# Compositional Verification of Probabilistic Systems using Learning

Lu Feng, Marta Kwiatkowska, David Parker

# Compositional Verification of Probabilistic Systems using Learning

Lu Feng, Marta Kwiatkowska, David Parker
*Oxford University Computing Laboratory, Parks Road, Oxford, OX1 3QD*
*Email: {lu.feng, marta.kwiatkowska, david.parker}@comlab.ox.ac.uk*

***Abstract*—We present a fully automated technique for compositional verification of probabilistic systems. Our approach builds upon a recently proposed assume-guarantee framework for probabilistic automata, in which assumptions and guarantees are probabilistic safety properties, represented using finite automata. A limitation of this work is that the assumptions need to be created manually. To overcome this, we propose a novel learning technique based on the L* algorithm, which automatically generates probabilistic assumptions using the results of queries executed by a probabilistic model checker. Learnt assumptions either establish satisfaction of the verification problem or are used to generate a probabilistic counterexample that refutes it. In the case where an assumption cannot be generated, lower and upper bounds on the probability of satisfaction are produced. We illustrate the applicability of the approach on a range of case studies.**

***Keywords*-Compositional verification; probabilistic model checking; probabilistic automata; learning.**

## I. Introduction

Probabilistic model checking offers a powerful set of techniques for formally analysing quantitative properties of systems that exhibit stochastic behaviour. Examples of systems to which these methods have been applied include randomised communication protocols, probabilistic security protocols and randomised distributed algorithms. As with any formal verification technique, the principal challenge of probabilistic model checking is scalability. Faithful models of large-scale complex systems can quickly exceed the capabilities of today's techniques and tools.

A promising direction to improve scalability is the use of *compositional* techniques, where the verification process is decomposed into a separate analysis for each component of the system being verified. A popular approach to this is the *assume-guarantee* paradigm. For example, the verification of a property $G$ on a two-component system $M_1 \| M_2$ can be decomposed into two separate problems: (i) checking that, under the assumption that property $A$ holds, $M_2$ is guaranteed to satisfy $G$, denoted $\langle A \rangle\, M_2\, \langle G \rangle$; and (ii) checking that $M_1$ satisfies the assumption $A$ (under any context).

This paper focuses on compositional verification for *probabilistic automata*, which are a natural and widely used modelling formalism for systems that exhibit both probabilistic and nondeterministic behaviour. Devising compositional verification techniques for probabilistic systems requires considerable care. Despite several important developments regarding the underlying models, formalisms and theory (see in particular [1], [2], [3], [4]), there has been a lack of progress in terms of practically implementable techniques.

An exception to this is the recent work of [5] which proposed an assume-guarantee verification technique for probabilistic automata and applied it to several large case studies. This is based on assume-guarantee queries of the form $\langle A \rangle_{\geq p_A}\, M\, \langle G \rangle_{\geq p_G}$, where the assumptions $\langle A \rangle_{\geq p_A}$ and guarantees $\langle G \rangle_{\geq p_G}$ are probabilistic safety properties. Informally, this means that "whenever $M$ is part of a system satisfying property $A$ with probability at least $p_A$, then the system will guarantee property $G$ with probability at least $p_G$". A limitation of this work, however, is that it requires non-trivial manual effort to create appropriate assumptions.

In this paper, we address this limitation by proposing a fully automated approach to the generation of such assumptions, based on *learning* techniques. Learning, and in particular the well-known L* learning algorithm [6], have proved to be well suited to generating assumptions for compositional verification of non-probabilistic systems [7] and are currently attracting considerable interest in the verification community.

We propose a novel learning technique, based on L*, for generating *probabilistic* assumptions. Given components $M_1, M_2$ and probabilistic safety property $\langle G \rangle_{\geq p_G}$, our framework attempts to build an assumption $\langle A \rangle_{\geq p_A}$ that can be used to prove that $M_1 \| M_2$ satisfies $\langle G \rangle_{\geq p_G}$, without constructing the full model $M_1 \| M_2$.

Like in [7], this is done by generating a series of possible assumptions which are analysed by a (probabilistic) model checker. The results of this analysis, in the form of (probabilistic) counterexamples, are used to guide the learning process by refining the current assumption. During this process, we may also discover a counterexample illustrating that $M_1 \| M_2$ does *not* satisfy $\langle G \rangle_{\geq p_G}$.

Because the assume-guarantee framework of [5] is incomplete, there may be no suitable assumption that can be learnt. To address this problem, we adopt a *quantitative* approach: our learning technique also generates *lower and upper bounds* on the (minimum) probability of satisfying $G$.

We have implemented our technique and successfully applied it to several large case studies. This includes instances where the entire process of learning an assumption and then applying compositional verification is more efficient than conventional, non-compositional verification.

**Related work.** As mentioned above, there has been significant progress in developing modelling formalisms and proof techniques for compositional verification of probabilistic systems [1], [2], [3], [4], including application of these to (manual) verification of large, complex randomised algorithms [8]. These do not, however, focus on practical implementations. This paper builds on the probabilistic assume-guarantee framework of [5], which was shown to be efficiently implementable and capable of outperforming non-compositional techniques on several large case studies.

Our approach is also inspired by the work of Giannakopoulou, Pasareanu et al. (see e.g. [7]), which learns assumptions for assume-guarantee verification of *non-probabilistic* systems, i.e. labelled transition systems. This paper is the first to learn *probabilistic* assumptions for compositional verification.

Finally, we mention work on the related problem of *synthesis* [9], [10], i.e. constructing probabilistic systems according to a formal specification. However, such techniques have not yet been implemented.

**Paper structure.** In the next section, we provide some background on probabilistic automata, probabilistic assume-guarantee reasoning and the L* algorithm. Section III introduces the generation of probabilistic counterexamples in the context of probabilistic assume-guarantee verification. Section IV presents our learning framework and Section V describes experimental results from its implementation on a range of case studies. Section VI concludes the paper.

## II. Background

In the following, we use $Dist(S)$ to denote the set of all discrete probability distributions over a set $S$.

### A. Probabilistic Automata

Probabilistic automata [1], [11] model systems whose behaviour is both probabilistic and nondeterministic.

**Definition 1.** A *probabilistic automaton* (PA) is a tuple $M = (S, \overline{s}, \alpha_M, \delta_M)$ where $S$ is a set of states, $\overline{s} \in S$ is an initial state, $\alpha_M$ is an alphabet, and $\delta_M \subseteq S \times (\alpha_M \cup \{\tau\}) \times Dist(S)$ is a probabilistic transition relation.

In a state $s$ of a PA $M$, one or more *transitions*, denoted $s \xrightarrow{a} \mu$, are available, where $a \in \alpha_M \cup \{\tau\}$ is an action label, $\mu$ is a probability distribution over states and $(s, a, \mu) \in \delta_M$. Each step of an execution of the PA first makes a *nondeterministic* choice between available transitions from $s$, and then a *probabilistic* choice of successor state according to the selected distribution $\mu$. A *path* through $M$ is a (finite or infinite) sequence $s_0 \xrightarrow{a_0,\mu_0} s_1 \xrightarrow{a_1,\mu_1} \cdots$ such that $s_0 = \overline{s}$ and, for each $i \geq 0$, $s_i \xrightarrow{a_i} \mu_i$ is a transition and $\mu_i(s_{i+1}) > 0$. We let $Path_M$ denote the set of paths in $M$.

For a path $\pi$, its *trace* $tr(\pi)$ is the sequence $\langle a_0, a_1, \ldots \rangle$ of its action labels, after removal of any "internal" $\tau$ actions. For a trace $t$, we denote by $t{\restriction_\alpha}$ the projection of $t$ onto a subset $\alpha$ of its alphabet. We extend $tr$ and $\restriction$ to sets of paths and traces in the obvious way.

It is also possible to allow the transitions in a PA to be sub-distributions (i.e. sum to less than 1), with the interpretation that a PA can, with some probability, choose to deadlock in some states [1]. In this paper, we will refer to such models as *sub-stochastic PAs*.

To reason about PAs, we use the notion of *adversaries* (sometimes called schedulers or strategies), which resolve the nondeterministic choices in a model, based on its execution history. Formally, an adversary $\sigma$ maps any finite path to a distribution over the available transitions in the last state of the path. There are several important classes of adversaries. An adversary is *deterministic* if it always selects a single transition (with probability 1); otherwise it is *randomised*. It is *memoryless* if its choice depends only on the current state, rather than the full execution history; otherwise it is *history dependent*. A special class of the latter are *finite-memory* adversaries, which store information about the history in a finite-state automaton (see e.g. [12] for a precise definition).

We denote by $Path^\sigma_M$ the set of all paths through $M$ when controlled by adversary $\sigma$, and by $Adv_M$ the set of all possible adversaries for $M$. Under an adversary $\sigma$, we define a probability space $Pr^\sigma_M$ over the set of paths $Path^\sigma_M$, which captures the (purely probabilistic) behaviour of $M$ under $\sigma$.

The behaviour of a PA $M$ with states $S$ under a deterministic, memoryless adversary $\sigma$ can be represented by another PA with states $S$ in which each $s \in S$ contains only the choice made by $\sigma$ in $s$. In similar fashion, if $\sigma$ is a deterministic, finite-memory adversary, the behaviour of $M$ under $\sigma$ can be represented by a PA with states $S \times Q$ where $Q$ is the set of states of $\sigma$'s automaton.

Finally, we mention two other required notions for PAs: *parallel composition* [1] and *alphabet extension* [5]. We use the standard definition of parallel composition of PAs $M_1$ and $M_2$, denoted $M_1 \| M_2$, in which $M_1$ and $M_2$ synchronise over all common actions [1]. For space reasons, we omit the full definition. The alphabet extension of PA $M$ with alphabet $\alpha$, denoted $M[\alpha]$, is obtained from $M$ by adding an $a$-labelled self-loop to every state for each $a \in \alpha \backslash \alpha_M$.

### B. Model Checking Probabilistic Automata

In this paper, we focus on action-based properties of PAs, defined in terms of their traces. In particular, we focus on properties described by regular languages over actions.

A *regular safety property* $G$ represents a set of infinite words, denoted $\mathcal{L}(G)$, that is characterised by a regular language of *bad prefixes*, finite words of which any (possibly empty) extension is not in $\mathcal{L}(G)$. In practice, we define the set of bad prefixes for $G$ by a (complete) deterministic finite automaton (DFA) $G^{err}$ over alphabet $\alpha_G$. The language $\mathcal{L}(G)$ is defined as $\mathcal{L}(G) = \{w \in (\alpha_G)^\omega \mid \text{no prefix of } w \text{ is in } \mathcal{L}(G^{err})\}$ where $\mathcal{L}(G^{err}) \subseteq (\alpha_G)^+$ is the regular language for DFA $G^{err}$.

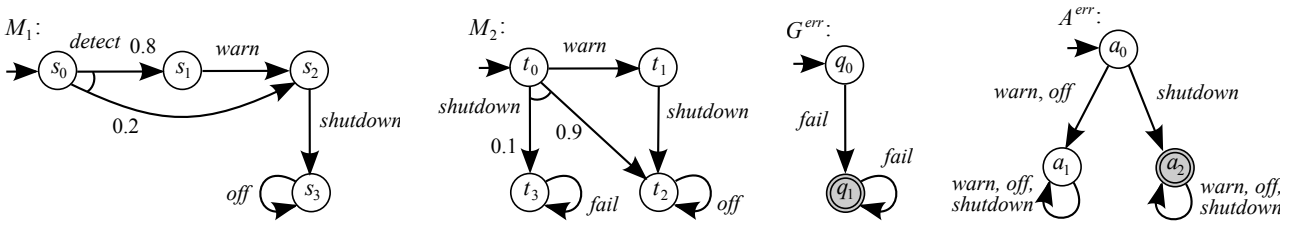


Figure 1. Two probabilistic automata $M_1, M_2$ and the DFAs for two safety properties $G, A$ (example taken from [5])

Given a PA $M$ and regular safety property $G$ with $\alpha_G \subseteq \alpha_M$, an infinite path $\pi$ of $M$ satisfies $G$, denoted $\pi \models G$, if $tr(\pi)\restriction_{\alpha_G} \in \mathcal{L}(G)$. For a finite path $\pi'$ of $M$, we say that $\pi' \models G$ if some infinite path $\pi$ of which $\pi'$ is a prefix satisfies $G$. For an adversary $\sigma \in Adv_M$, we define the probability of $M$ under $\sigma$ satisfying $G$ as:

$$Pr^{\sigma}_{M}(G) \stackrel{\text{def}}{=} Pr^{\sigma}_{M}\{\pi \in Path^{\sigma}_{M} \mid \pi \models G\}$$

We then define the minimum probability of satisfying $G$ as:

$$Pr^{\min}_{M}(G) \stackrel{\text{def}}{=} \inf_{\sigma \in Adv_M} Pr^{\sigma}_{M}(G)$$

A *probabilistic safety property* $\langle G \rangle_{\geq p_G}$ comprises a regular safety property $G$ and a rational probability bound $p_G$. We say that a PA $M$ satisfies the property, denoted $M \models \langle G \rangle_{\geq p_G}$, if the probability of satisfying $G$ is at least $p_G$ for any adversary:

$$\begin{array}{rcl} M \models \langle G \rangle_{\geq p_G} & \Leftrightarrow & \forall \sigma \in Adv_M \,.\, Pr^{\sigma}_{M}(G) \geq p_G \\ & \Leftrightarrow & Pr^{\min}_{M}(G) \geq p_G \,. \end{array}$$

Practical examples of safety properties for PAs include "the probability of a failure occurring is at most 0.01", "event *warning* always occurs before event *shutdown* with probability at least 0.98", and "the probability of terminating within $k$ time-units is at least 0.75".

Model checking a probabilistic safety property $\langle G \rangle_{\geq p_G}$ on a PA $M$ requires computation of the minimum probability $Pr^{\min}_{M}(G)$. This reduces, using standard automata-based techniques for probabilistic model checking, to calculating the maximum probability of reaching a set of accepting states in the PA $M \otimes G^{err}$ formed as the product of PA $M$ and the DFA $G^{err}$ representing $G$ (see [5] for details).

To check probabilistic safety properties on PAs, it suffices to consider deterministic, finite-memory adversaries, that is to say there always exists such an adversary $\sigma$ for which $Pr^{\sigma}_{M}(G) = Pr^{\min}_{M}(G)$.

Note also that, in the special case of *qualitative* safety properties, the following holds:

$$M \models \langle G \rangle_{\geq 1} \quad \Leftrightarrow \quad \forall \pi \in Path_M \,.\, \pi \models G$$

which can be checked with a simple graph analysis.

**Example 1.** Figure 1 shows two PAs $M_1$ and $M_2$ (taken from [5]). $M_1$ is a controller that issues a *shutdown* command to a device, modelled by $M_2$. Prior to this, $M_1$ tries to send a *warn* signal. If the attempt is not successful, there is a chance that $M_2$ will not shut down correctly, resulting in a failure. Figure 1 also shows the DFA $G^{err}$ for a safety property $G$: "action *fail* never occurs". It can be verified that $M_1 \| M_2 \models \langle G \rangle_{\geq 0.98}$ (since the maximum probability of reaching a $q_1$ state in $(M_1 \| M_2) \otimes G^{err}$ is $0.2 \cdot 0.1 = 0.02$).

### C. Probabilistic Assume-Guarantee Reasoning

In this paper, we use the compositional verification framework of [5]. This is based on the assume-guarantee paradigm, where assumptions and guarantees are probabilistic safety properties. The key idea is the notion of *probabilistic assume-guarantee triples* of the form $\langle A \rangle_{\geq p_A} M \langle G \rangle_{\geq p_G}$, in which $\langle A \rangle_{\geq p_A}$ and $\langle G \rangle_{\geq p_G}$ are probabilistic safety properties and $M$ is a PA. Informally, the meaning of this is "whenever $M$ is part of a system satisfying $A$ with probability at least $p_A$, then the system will satisfy $G$ with probability at least $p_G$". Formally:

**Definition 2.** If $\langle A \rangle_{\geq p_A}$ and $\langle G \rangle_{\geq p_G}$ are probabilistic safety properties, $M$ is a PA and $\alpha_G \subseteq \alpha_A \cup \alpha_M$, then:

$$\langle A \rangle_{\geq p_A} M \langle G \rangle_{\geq p_G}$$
$$\Leftrightarrow$$
$$\forall \sigma \in Adv_{M[\alpha_A]} . \left( Pr^{\sigma}_{M[\alpha_A]}(A) \geq p_A \Rightarrow Pr^{\sigma}_{M[\alpha_A]}(G) \geq p_G \right)$$

where $M[\alpha_A]$ is, as described in Section II-A, $M$ with its alphabet extended to include $\alpha_A$.

Determining whether an assume-guarantee triple holds reduces to *multi-objective* probabilistic model checking [5], [13], which can be solved efficiently by solving an LP problem. In the absence of an assumption (denoted by $\langle true \rangle$) checking a triple reduces to normal model checking:

$$\langle true \rangle \, M \, \langle G \rangle_{\geq p_G} \Leftrightarrow M \models \langle G \rangle_{\geq p_G}$$

This reduction holds because $\langle G \rangle_{\geq p_G}$ is a safety property. We also note that, for the case of *qualitative* assumptions of the form $\langle A \rangle_{\geq 1}$, the LP problem can be bypassed and checking the triple $\langle A \rangle_{\geq 1} \, M \, \langle G \rangle_{\geq p_G}$ also reduces to normal (probabilistic) model checking.

Using these definitions, [5] presents several proof rules for compositional probabilistic verification. In this paper, we focus on the following *asymmetric* rule (ASYM). For PAs $M_1, M_2$, probabilistic safety properties $\langle A \rangle_{\geq p_A}, \langle G \rangle_{\geq p_G}$ such that $\alpha_A \subseteq \alpha_{M_1}$ and $\alpha_G \subseteq \alpha_{M_2} \cup \alpha_A$:

$$\frac{\begin{array}{c} \langle true \rangle \, M_1 \, \langle A \rangle_{\geq p_A} \\ \langle A \rangle_{\geq p_A} \, M_2 \, \langle G \rangle_{\geq p_G} \end{array}}{\langle true \rangle \, M_1 \parallel M_2 \, \langle G \rangle_{\geq p_G}} \qquad \textit{(ASYM)}$$

Thus, given an appropriate probabilistic assumption $\langle A \rangle_{\geq p_A}$, verifying that $M_1 \| M_2 \models \langle G \rangle_{\geq p_G}$ can be done compositionally by model checking a safety property on $M_1$ (for premise 1) and an assume guarantee triple on $M_2$ (for premise 2).

**Example 2.** We return to the PAs $M_1, M_2$ and safety property $G$ from Figure 1. As stated in Example 1, we have that $M_1 \| M_2 \models \langle G \rangle_{\geq 0.98}$. This can also be checked compositionally, using rule (ASYM) and the probabilistic safety property $\langle A \rangle_{\geq 0.8}$, for which the DFA $A^{err}$ is also shown in Figure 1. To do so, we check $\langle true \rangle\ M_1\ \langle A \rangle_{\geq 0.8}$ and $\langle A \rangle_{\geq 0.8}\ M_2\ \langle G \rangle_{\geq 0.98}$, both of which hold.

We can also check *quantitative* assume-guarantee triples. As shown in [5], for a PA $M$, regular safety properties $A, G$ and a fixed value of $p_A$, we can compute (again through multi-objective model checking) the tightest lower-bounded interval $I_G \subseteq [0,1]$ for which the triple $\langle A \rangle_{\geq p_A}\ M\ \langle G \rangle_{I_G}$ holds. Conversely, for a fixed value of $p_G$, we can compute the widest lower-bounded interval $I_A \subseteq [0,1]$ for which the triple $\langle A \rangle_{I_A}\ M\ \langle G \rangle_{\geq p_G}$ holds. We denote these two queries, respectively, as:

$$\langle A \rangle_{\geq p_A}\ M\ \langle G \rangle_{I_G = ?} \quad \text{and} \quad \langle A \rangle_{I_A = ?}\ M\ \langle G \rangle_{\geq p_G}$$

Intuitively, these allow us to compute the strongest possible guarantee that can be obtained for some assumption $\langle A \rangle_{\geq p_A}$ and the weakest possible assumption[1] that guarantees a particular $\langle G \rangle_{\geq p_G}$. Note that, for the latter type of query, $I_A$ can in fact be empty. This occurs when there are adversaries of $M$ that satisfy $\langle A \rangle_{\geq 1}$ but violate $\langle G \rangle_{\geq p_G}$, i.e. even under the strongest possible assumption $\langle A \rangle_{\geq 1}$ for $A$, we cannot guarantee that $\langle G \rangle_{\geq p_G}$ holds.

### D. The L* Learning Algorithm

The L* algorithm, which was proposed by Angluin [6], is one of the most influential, cited and extended online learning algorithms for regular languages. L* learns a *minimal* DFA accepting an unknown regular language $\mathcal{L}$, by interacting with a *teacher*. The teacher responds to two kinds of questions: *membership queries* (i.e., whether some word is in the target language) and *conjectures* (i.e., whether a hypothesised DFA $H$ accepts the target language). In the latter case, if the conjecture is not correct, the teacher must provide a counterexample illustrating this (i.e. a word in the symmetric difference between $\mathcal{L}$ and $\mathcal{L}(H)$). L* is attractive because it is guaranteed to produce a minimal DFA and it runs in polynomial time.

Over the past decade, L* has become popular in the context of verification. In particular, as proposed by Giannakopoulou, Pasareanu et al. (see e.g. [7]), it has been successfully applied to the automatic generation of assumptions for assume-guarantee verification of non-probabilistic systems. The key innovation was to rephrase the problem of generating an assumption as the problem of learning a (prefix-closed) regular language characterising the *weakest assumption* about a component that suffices for verification.

In this approach, the questions asked of the teacher are translated into problems that can be executed by a model checker. In the case of conjectures, counterexamples produced during model checking are used to generate the required counterexamples for L*. Actually, in practice, the weakest assumption is rarely learnt: the process either finds a simpler (stronger) assumption that suffices for verification, or discovers (by analysing the counterexamples produced) that the property being checked does not in fact hold.

## III. Probabilistic Counterexamples for Compositional Verification

*Counterexamples* are an essential ingredient of (non-probabilistic) model checking. They provide valuable feedback to the user of a model checker about the reason why a property is violated. They are also crucial to the success of verification techniques such as counterexample-guided abstraction refinement and learning-based assume-guarantee model checking. In the context of probabilistic verification, generation of counterexamples is more complex and is currently an active area of research. The basic difficulty is that, whereas a non-probabilistic property such as "an error never occurs" can be refuted with a single finite path to an error state, a probabilistic property such as "an error state is reached with probability at most $p$" is refuted by a *set* of such paths whose total probability exceeds $p$.

The fundamental techniques in this area were proposed in [14], which considers generation of counterexamples for probabilistic reachability properties (expressed with the logic PCTL) of discrete-time Markov chains (DTMCs). They show, for example, that counterexamples for PCTL properties of the form $\mathrm{P}_{\leq p}[\Diamond a]$, i.e. with non-strict, upper probability bounds, can be constructed as a finite set of finite paths that reach $a$. Furthermore, they characterise the notion of *smallest counterexample* which illustrates the violation with as few paths as possible. They also show how to handle properties with lower probability bounds (by identifying strongly connected components) and strict probability bounds (by using regular expressions).

Counterexamples for more complex models and properties can often be reduced to this basic case. For example, model checking of more expressive logics such as LTL on DTMCs reduces to computing reachability probabilities on a larger, product DTMC. Extending to probabilistic automata is also relatively straightforward (see e.g. [15]) since the first step is to find an adversary causing a violation; the problem then reduces to generating a counterexample for a DTMC.

Since probabilistic counterexamples are crucial to this work, and our needs are rather specific, we describe in the remainder of this section how these basic techniques can be adapted and extended to our setting.

[1] Not to be confused with the "weakest assumption" defined in [7].

### A. Counterexamples for Safety Properties

We focus first on the case of counterexamples for safety properties of PAs. Recall that, for PA $M$ to satisfy $\langle G \rangle_{\geq p_G}$, we require that $Pr^{\sigma}_{M}(G) \geq p_G$ for all adversaries $\sigma$. So, to refute this, we require an adversary for which this does not hold and a set of paths, under this adversary, that illustrates this. As mentioned in Section II-B, deterministic finite-memory adversaries suffice to verify (or refute) safety properties. Furthermore, we can represent the set of paths of $M$ under such an adversary $\sigma$ by a larger PA, which we denote $M^{\sigma}$. For every path $\pi$ in $M^{\sigma}$, we have $\pi{\restriction_M} \in Path^{\sigma}_{M}$, where $\pi{\restriction_M}$ denotes the projection of $\pi$ onto $M$.

**Definition 3.** For a PA $M$ and a probabilistic safety property $\langle G \rangle_{\geq p_G}$ with $M \not\models \langle G \rangle_{\geq p_G}$, a *counterexample* for $\langle G \rangle_{\geq p_G}$ is a pair $(\sigma, c)$ of a (deterministic, finite-memory) adversary $\sigma$ for $M$ with $Pr^{\sigma}_{M}(G) < p_G$ and a set $c$ of finite paths in $M^{\sigma}$ such that $Pr(c) > 1 - p_G$ and $\pi{\restriction_M} \not\models G$ for all $\pi \in c$.

The process of obtaining a counterexample $(\sigma, c)$ for $M \not\models \langle G \rangle_{\geq p_G}$ is as follows. As described in Section II-B, model checking $\langle G \rangle_{\geq p_G}$ on $M$ requires computation of $Pr^{\min}_{M}(G)$, which reduces to the problem of computing the maximum probability of reaching an accepting state in the product PA $M \otimes G^{err}$. The (deterministic, finite-memory) adversary $\sigma$ for $M$ is obtained directly from the (deterministic, memoryless) adversary of $M \otimes G^{err}$ under which this reachability probability is above $1 - p_G$. The set of paths $c$ is taken from $M^{\sigma}$ which, since $\sigma$ is deterministic, is a DTMC. Since $c$ equates to a counterexample for a non-strict, upper-bounded reachability property, as [14] shows, a finite set of finite paths suffices.

In fact, our learning techniques require not just a set of violating paths $c$, but also the *fragment* of the PA $M$ which contains these paths. This fragment, which we denote $M^{\sigma,c}$, is a (sub-stochastic) PA obtained from $M^{\sigma}$ by removing all transitions that do not appear in any path of $c$.

**Definition 4.** Let $M$ be a PA, $\sigma$ a deterministic, finite-memory adversary and $M^{\sigma} = (S, \overline{s}, \alpha_M, \delta)$ be the PA $M^{\sigma}$. If $c$ is a set of (finite or infinite) paths of $M^{\sigma}$, the corresponding *PA fragment*, denoted by $M^{\sigma,c}$, is the sub-stochastic PA $(S, \overline{s}, \alpha_M, \delta')$ where $\delta'$ is defined as follows. For distribution $\mu \in Dist(S)$, we define the sub-distribution $\mu^c$ over $S$ as $\mu^c(s') = \mu(s')$ if the state $s'$ appears in some path in $c$, and $\mu^c(s') = 0$ otherwise. For each $s \xrightarrow{a} \mu$ in $\delta$, then $\delta'$ contains $s \xrightarrow{a} \mu^c$ if and only if $\mu^c$ is non-empty.

Note that, due to the possibility of loops in $M^{\sigma}$, a fragment $M^{\sigma,c}$ may contain paths that are not in $c$. However, PA fragments have the following useful properties.

**Proposition 1.** For PAs $M$ and $M'$, PA fragment $M^{frag}$ of $M$ and probabilistic safety property $\langle G \rangle_{\geq p_G}$, we have:

(a) $M \models \langle G \rangle_{\geq p_G} \Rightarrow M^{frag} \models \langle G \rangle_{\geq p_G}$
(b) $M \| M' \models \langle G \rangle_{\geq p_G} \Rightarrow M^{frag} \| M' \models \langle G \rangle_{\geq p_G}$
(c) $M^{frag} \| M' \not\models \langle G \rangle_{\geq p_G} \Rightarrow M \| M' \not\models \langle G \rangle_{\geq p_G}$.

*Proof:* These properties can easily be shown using the notion of *strong probabilistic simulation* [11]. It is straightforward to show that $M$ strongly simulates $M^{frag}$, denoted $M^{frag} \precsim M$. Correctness of part (a) is shown by the fact that $\precsim$ preserves safety properties [11]. Strong probabilistic simulation is also *compositional* [11] (meaning that $M^{frag} \precsim M \Rightarrow M^{frag} \| M' \precsim M \| M'$), from which part (b) follows. Lastly, part (c) is a direct consequence of part (b). □

### B. Counterexamples for Assume-Guarantee Triples

We also need to consider counterexamples for probabilistic assume-guarantee triples. Recall (see Definition 2) that a triple $\langle A \rangle_{\geq p_A} M \langle G \rangle_{\geq p_G}$ is false if some adversary of $M[\alpha_A]$ satisfies assumption $\langle A \rangle_{\geq p_A}$ but violates the guarantee $\langle G \rangle_{\geq p_G}$. We first need the notion of a *witness*.

**Definition 5.** For a PA $M$ and probabilistic safety property $\langle A \rangle_{\geq p_A}$ for which $M \models \langle A \rangle_{\geq p_A}$, a *witness* is a pair $(\sigma, w)$ comprising a (deterministic, finite-memory) adversary $\sigma$ for $M$ with $Pr^{\sigma}_{M}(A) \geq p_A$ and a set $w$ of infinite paths in $M^{\sigma}$ such that $Pr(w) \geq p_A$ and $\pi{\restriction_M} \models A$ for all $\pi \in w$.

To compute a witness $(\sigma, w)$, adversary $\sigma$ is obtained exactly as for generating a counterexample $(\sigma, c)$: through probabilistic reachability on the product PA $M \otimes A^{err}$. Finding paths $w$ in $M^{\sigma}$ that show $Pr^{\sigma}_{M}(A) \geq p_A$ is then equivalent to building a counterexample in a DTMC for a strict, lower-bounded reachability property. This can be done using the techniques of [14] (analysis of strongly connected components, followed by construction of a regular expression).

**Definition 6.** For a PA $M$ and probabilistic safety properties $\langle A \rangle_{\geq p_A}$ and $\langle G \rangle_{\geq p_G}$ such that $\langle A \rangle_{\geq p_A} M \langle G \rangle_{\geq p_G}$ is false, a *counterexample* for $\langle A \rangle_{\geq p_A} M \langle G \rangle_{\geq p_G}$ is a tuple $(\sigma, w, c)$ such that $(\sigma, w)$ is a witness for $\langle A \rangle_{\geq p_A}$ in $M[\alpha_A]$ and $(\sigma, c)$ is a counterexample for $\langle G \rangle_{\geq p_G}$ in $M[\alpha_A]$.

Construction of a counterexample $(\sigma, w, c)$ for a triple $\langle A \rangle_{\geq p_A} M \langle G \rangle_{\geq p_G}$ proceeds as follows. The process of checking whether the query is true (done using multi-objective model checking [5], [13] on the product of $M[\alpha_A]$ with $A^{err}$ and $G^{err}$) also yields the adversary $\sigma$ when the query is false. The counterexample $c$ and witness $w$ can be obtained from the product in similar fashion to the cases described above. Note that, in the case where the assumption is qualitative (i.e. $p_A = 1$), the witness $w$ comprises *all* paths of $M^{\sigma}$ and so its explicit construction can be avoided.

## IV. The Learning Framework

In this section, we present a framework that learns assumptions for two-component probabilistic systems based on the asymmetric probabilistic assume-guarantee rule (ASym) of [5]. The inputs are component models $M_1, M_2$, a probabilistic safety property $\langle G \rangle_{\geq p_G}$ and an alphabet $\alpha_A$. The aim is to verify (or refute) $M_1 \| M_2 \models \langle G \rangle_{\geq p_G}$ by generating a

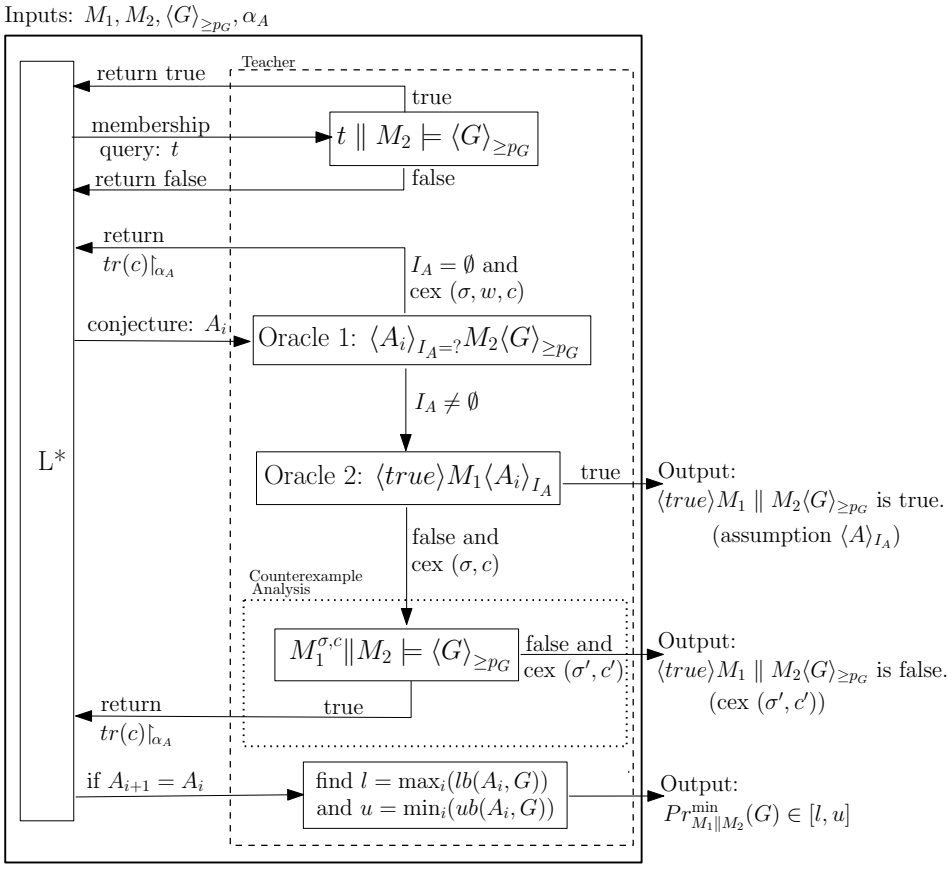


Figure 2. L*-based learning framework for the rule (ASYM)

probabilistic assumption $\langle A \rangle_{\geq p_A}$ over $\alpha_A$. Our approach is *quantitative*, in that it also yields lower and upper bounds on the minimum probability of satisfying $G$.

### A. Overview

A summary of the framework is shown in Figure 2. It is built on top of the L* algorithm, and inspired by techniques that apply this to assume-guarantee verification of labelled transition systems [7]. The introduction of probabilities brings several complications. Perhaps the most important is the issue of completeness. The approach of [7] is *complete*, meaning that if the property being verified of the system is satisfied, then there always exists an assumption that can be used to verify the property compositionally. This so-called *weakest assumption* can be formally defined, as a regular language, and used as the target language for L*. The probabilistic assume-guarantee framework of [5] is *incomplete*: even if the property is satisfied, there may be no assumption that permits a compositional verification.

Crucially, though, L*-based implementations of assume-guarantee verification do not, in practice, actually construct the weakest assumption. Instead, the aim is to, in the process of learning it, either find a simpler, stronger assumption that is sufficient to verify the property being checked or generate a counterexample that refutes it. We adopt a similar approach here: we use L* to generate a succession of conjectured assumptions and, for each one, execute probabilistic model checking queries that may either verify or refute the property $\langle G \rangle_{\geq p_G}$ on $M_1 \| M_2$. If neither is possible, information from model checking, in the form of counterexamples, guides the learning process towards a new, refined assumption.

In our context, we need to learn a *probabilistic* assumption, i.e. a probabilistic safety property $\langle A \rangle_{\geq p_A}$. However, as we will show, we can reduce this task to the problem of learning a *non-probabilistic* assumption, i.e. a regular safety property $A$. This is because, for a fixed $A$, it is possible to determine whether there is any probability threshold $p_A$ for which $\langle A \rangle_{\geq p_A}$ suffices as a probabilistic assumption. To avoid confusion, in the remainder of this section, we refer to $A$ and $\langle A \rangle_{\geq p_A}$ as the "assumption" and "probabilistic assumption", respectively.

This means that our framework can be built around the standard L* algorithm, which generates a (regular language) assumption $A$, guided by a teacher. The task of determining whether a corresponding probabilistic assumption $\langle A \rangle_{\geq p_A}$ can be created is performed by the teacher.

The overall structure of the interaction between L* and the teacher is similar to the non-probabilistic case [7]. The teacher first responds to several *membership queries*. Then, L* provides a *conjecture* for $A$ to the teacher. The teacher

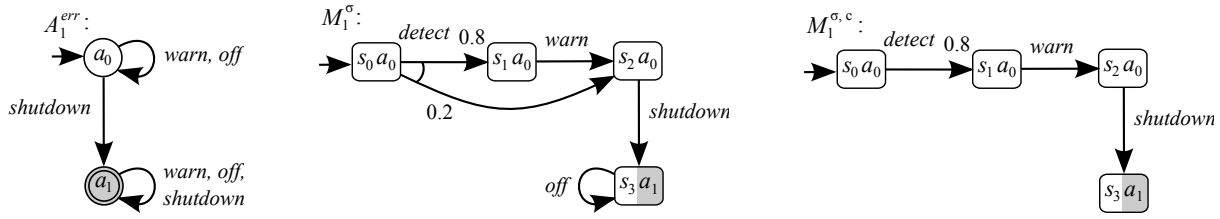


Figure 3. DFA $A_1^{err}$ for a learnt assumption $A_1$, PA $M_1^{\sigma}$ and PA fragment $M_1^{\sigma,c}$ (see Examples 3 and 4)

uses two oracles (executed by a probabilistic model checker) to analyse the conjecture and determine whether it can verify or refute $M_1 \| M_2 \models \langle G \rangle_{\geq p_G}$. These correspond to the top two outputs on the right hand side of Figure 2. If neither is possible, the teacher generates and returns traces to L* based on counterexamples generated during model checking.

Here, there is another important difference with the use of L* for non-probabilistic assume-guarantee verification. In [7], the learning is driven by the weakest assumption. When the teacher finds a conjecture to be unsuitable, it is guaranteed to be able to find a trace illustrating an inconsistency between the current assumption and the weakest assumption. In our framework, there may not exist such an assumption so this is not always possible. The feedback provided by the teacher should be seen as *heuristics* that guide L* towards an appropriate assumption.[2]

To address this limitation, we equip our framework with the ability to, at any point in its execution, produce a lower and an upper bound on the minimum probability of $G$ holding, based on the assumptions generated so far. This means that, if the algorithm reaches a point where the teacher is unable to provide feedback for L* to produce a new assumption, it can still provide valuable quantitative information to the user (this is indicated by the third possible output at the bottom of Figure 2). Furthermore, we can choose to interrupt the learning process at any point and obtain this information.

In the following sections, we describe each aspect of the learning algorithm in more detail.

### B. Answering Membership Queries

The L* learning procedure is guided by the results of membership queries as to whether a given finite trace $t$ should be included in the assumption $A$ being generated. Since, as highlighted above, an assumption may not exist, we cannot guarantee definitive answers to these queries. The criterion we use is to check whether $t \| M_2 \models \langle G \rangle_{\geq p_G}$, where $t$ here denotes a PA representing the trace, i.e a linear, $|t|+1$ state PA in which each transition has probability 1.

Note that if $t$ *does* cause a violation, i.e. $t \| M_2 \not\models \langle G \rangle_{\geq p_G}$, then $t$ should certainly not be in $A$. This is because any assumption $A$ that contains $t$ will thus not satisfy premise 2 $\langle A \rangle_{\geq p_A} M_2 \langle G \rangle_{\geq p_G}$ of (ASYM) for any value of $p_A$.

[2]Since we use L* in a non-standard fashion, we use the original version of [6], rather than optimisations e.g. due to Rivest & Schapire.

The converse does not hold, i.e. $t \| M_2 \models \langle G \rangle_{\geq p_G}$ does not imply that $t$ should be in $A$. This is because multiple traces that do not lead to a violation individually may do so when combined into a single assumption. Unfortunately, we cannot establish this with an analysis of $t$ in isolation. As we will show later in the paper, however, the proposed scheme for answering membership queries works well in practice.

**Example 3.** We execute the learning algorithm on PAs $M_1, M_2$ and property $\langle G \rangle_{\geq 0.98}$ from Example 1 and with alphabet $\alpha_A$={warn, shutdown, off}. To build its first conjecture, L* makes membership queries for the traces: $\langle warn \rangle$, $\langle off \rangle$, $\langle shutdown \rangle$, $\langle shutdown, warn \rangle$, $\langle shutdown, off \rangle$ and $\langle shutdown, shutdown \rangle$. Of these, the first two return $true$ since they do not cause a violation of $\langle G \rangle_{\geq 0.98}$ in $M_2$. All of the others return $false$ since they result in a violation (occurrence of action $fail$ with probability $0.1 > 1-0.98 = 0.02$). The resulting conjecture $A_1$ is illustrated, by its error DFA $A_1^{err}$, in Figure 3.

### C. Answering Conjectures

The second job of the teacher is to answer conjectures, i.e. to check whether a generated assumption $A$ can be used to apply rule (ASYM). For this, $A$ needs to, for some probability bound $p_A$, satisfy both premise 1, $\langle true \rangle\ M_1\ \langle A \rangle_{\geq p_A}$, and premise 2, $\langle A \rangle_{\geq p_A}\ M_2\ \langle G \rangle_{\geq p_G}$. As shown in Figure 2, the teacher checks this using two separate *oracles*, one for each premise.

**Oracle 1** checks the quantitative assume-guarantee query $\langle A \rangle_{I_A=?}\ M_2\ \langle G \rangle_{\geq p_G}$ corresponding to premise 2 of (ASYM), i.e. it determines the widest interval $I_A \subseteq [0,1]$ for which the premise holds using assumption $A$. As described in Section II-C, either $I_A$ is a non-empty, closed, lower-bounded interval, in which case $\langle A \rangle_{I_A}$ is a valid safety property, or $I_A$=$\emptyset$. In the former case, we will proceed to Oracle 2 to check premise 1 with $\langle A \rangle_{I_A}$.

In the latter case, $I_A$=$\emptyset$ indicates that, even under the assumption $\langle A \rangle_{\geq 1}$, $M_2$ would still not satisfy $\langle G \rangle_{\geq p_G}$ so $A$ must be refined, regardless of the validity of premise 1. To refine $A$ we generate a probabilistic counterexample $(\sigma, w, c)$ to show that $\langle A \rangle_{\geq 1}\ M_2\ \langle G \rangle_{\geq p_G}$ does not hold and then use this to generate traces for L*. More precisely, we take the set $\mathcal{T} = tr(c)\restriction_{\alpha_A}$ of traces for paths in $c$, restricted to the alphabet $\alpha_A$. Intuitively, we want to find a trace which is currently included in $A$ but should in fact be excluded since it causes a violation of $\langle G \rangle_{\geq p_G}$. By construction, all

paths in $c$ satisfy $A$ (since $A$ is satisfied with probability 1 under adversary $\sigma$).

Consider first the case where $\mathcal{T}$ comprises a single trace $t$. In this instance, reasoning as for membership queries above, since $t$ corresponds to a path through $M_2$ that causes $\langle G \rangle_{\geq p_G}$ to be false, any assumption $A$ that contains $t$ will not satisfy premise 2 of (ASYM). Hence, $t$ should not be in the learnt assumption $A$. Unfortunately, if $\mathcal{T}$ includes multiple traces, it is unclear whether the same is true. In this case, we return all traces in $\mathcal{T}$ to L*. However, we can increase the likelihood that $c$ contains a single trace by choosing $c$ to be the *smallest* counterexample [14] violating $G$.

**Oracle 2**, which is invoked when the interval $I_A$ from Oracle 1 is non-empty, checks premise 1 of (ASYM), i.e. it verifies whether $\langle true \rangle\, M_1\, \langle A \rangle_{I_A}$ is satisfied. If so, then we have found an assumption $\langle A \rangle_{I_A}$ that satisfies both premises of (ASYM), thus proving that $M_1 \| M_2$ satisfies $\langle G \rangle_{\geq p_G}$, and we can terminate the learning algorithm.

If, on the other hand, premise 1 is not satisfied, then we construct a counterexample $(\sigma, c)$ showing $M_1 \not\models \langle A \rangle_{I_A}$. Since, from Oracle 1, we know that $M_2$ is only guaranteed to satisfy $\langle G \rangle_{\geq p_G}$ if $\langle A \rangle_{I_A}$ is true, $(\sigma, c)$ is potentially a counterexample for $M_1 \| M_2$.

**Counterexample analysis** is then applied to determine whether $(\sigma, c)$ is a real counterexample for $M_1 \| M_2$. To do so, we construct the PA fragment $M_1^{\sigma,c}$ and check if $M_1^{\sigma,c} \| M_2 \models \langle G \rangle_{\geq p_G}$ holds. If not, we can conclude that $M_1 \| M_2 \not\models \langle G \rangle_{\geq p_G}$ (see Proposition 1(c)) and the algorithm terminates. Furthermore, a counterexample from the verification of $M_1^{\sigma,c} \| M_2$ also serves as a counterexample to illustrate the violation of $\langle G \rangle_{\geq p_G}$ by $M_1 \| M_2$.

Otherwise, $(\sigma, c)$ is not a real counterexample and we again need to refine the assumption $A$ by returning appropriate traces to L*. Here, the situation is the opposite to that of Oracle 1. Intuitively, our aim is to find a trace $t$ that is not currently in the assumption $A$ but should be. Consider as above the case where $\mathcal{T} = tr(c)\restriction_{\alpha_A}$ comprises a single trace $t$. Since this trace comes from a counterexample showing $M_1 \not\models \langle A \rangle_{\geq I_A}$, we know $t$ is not in $A$. Furthermore, from the results of the counterexample analysis, it is likely (but not guaranteed) that $t \| M_2 \models \langle G \rangle_{\geq p_G}$.

**Example 4.** We resume the execution of the algorithm described in Example 3. The first conjectured assumption $A_1$ (shown in Figure 3) is passed to Oracle 1, which checks a quantitative query $\langle A_1 \rangle_{I_A=?}\, M_2\, \langle G \rangle_{\geq 0.98}$ yielding the result $I_A = [0.8, 1]$. Intuitively, this means that, under the assumption that a *shutdown* action *never* occurs with probability 0.8 or more, $M_2$ would satisfy the property.

Since $I_A \neq \emptyset$, the teacher proceeds to Oracle 2, which checks $\langle true \rangle\, M_1\, \langle A_1 \rangle_{I_A}$, i.e. it checks whether the minimum probability of $M_1$ satisfying $A_1$ is at least 0.8. The actual minimum probability is 0 (since action *shutdown* eventually occurs with probability 1) so the property is false. We construct a counterexample $(\sigma, c)$ comprising adversary $\sigma$ and (smallest) set of paths $c$ comprising the single path: $\pi = (s_0, a_0) \xrightarrow{detect} (s_1, a_0) \xrightarrow{warn} (s_2, a_0) \xrightarrow{shutdown} (s_3, a_1)$ which has probability 0.8. The PA $M_1^\sigma$ and PA fragment $M_1^{\sigma,c}$ are both shown in Figure 3. Counterexample analysis shows that $M_1^{\sigma,c} \| M_2$ *does* satisfy $\langle G \rangle_{\geq 0.98}$ so we return the trace $\pi\restriction_{\alpha_A} = \langle warn, shutdown \rangle$ to L*.

This results in several more membership queries, after which a second conjecture $A_2$ is produced. This is identical to the assumption $A$ from Example 2 (Figure 1). Oracle 1 again gives the result $I_A = [0.8, 1]$ but, this time, Oracle 2 confirms that $\langle true \rangle\, M_1\, \langle A_2 \rangle_{\geq 0.8}$. Thus, the framework terminates concluding that $M_1 \| M_2 \models \langle G \rangle_{\geq 0.98}$.

**Example 5.** Consider now the execution of the learning algorithm with the same two components $M_1, M_2$ but on safety property $\langle G \rangle_{\geq 0.99}$. The first conjecture generated by L* is the same as $A_1$ from the previous example (Figure 3). However, Oracle 1 returns a different interval: $I_A = [0.9, 1]$. Oracle 2 finds that $\langle true \rangle\, M_1\, \langle A_1 \rangle_{\geq 0.9}$ is false (since, as before, $A_1$ has minimum probability 0). In the corresponding counterexample $(\sigma, c_2)$, adversary $\sigma$ is as for Example 4 and $c_2$ contains the single path with probability 0.2: $\pi = (s_0, a_0) \xrightarrow{detect} (s_2, a_0) \xrightarrow{shutdown} (s_3, a_1)$. Counterexample analysis shows that $M_1^{\sigma,c_2} \| M_2$ does not satisfy $\langle G \rangle_{\geq 0.99}$ since it contains the path reaching *fail* with probability $0.2 \cdot 0.1 = 0.02$ which exceeds $1 - 0.99 = 0.01$. Thus, $(\sigma, c_2)$ is a real counterexample, and the framework terminates concluding that $M_1 \| M_2 \not\models \langle G \rangle_{\geq 0.99}$.

### D. Generation of Lower and Upper Bounds

As discussed previously, there is a third possible output from our learning algorithm (shown at the bottom of Figure 2). To refine a conjecture, L* requires that any trace returned as a counterexample by the teacher has not already been tested with a membership query. In the probabilistic setting, this cannot be guaranteed, so if this occurs, we terminate the algorithm without generating further conjectures.

Fortunately, as we will now show, for any conjectured assumption $A$, it is possible to produce lower and upper bounds on the (minimum) probability of satisfying the safety property $G$. We denote these $lb(A, G)$ and $ub(A, G)$. Computation of these bounds proceeds as follows. First we compute $p_A^* = Pr_{M_1}^{\min}(A)$ and, simultaneously, generate an adversary $\sigma \in Adv_{M_1}$ that achieves this minimum probability. Next, we check the quantitative assume-guarantee query $\langle A \rangle_{\geq p_A^*}\, M\, \langle G \rangle_{I_G=?}$ and, from the resulting interval, take:

$$lb(A, G) = min(I_G)$$

For the upper bound, we compute:

$$ub(A, G) = Pr_{M_1^\sigma \| M_2}^{\min}(G)$$

using the adversary $\sigma$ from above. Then:

**Proposition 2.** $lb(A, G) \leq Pr_{M_1 \| M_2}^{\min}(G) \leq ub(A, G)$.

| Case study [parameters] | | Component sizes | | Compositional | | | Non-compositional |
|---|---|---|---|---|---|---|---|
| | | $\lvert M_2 \otimes G^{err} \rvert$ | $\lvert M_1 \rvert$ | $\lvert A \rvert$ | Time (s) | Bounds $[l, u]$ | Result |
| *client-server* | 4 | 223 | 25 | 5 | 6.9 | [0.9, 0.989999] | 0.9 |
| (1 failure) | 6 | 1,695 | 49 | 7 | 20.5 | [0.9, 0.989999] | 0.9 |
| $[N]$ | 8 | 13,367 | 81 | 9 | 1,366.3 | [0.9, 0.989999] | 0.9 |
| *client-server* | 3 | 229 | 16 | 4 | 6.6 | [0.729, 0.967238] | 0.729 |
| ($N$ failures) | 4 | 1,121 | 25 | 5 | 13.1 | [0.656099, 0.965528] | 0.6561 |
| $[N]$ | 5 | 5,397 | 36 | 6 | 87.5 | [0.59049, 0.958651] | 0.59049 |
| | 2 3 2 | 391 | 337 | 5 | 17.5 | [0.891667, 1] | 0.891667 |
| *consensus* | 2 3 20 | 391 | 3,217 | 5 | 24.2 | [0.987508, 1] | 0.987508 |
| *protocol* | 2 4 2 | 573 | 113,569 | 10 | 108.4 | [0.988263, 1] | 0.988263 |
| $[N\ R\ K]$ | 3 3 2 | 8,843 | 4,065 | 14 | 681.7 | [0.770911, 1] | 0.770911 |
| | 3 3 20 | 8,843 | 38,193 | 14 | 863.8 | [0.975040, 1] | **time-out** |
| *sensor* | 1 | 42 | 72 | 2 | 3.5 | [0.984000, 0.984000] | 0.984000 |
| *network* | 2 | 42 | 1,184 | 2 | 3.7 | [0.916800, 0.916800] | 0.916800 |
| $[N]$ | 3 | 42 | 10,662 | 2 | 4.6 | [0.903360, 0.903360] | 0.903360 |

Figure 4. Experimental results illustrating performance of the learning algorithm

*Proof:* For the lower bound, by construction, both $M_1 \models \langle A \rangle_{\geq p^*_A}$ and $\langle A \rangle_{\geq p^*_A} M \langle G \rangle_{\geq lb(A,G)}$ hold. Thus, $lb(A, G) \leq Pr^{\min}_{M_1 \| M_2}(G)$ follows from rule (ASYM). For the upper bound, since $M_1^\sigma$ is a fragment of $M_1$, Proposition 1(b) yields that: $Pr^{\min}_{M_1 \| M_2}(G) \leq Pr^{\min}_{M_1^\sigma \| M_2}(G) = ub(A, G)$. □

This means that, if the algorithm terminates because no further conjectures are possible, we can provide bounds from the current $A$. In fact, an interesting property of L* is that the series of conjectures produced is not monotonic (in terms of language inclusion). So, the lower/upper bounds from earlier assumptions may produce tighter bounds and we actually return the tightest bounds produced from any assumption.

It is worth pointing out that the steps outlined above to produce the bounds are carried out in many cases anyway. So, generating this information comes at little extra cost.

### E. Correctness and Termination

As we have seen, there are three possible outputs from the execution of the learning algorithm: (i) verification of $\langle G \rangle_{\geq p_G}$; (ii) refutation of $\langle G \rangle_{\geq p_G}$; and (iii) provision of lower/upper bounds on the minimum probability of satisfying $G$. The correctness of these three conclusions has been explained in the sections above. It is important to emphasise that this correctness is *independent* of the choices made by Oracles 1 and 2 (with regards to the traces that they return to L*) *and* of the conjectures generated by L*.

As discussed earlier in this section, due to the incompleteness of the underlying compositional verification framework, we cannot hope to guarantee that the learning algorithm will terminate and produce a definitive answer as to whether the property is satisfied or not. Instead, we provide the option to obtain lower and upper bounds at any point.

## V. Experimental Results

### A. Implementation & Case Studies

We have built a prototype tool that implements the learning framework described in this paper. The inputs to the tool are a model described in the PRISM modelling language, a specification of which PRISM modules comprise each component and a probabilistic safety property. The queries executed during the learning process are performed by either PRISM [16] or the extension of PRISM developed for multi-objective model checking in [5]. For the implementation of the L* algorithm, we use the `libalf` [17] learning library. To generate counterexamples, we build adversaries using PRISM and then apply the techniques of [14] using CARMEL which implements Eppstein's algorithm. Experiments were run on a 1.86GHz PC with 2GB RAM and we imposed a time-out of 24 hours.

We have applied our techniques to several large case studies. The first is a variant of the client-server example from [7], a commonly used benchmark for non-probabilistic assume-guarantee verification. We inject (probabilistic) failures into one or more of the $N$ clients, and then analyse the minimum probability that a mutual exclusion property holds. Next, we study Aspnes & Herlihy's randomised consensus algorithm, analysed in [5], that models $N$ distributed processes trying to reach consensus using a shared coin protocol parameterised by $K$. We are interested in the minimum probability of consensus being reached within $R$ rounds. Lastly, we verify a network of $N$ sensors, which can exhibit unsafe behaviour due to the occurrence of message failures. The case studies selected were those that were amenable to a compositional verification using the rule (ASYM). All models and properties used are available online.[3]

### B. Results & Discussion

Figure 4 shows experimental results for these case studies. The "Component sizes" columns give the sizes (number of states) of the two components, $M_1$ and $M_2$, in each model; for $M_2$, this also includes the automaton for the safety property $G$ being checked. In all cases, our implementation successfully generated an assumption to verify the required property. The table shows the size, $|A|$, of this assumption (for $M_1$) and the total time taken to learn it.

[3] `http://www.prismmodelchecker.org/files/qest10/`.

Even when it successfully verifies a property, the learning algorithm can still generate lower and upper bounds on the minimum probability of satisfying $G$ (as described in Section IV-D). These bounds are also shown in the table. For comparison, we include (where possible) the exact result from non-compositional verification using PRISM.

We also mention briefly the assumption alphabet $\alpha_A$ used in these experiments. For the last two case studies, we fixed $\alpha_A$ to be the *interface alphabet* [7], i.e. $\alpha_A = (\alpha_{M_2} \cup \alpha_G) \cap \alpha_{M_1}$. This includes all actions of $M_1$ (for which we are learning an assumption) that can either synchronise with $M_2$ or may be required to satisfy $G$. For the client-server example, we used a reduced alphabet, as proposed in [7], that improves efficiency.

The results in Figure 4 are very encouraging. We first observe that, for all case studies, $|A|$ is significantly smaller than $|M_1|$, i.e. we successfully learn an assumption that is sufficient for verification and still much more compact than the component that it represents. We also see that the *lower* probability bounds produced by our compositional technique coincide with the exact values in all cases. The *upper* bounds provide useful information in several cases; in fact, for the sensor network example, the lower and upper bounds match, yielding an exact answer.

At present, our primary focus is on the feasibility of generating assumptions and on the quality of these assumptions (e.g. their size and the accuracy of the results they provide). Hence, we do not consider a comparison of execution times between our (prototypical) tool and the (highly-optimised) PRISM. Despite this, it is worth mentioning that, in two cases, the process of compositional verification using learning is faster than non-compositional verification: for consensus (2,3,20), PRISM takes 104 seconds (more than 4 times as long) and, for consensus (3,3,20), PRISM did not finish within 24 hours.

## VI. Conclusions

We have presented a fully automated assume-guarantee framework for verification of probabilistic automata. Assumptions, which are represented as probabilistic safety properties in the style of [5], are constructed automatically using an L*-based learning algorithm. The results produced are quantitative: in the case where an assumption cannot be produced that verifies or refutes the property being verified, lower and upper bounds on the probability of satisfying the property are produced. We have demonstrated the effectiveness of our approach by using it to generate assumptions for a range of case studies.

In the future, we plan to extend the learning framework to support more assume-guarantee rules, e.g. the circular rule and the N-component rule of [5]. We also intend to optimise the efficiency of the techniques, e.g. using alphabet refinement [7] and symbolic (BDD-based) implementations of L*. Finally we plan to explore additional case studies.


## Acknowledgements

The authors are supported in part by EPSRC projects EP/D07956X and EP/F001096, EU FP7 project CONNECT and ERC Advanced Grant VERIWARE.


## References


[1] R. Segala, "Modelling and verification of randomized distributed real time systems," Ph.D. dissertation, Massachusetts Institute of Technology, 1995.

[2] N. Lynch, R. Segala, and F. Vaandrager, "Observing branching structure through probabilistic contexts," *SIAM Journal on Computing*, vol. 37, no. 4, 2007.

[3] L. Cheung, N. Lynch, R. Segala, and F. Vaandrager, "Switched PIOA: Parallel composition via distributed scheduling," *TCS*, vol. 365, no. 1-2, pp. 83–108, 2006.

[4] L. de Alfaro, T. Henzinger, and R. Jhala, "Compositional methods for probabilistic systems," in *Proc. CONCUR'01*, ser. LNCS, vol. 2154. Springer, 2001.

[5] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, "Assume-guarantee verification for probabilistic systems," in *Proc. TACAS'10*, ser. LNCS, vol. 6105, 2010, pp. 23–37.

[6] D. Angluin, "Learning regular sets from queries and counterexamples," *Information and Computation*, vol. 75, no. 2, pp. 87–106, 1987.

[7] C. Pasareanu, D. Giannakopoulou, M. Bobaru, J. Cobleigh, and H. Barringer, "Learning to divide and conquer: Applying the L* algorithm to automate assume-guarantee reasoning," *FMSD*, vol. 32, no. 3, pp. 175–205, 2008.

[8] A. Pogosyants, R. Segala, and N. Lynch, "Verification of the randomized consensus algorithm of Aspnes and Herlihy: A case study," *Distributed Computing*, vol. 13, no. 4, 2000.

[9] C. Baier, M. Groesser, M. Leucker, B. Bollig, and F. Ciesinski, "Controller synthesis for probabilistic systems," in *Proc. IFIP TCS'04*, 004, pp. 493–5062.

[10] A. Kucera and O. Strazovský, "On the controller synthesis for finite-state Markov decision processes," *Fundam. Inform.*, vol. 82, no. 1-2, pp. 141–153, 2008.

[11] R. Segala and N. Lynch, "Probabilistic simulations for probabilistic processes," *Nordic Journal of Computing*, vol. 2, no. 2, pp. 250–273, 1995.

[12] C. Baier and J.-P. Katoen, *Principles of Model Checking*. MIT Press, 2008.

[13] K. Etessami, M. Kwiatkowska, M. Vardi, and M. Yannakakis, "Multi-objective model checking of Markov decision processes," in *Proc. TACAS'07*, ser. LNCS, vol. 4424. Springer, 2007, pp. 50–65.

[14] T. Han, J.-P. Katoen, and B. Damman, "Counterexample generation in probabilistic model checking," *IEEE Transactions on Software Engineering*, vol. 35, no. 2, pp. 241–257, 2009.

[15] M. Andrés, P. D'Argenio, and P. van Rossum, "Significant diagnostic counterexamples in probabilistic model checking," in *Proc. HVC'08*, 2008, pp. 129–148.

[16] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," in *Proc. TACAS'06*, ser. LNCS, vol. 3920. Springer, 2006, pp. 441–444.

[17] B. Bollig, J.-P. Katoen, C. Kern, M. Leucker, D. Neider, and D. Piegdon, "libalf: The automata learning framework," in *Proc. CAV'10*, ser. LNCS. Springer, 2010, to appear.