\boldsymbol{\upsilon}'_1 \leq \sum_{0\leq i<|V|} \rho^i\boldsymbol{\upsilon}^i_1$ and $\boldsymbol{\upsilon}'_2 \geq \sum_{0\leq i<|V|} \rho^i\boldsymbol{\upsilon}^i_2$, where $\leq$ and $\geq$ denote pointwise comparisons between vectors.

Criterion (C1) is included because it is never possible to find a counterexample word with accepting probability less than $\boldsymbol{\upsilon}'_1\boldsymbol{\kappa}_1 = 0$. Criterion (C2) is included because any node satisfying it would guarantee $\boldsymbol{\upsilon}'_1\boldsymbol{\kappa}_1 \leq \boldsymbol{\upsilon}'_2\boldsymbol{\kappa}_2$; 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. 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 if *queue* becomes empty, concluding that $A_1 \sqsubseteq A_2$.

Correctness and termination. The correctness of the semi-algorithm in Fig. 3 is shown formally in [8]. A guarantee of termination, on the other hand, cannot be expected due to the undecidability of the underlying problem.

5 L*-Style Learning for PFAs

In this section, we propose a novel method to learn a PFA for a target weighted language generated by an unknown PFA. It works in a similar style to the well-known L* algorithm [2] for learning regular languages: it constructs an *observation table* (of acceptance probabilities for each word) based on two types of *queries* posed to a *teacher*. *Membership queries* ask the probability of accepting a particular word in the target PFA; *equivalence queries* ask whether a hypothesised PFA yields exactly the target language.

Fig. 4 shows the learning algorithm. It builds an observation table (P, E, T), where P is a finite, non-empty, prefix-closed set of words, E is a finite, non-empty, 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 the target language ($\cdot$ denotes concatenation over sets). 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. The value $T(u{\cdot}e)$ of the entry at row u and column e is the acceptance probability of the word $u{\cdot}e$. We use $row(u)$ to represent the $|E|$-dimensional row vector in the table labelled by the prefix $u \in (P \cup P{\cdot}\alpha)$.

Inspired by [4], which gives an L*-style algorithm for learning multiplicity automata, we define the notions of *closed* and *consistent* observation tables by estab-

Input: The alphabet α 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 ε is the empty word
2: fill T by asking membership queries for ε 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

Fig. 4. L*-style learning algorithm for PFAs

lishing linear dependencies between row vectors. 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 $row(u \cdot a) = \sum_{u_i \in P} \phi_i row(u_i)$ and *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 $\forall a \in \alpha, e \in E. \sum_{u_i \in P} \psi_i T(u_i \cdot a \cdot e) = 0$. The need for coefficients to be non-negative (for *closed*) is a stronger condition than in [4].

As shown in Fig. 4, the observation table is filled with the results of membership queries until it is both closed and consistent. At each step, if (P, E, T) is not closed (resp. consistent), then the algorithm finds $u \in P, a \in \alpha$ (resp. $a \in \alpha, e \in E$) that make it not closed (resp. consistent), according to the definitions above, and adds $u.a$ (resp. $a.e$) to the table. When (P, E, T) is closed and consistent, the learning algorithm builds a hypothesis PFA A (see below) and poses an equivalence query. If the teacher answers "no" (that A does not yield the target language), a counterexample $c \in \alpha^*$ is given, for which $Pr^A(c)$ is incorrect. 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.

Construction of a hypothesis PFA $A = (S, \overline{s}, \alpha, \mathbf{P})$, from a closed and consistent table (P, E, T), proceeds as follows. First, we find 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 are non-negative rational coefficients λ_i such that, for all $u \in P$, $row(u) = \sum_{v_i \in con(P)} \lambda_i row(v_i)$. The set of states in the PFA is then $S = \{s_0, \ldots, s_{n-1}\}$, where each state s_i corresponds to a row vector in $\{row(v) | v \in con(P)\}$ and the initial state $\overline{s}$ corresponds to $row(\varepsilon)$. To obtain $\mathbf{P}(a)$ for each $a \in \alpha$, we compute, for $s_i \in S$, rational coefficients γ_j such that $row(s_i \cdot a) = \sum_{s_j \in S} \gamma_j row(s_j)$ and then define $\mathbf{P}(a)[s_i, s_j] := \gamma_j \cdot (T(s_j \cdot \varepsilon)/T(s_i \cdot \varepsilon))$.

Correctness and termination. When the learning algorithm terminates, it returns a correct PFA, as guaranteed by the equivalence query check. Unfortunately, we cannot prove the termination of our method. For L*, the corresponding proof uses the existence of a unique minimal DFA for a regular language. 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 [6], converting a multiplicity automaton to a PFA (even for the subclass that define stochastic languages) is not always possible.

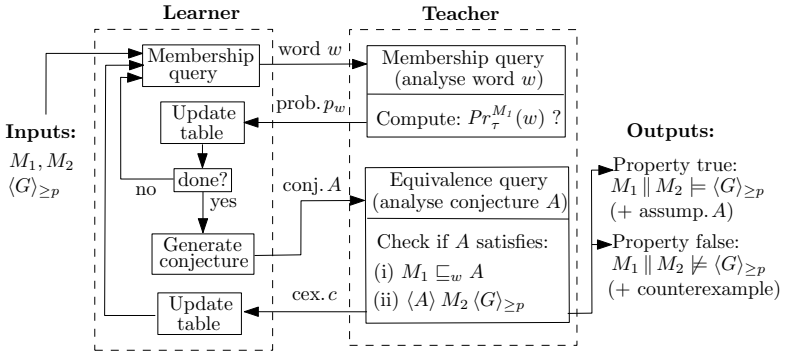

Fig. 5. L*-style PFA learning loop for probabilistic assumption generation.

6 Learning Assumptions for Compositional Verification

Finally, we build upon 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. 5 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-PIOS) 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 $Pr_\tau^{M_1}(w)$ of word w in M_1. It answers *equivalence queries* (about conjectured PFA A) by checking if A satisfies both premises of rule (ASYM-PIOS): (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 2, 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* [11] C: a set of paths showing $pios(A)||M_2 \not\models \langle G \rangle_{\geqslant p}$. Following the same approach as [9], 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 extended version of this paper [8].

From the arguments above, we can show that, when the learning loop terminates, it always yields a correct result. It should be pointed out, though, that 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. Furthermore, weak language inclusion checks use the semi-algorithm of Section 4, which is not guaranteed to terminate.

7 Implementation and Results

We have implemented the PFA language inclusion check from Section 4, the PFA learning algorithm from Section 5 and the assumption-generation loop described in Section 6. Based on these, we have built a prototype tool that performs fully-automated assume-guarantee verification, as described in Section 3. Due to space limitations, we refer the reader to [8] for further details of this implementation, as well as experimental results from its application to several benchmark case studies.

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)
6. Denis, F., Esposito, Y.: Learning classes of probabilistic automata. In: COLT (2004)
7. Doyen, L., Henzinger, T.A., Raskin, J.F.: Equivalence of labeled Markov chains. Int. J. Found. Comput. Sci. 19(3), 549–563 (2008)
8. Feng, L., Han, T., Kwiatkowska, M., Parker, D.: Learning-based compositional verification for synchronous probabilistic systems. Tech. Rep. RR-11-05, Department of Computer Science, University of Oxford (2011)
9. Feng, L., Kwiatkowska, M., Parker, D.: Compositional verification of probabilistic systems using learning. In: Proc. QEST'10. pp. 133–142. IEEE CS Press (2010)
10. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: Proc. TACAS'11 (2011)
11. Han, T., Katoen, J.P., Damman, B.: Counterexample generation in probabilistic model checking. IEEE Trans. Software Eng. 35(2), 241–257 (2009)
12. de la Higuera, C., Oncina, J.: Learning stochastic finite automata. In: ICGI (2004)
13. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: Proc. TACAS'10 (2010)
14. 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)
15. Rabin, M.: Probabilistic automata. Information and Control 6, 230–245 (1963)
16. Segala, R.: Modelling and Verification of Randomized Distributed Real Time Systems. Ph.D. thesis, Massachusetts Institute of Technology (1995)
17. Tzeng, W.G.: Learning probabilistic automata and Markov chains via queries. Mach. Learn. 8, 151–166 (1992)
18. Tzeng, W.G.: A polynomial-time algorithm for the equivalence of probabilistic automata. SIAM J. Comput. 21(2), 216–227 (1992)