Learning Probabilistic Systems from Tree Samples

Anvesh Komuravelli  
Computer Science Department  
Carnegie Mellon University  
Pittsburgh, PA, USA

Corina S. Păsăreanu  
Carnegie Mellon Silicon Valley  
NASA Ames Research Center  
Moffett Field, CA, USA

EDmund M. Clarke  
Computer Science Department  
Carnegie Mellon University  
Pittsburgh, PA, USA

Abstract—We consider the problem of learning a non-deterministic probabilistic system consistent with a given finite set of positive and negative tree samples. Consistency is defined with respect to strong simulation conformance. We propose learning algorithms that use traditional and a new stochastic state-space partitioning, the latter resulting in the minimum number of states. We then use them to solve the problem of active learning, that uses a knowledgeable teacher to generate samples as counterexamples to simulation equivalence queries. We show that the problem is undecidable in general, but that it becomes decidable under a suitable condition on the teacher which comes naturally from the way samples are generated from failed simulation checks. The latter problem is shown to be undecidable if we impose an additional condition on the learner to always conjecture a minimum state hypothesis. We therefore propose a semi-algorithm using stochastic partitions. Finally, we apply the proposed (semi-) algorithms to infer intermediate assumptions in an automated assume-guarantee verification framework for probabilistic systems.

Index Terms—probability, transition, system, simulation, conformance, active learning, tree, partition, assume-guarantee

1. Introduction

We study the problem of learning an unknown non-deterministic Labeled Probabilistic Transition System (LPTS) from tree samples. The motivation for this work was to investigate learning techniques for automating assume-guarantee style [25] compositional verification of strong simulation conformance [28] between LPTSes. Strong simulation for LPTSes is decidable in polynomial time [4] and yields stochastic tree counterexamples when it fails [19]. Stochastic trees are tree-shaped LPTSes (see Section II) with probabilities appearing on the transitions.

Compositional verification [11] is a promising approach for alleviating the state explosion problem in model checking [12]. Learning from trace [2], [23] and tree [9] counterexamples has been successfully applied before for automating the approach in a non-probabilistic setting, for checking trace inclusion [26], [10] and simulation conformance [9], respectively. The most closely related work [9] reduces simulation conformance to tree language inclusion and uses learning for deterministic tree automata to automatically generate the assumptions used in compositional reasoning. In the probabilistic setting, existing literature has dealt with learning from samples consisting of trees with information regarding the probability of acceptance [7], but learning from stochastic trees has not been considered before. Moreover, there is no existing probabilistic variant of a tree automaton to recognize stochastic tree languages. This motivated us to consider learning an LPTS directly, without working with tree languages or tree automata.

We consider first the problem of learning a non-deterministic LPTS that is consistent with respect to a set of positive and negative stochastic tree samples, where consistency is defined in terms of strong simulation conformance. For the purpose of verification, we want the learnt models to be minimal or at least to have a good upper bound on their size. We describe two algorithms, each using a different way of partitioning the state-space of the positive samples. One algorithm uses traditional state-space partitioning (Section III-A) resulting in the least number of partitions, while the other uses a new stochastic partitioning (Section III-B) resulting in the least number of states.

We then apply the above algorithms to solve the problem of learning an unknown target in Section IV. This is done in the framework of active learning with the help of a knowledgeable teacher. Typically active learning algorithms assume a teacher that answers two types of queries - membership (of a sample in the unknown target) and equivalence (between the conjectured model and the unknown target) [2]. However we observe that membership queries are not straightforward to create in our case as the learner would need to guess the transition probabilities, along with the tree-structure. Therefore, we only assume the teacher can answer equivalence queries – the teacher checks simulation equivalence (two-way simulation conformance) between a conjectured LPTS and the target LPTS and returns positive or negative stochastic trees when the check fails.

We show that active learning for LPTSes is undecidable in general. We then propose a learning algorithm that works under an assumption on the teacher which comes naturally from the way the tree counterexamples are generated from failed simulation checks. As we are interested in learning an LPTS of the least number of states, we also consider imposing a restriction on the learner to always conjecture a minimum state hypothesis. Learning with this restriction also turns out to be undecidable and we propose a semi-algorithm using stochastic partitions.
LPTSes are related to probabilistic automata (PA) [27]. Algorithms to learn PAs have only been proposed in restricted settings of stronger assumptions on a teacher [29] or approximate learning [13], [21]. Algorithms to learn a multiplicity automaton, which generalizes a PA by replacing the probabilities with arbitrary rationals, have also been proposed [5]. Adapting these to solve verification problems involving probabilistic transition systems is difficult and results in non-terminating algorithms [14]. On the other hand, we show in Section V that one can readily apply the algorithms we propose to infer intermediate assumptions in an automated assume-guarantee style framework for the verification of strong simulation conformance between LPTSes. This yields the first complete and fully automated learning framework for compositional verification of probabilistic systems. Moreover, one can extend this framework to check logical properties, such as the fragment weakly safe PCTL [8], which are preserved by the conformance and also have tree counterexamples.

Other Related Work. Learning for automating compositional reasoning of probabilistic systems has been proposed before [15] in the context of checking probabilistic reachability properties, which are refuted by sets of trace counterexamples. The approach uses a variant of L* [2], a learning algorithm for DFAs, to automatically learn deterministic systems, following previous work in the non-probabilistic setting [26]. The approach uses a sound but incomplete rule, and therefore, it is not guaranteed to terminate (completeness is necessary for termination). A complete rule for such properties restricted to systems without non-determinism has been considered recently [4]. It uses learning with probabilistic trace inclusion as the conformance relation which is undecidable. Also, the learning algorithm is not guaranteed to terminate. In contrast, we use simulation conformance which is decidable in polynomial time and leads to a sound and complete rule (Section V). We are also able to guarantee termination for the algorithm proposed in Section V when using classical partitions to infer a consistent LPT.

Our work draws inspiration from a previous work [18] that automates assumption generation by using an algorithm for learning the minimal separating automaton from positive and negative trace counterexamples. The counterexamples are provided via model checking in an assume-guarantee framework. Similar to our work, they use a partitioning approach, where the goal is to find a folding of the counterexamples into the learnt model. A different approach has been proposed to find the separating automaton based on L* which makes use of membership queries, in addition to equivalence queries [10]. All these works were done in the context of non-probabilistic reasoning under trace semantics and thus, are different from our setting.

Learning a minimum-state automaton from positive and negative samples is a well studied problem [3], [24], [16] that is known to be hard [17]. Algorithms have also been proposed for samples with stochastic information, i.e. the probability of acceptance of a trace or a tree [6], [7], learning stochastic finite (tree) automata. As also previously said, we cannot immediately borrow existing results from the above automata-theoretic approaches.

II. PRELIMINARIES

Labeled Probabilistic Transition Systems. Let S be a non-empty set. Dist(S) is defined to be the set of discrete probability distributions over S. We assume that all the probabilities specified explicitly in a distribution are rationals in [0,1]; there is no unique representation for all real numbers on a computer and floating-point numbers are essentially rationals. For s ∈ S, δs is the Dirac distribution on s, i.e. δs(s) = 1 and δs(t) = 0 for all t ≠ s. For μ ∈ Dist(S), the support of μ, denoted Supp(μ), is defined to be the set {s ∈ S | μ(s) > 0} and for X ⊆ S, μ(X) stands for ∑s∈X μ(s). The models we consider, defined below, have both probabilistic and non-deterministic behavior. Thus, there can be a non-deterministic choice between two probability distributions, even for the same action. Such modeling is typically used for underspecification. Moreover, the theory described does not become any simpler by disallowing non-deterministic choice for a given action (see the discussion on counterexamples at the end of this section).

Definition 1 (LPTS). A Labeled Probabilistic Transition System (LPTS) is a tuple ⟨S, s0, α, τ⟩ where S is a set of states, s0 ∈ S is a distinguished start state, α is a set of actions and τ ⊆ S × α × Dist(S) is a probabilistic transition relation. For s ∈ S, s ∈ α and μ ∈ Dist(S), we denote (s, s, μ) ∈ τ by s → s μ and say that (s, s, μ) is a transition on τ.

An LPTS is called reactive if τ is a partial function from S × α to Dist(S) (i.e. at most one transition on a given action from a given state).

Throughout this paper, we use filled circles to denote start states in the pictorial representations of LPTSes. For example, Figure 1 shows three LPTSes. For μ = {((s1, ½), (s2, ½))}, L1 has the transition s1 → μ. All the LPTSes in the figure are reactive as no state has more than one transition on a given action. In the literature, an LPTS is also called a simple probabilistic automaton [28]. Similarly, a reactive LPTS is also called a (Labeled) Markov Decision Process. Also, note that an LPTS with all the distributions restricted to Dirac distributions is the classical (non-probabilistic) Labeled Transition System (LTS); thus a reactive LTS corresponds to the standard notion of a deterministic LTS. We only consider finite state, finite alphabet and finitely branching (i.e. finitely many transitions from any state) LPTSes. We use ⟨S0, s0, α0, τ0⟩ for an LPTS L0 and ⟨S1, s1, α1, τ1⟩ for an LPTS L1.

We are also interested in LPTSes with a tree structure, i.e. the start state is not in the support of any distribution and every
other state is in the support of exactly one distribution. We call such LPTSes stochastic trees or simply trees. For example, \( C_p, p \in (0, 1) \), in Figure 1 is a tree.

**Strong Simulation.** In the non-probabilistic case, for two labeled transition systems (LPTSes), a pair of states belonging to a strong simulation relation depends on whether certain other pairs of successor states also belong to the relation. For LPTSes, one has successor distributions instead of successor states; a pair of states belonging to a strong simulation relation should now depend on whether certain other pairs in the supports of these successor distributions also belong to \( R \). We thus need a binary relation between distributions, Strong Simulation. A strong simulation relation depends on whether certain other LPTSes, one has successor distributions instead of successor states; a pair of states belonging to a strong simulation relation should now depend on whether certain other pairs in the supports of these successor distributions also belong to \( R \). We thus need a binary relation between distributions, Strong Simulation. A strong simulation relation depends on whether certain other pairs in the supports of these successor distributions also belong to \( R \).

Consider an example with \( sRt \) and the transitions \( s \xrightarrow{\alpha} \mu_1 \) and \( t \xrightarrow{\alpha} \mu_2 \) with \( \mu_1 \) and \( \mu_2 \) as in Figure 2. In this case, one easy way to match the probabilities is to pair \( s_1 \) with \( t_1 \) and \( s_2 \) with \( t_2 \). This is sufficient if \( s_1Rt_1 \) and \( s_2Rt_2 \) also hold, in which case, we say that \( \mu_1 \sqsubseteq_R \mu_2 \). However, such a direct matching may not be possible in general. As shown in Figure 3, we need a more general notion of matching the probabilities. One can achieve that by splitting the probabilities under the distributions in such a way that one can then directly match the probabilities as in Figure 2. Now, if \( s_1Rt_1, s_1Rt_2, s_2Rt_2 \) and \( s_2Rt_3 \) also hold, we say that \( \mu_1 \sqsubseteq_R \mu_2 \). Note that there can be more than one possible splitting.

This is the central idea behind the following definition where the splitting is achieved by a weight function. For the rest of the section, let \( L_1 \) and \( L_2 \) be two LPTSes, \( \mu_1 \in \mathrm{Dist}(S_1) \), \( \mu_2 \in \mathrm{Dist}(S_2) \) and \( R \subseteq S_1 \times S_2 \).

**Definition 2.** Let \( \mu_1 \sqsubseteq_R \mu_2 \) iff there is a weight function \( w : S_1 \times S_2 \to \mathbb{Q} \cap [0, 1] \) such that

1. \( \mu_1(s_1) = \sum_{s_2 \in S_2} w(s_1, s_2) \) for all \( s_1 \in S_1 \),
2. \( \mu_2(s_2) = \sum_{s_1 \in S_1} w(s_1, s_2) \) for all \( s_2 \in S_2 \),
3. \( w(s_1, s_2) > 0 \) implies \( s_1Rt \) for all \( s_1 \in S_1, s_2 \in S_2 \).

\( \mu_1 \sqsubseteq_R \mu_2 \) can be checked by computing the maxflow in an appropriate network and checking if it equals 1.0. If \( \mu_1 \sqsubseteq_R \mu_2 \) holds, \( w \) in the above definition is one such maxflow function. As explained above, \( \mu_1 \sqsubseteq_R \mu_2 \) can be understood as matching all the probabilities (after splitting appropriately) under \( \mu_1 \) and \( \mu_2 \). Considering \( \mathrm{Supp}(\mu_1) \) and \( \mathrm{Supp}(\mu_2) \) as two partite sets, this is the weighted analog of saturating a partite set in bipartite matching, giving us the following analog of the well-known Hall’s Theorem for saturating \( \mathrm{Supp}(\mu_1) \).

**Lemma 1.** \( \mu_1 \sqsubseteq_R \mu_2 \) iff for every \( S \subseteq \mathrm{Supp}(\mu_1), \mu_1(S) \leq \mu_2(R(S)) \).

It follows that when \( \mu_1 \not\sqsubseteq_R \mu_2 \), there exists a witness \( S \subseteq \mathrm{Supp}(\mu_1) \) such that \( \mu_1(S) > \mu_2(R(S)) \). For example, if \( R(s_2) = \emptyset \) in Figure 2 its probability \( \frac{1}{2} \) under \( \mu_1 \) cannot be matched and \( S = \{ s_2 \} \) is a witness subset.

**Definition 3.** \( R \) is a strong simulation iff for every \( s_1Rt_2 \) and \( s_1 \xrightarrow{\alpha} \mu_1' \) there is a \( \mu_2' \) with \( s_2 \xrightarrow{\alpha} \mu_2' \) and \( \mu_1' \sqsubseteq_R \mu_2' \).

For \( s_1 \in S_1 \) and \( s_2 \in S_2 \), \( s_2 \) strongly simulates \( s_1 \), denoted \( s_1 \preceq s_2 \), iff there is a strong simulation \( T \) such that \( s_1Ts_2 \) strongly simulates \( L_1 \), also denoted \( L_1 \preceq L_2 \), iff \( s_1 \preceq s_2 \). For the latter, alternatively, we say that simulation conformance holds between \( L_1 \) and \( L_2 \).

**Definition 4.** \( \preceq \) is a preorder (i.e. reflexive and transitive).

Finally, we find the following characterization of \( \preceq \) useful
Lemma 4. Let \( L_1 \) be a tree and \( s_1 R s_2 \) iff for every \( s_1 \rightarrow a \mu_1 \), there exists \( s_2 \rightarrow a \mu_2 \) with \( \mu_1 \leq a \mu_2 \). Then, \( R \leq \preceq \).

Proof Sketch: \( R \leq \preceq \) by Def. \[ \preceq \leq R \] can be proved by induction on the height of a state of \( L_1 \) using Lemma \[ \preceq \].

Note that the condition on \( R \) in the lemma is stronger than the one to make it a strong simulation (Definition \[ \preceq \]). Also, if \( L_1 \) is not a tree, we can only conclude that \( R \leq \preceq \), in general. See Figure 4 for an example where \( R \leq \preceq \).

Counterexamples to \( \preceq \). In the active learning problem we are interested in (Section \[ \preceq \]), a learner uses counterexamples to simulation conformance as diagnostic information. We will now briefly discuss what these counterexamples are. Let \( L_1 \) and \( L_2 \) be two LPTSes.

Definition 5 (Language of an LPTS). Given an LPTS \( L \), we define its language, denoted \( \mathcal{L}(L) \), as the set \( \{ L' | L' \text{ is an LPTS and } L' \preceq L \} \).

Lemma 5. \( L_1 \preceq \preceq L_2 \) iff \( \mathcal{L}(L_1) \subseteq \mathcal{L}(L_2) \).

Proof: Necessity follows trivially from the transitivity of \( \preceq \) and sufficiency follows from the reflexivity of \( \preceq \) which implies \( L_1 \in \mathcal{L}(L_1) \).

Thus, a counterexample \( C \) can be defined as follows.

Definition 6 (Counterexample). A counterexample to \( L_1 \preceq \preceq L_2 \) is an LPTS \( C \) such that \( C \in \mathcal{L}(L_1) \setminus \mathcal{L}(L_2) \), i.e. \( C \preceq L_1 \) but \( C \preceq L_2 \).

Now, \( L_1 \) itself is a trivial choice for \( C \) but it does not give any more useful information than what we had before checking the conformance. Moreover, it is preferable to have \( C \) with a special and simpler structure to efficiently work with counterexamples. Fortunately, we have a simpler characterization using trees.

Theorem 1 \[ \preceq \]. If \( L_1 \preceq \preceq L_2 \), there is a tree which serves as a counterexample.

Proof Sketch: One can instrument the algorithm to compute the coarsest strong simulation described earlier to obtain a tree counterexample whenever a pair of states is removed from the current relation, making use of Lemma \[ \preceq \].

For example, \( C_p \) in Figure 1 for \( p \in \left[ 0, \frac{1}{2} \right) \), is a counterexample to \( L_1 \preceq \preceq L_2 \). In another work, we showed that structures simpler than trees are not sufficient as counterexamples, even when one of the models is reactive \[ \preceq \].

We note an important feature of the algorithm used to prove the above theorem \[ \preceq \]. A counterexample \( C \) generated by the algorithm is essentially a finite tree execution of \( L_1 \). That is, there is a total mapping \( M : S_C \rightarrow S_1 \) such that for every transition \( c \xrightarrow{a} \mu_c \) of \( C \), there exists \( M(c) \xrightarrow{a} \mu_1 \) such that \( M \) restricted to \( \text{Supp}(\mu_c) \) is an injection and for every \( c' \in \text{Supp}(\mu_c) \), \( \mu_c(c') = \mu_1(M(c')) \). Note that \( M \) is also a strong simulation. We call such a mapping an execution mapping from \( C \) to \( L_1 \) in the rest of the paper. An execution mapping is shown in brackets beside the states of \( C_p \) for \( p \in \left[ 0, \frac{1}{2} \right) \) in Figure 1. While our algorithm always generates counterexamples with an execution mapping, it is possible to have a tree counterexample, as per Definition \[ \preceq \] without such a mapping. For example, \( C_p \) in Figure 1 for \( p \in \left( 0, \frac{1}{2} \right) \) is also a counterexample with no such execution mapping.

The condition we impose on a teacher in the active learning problem (Section \[ \preceq \]) is regarding this execution mapping.

III. LEARNING A CONSISTENT LPTS

We are interested in the problem where we are given a finite set of positive stochastic trees (i.e. in the language of an LPTS), say \( \mathcal{P} \), and another finite set of negative stochastic trees (i.e. not in the language of an LPTS), say \( \mathcal{N} \). These trees constitute the samples for a learner. The goal is to learn an LPTS \( L \) such that \( \mathcal{P} \preceq \mathcal{L}(L) \) and \( \mathcal{N} \cap \mathcal{L}(L) = \emptyset \), i.e. \( \mathcal{P} \preceq L \) for every \( p \in \mathcal{P} \) and \( \mathcal{N} \preceq L \) for no \( n \in \mathcal{N} \). Such an \( L \) is said to be consistent with the tree samples. Without loss of generality, assume that \( \mathcal{P} \neq \emptyset \) as otherwise, a single state LPTS with no transitions is trivially consistent. Also, note that the LPTS obtained by merging the start states of all trees in \( \mathcal{P} \), say \( L_P \), trivially satisfies \( \mathcal{P} \preceq L_P \) for every \( p \in \mathcal{P} \). Now, if \( L \) is a consistent LPTS, it can be shown that \( L_P \preceq L \) and hence, by Lemma \[ \preceq \], \( L_P \) is also consistent. Thus, one can easily check, in polynomial time, if there exists a consistent LPTS by checking \( \mathcal{N} \preceq L_P \) for every \( n \in \mathcal{N} \). For this reason, we always assume the existence of a consistent LPTS. Clearly, the size of \( L_P \) is as large as that of \( \mathcal{P} \).

If possible, we would like to learn a model with the least size, or at least have a good upper bound on its size. Such models would be useful when automating assume-guarantee reasoning (see Section \[ \preceq \]). The algorithms we propose draw inspiration from the ones used to infer consistent non-probabilistic automata from counterexample traces \[ \preceq \], \[ \preceq \], \[ \preceq \] which are based on partitioning the state space of the counterexamples. Let \( S_P = \bigcup_{p \in \mathcal{P}} S_p \) and \( S_N = \bigcup_{n \in \mathcal{N}} S_n \). First, we consider an algorithm based on the traditional state space partitioning of \( S_P \). While there is an upper bound on the size of the learnt model, we show that such partitioning is insufficient to obtain a minimum state consistent probabilistic system (LPTS). However, as we will see in Section \[ \preceq \], we find it useful in learning an unknown target LPTS. We will then introduce a new way of partitioning the state space, which we call stochastic partitioning, enabling us to obtain a minimum state consistent LPTS.
A. Using State Partitions

The first algorithm uses traditional partitions of $S_P$. For a partition $\Pi$ of $S_P$, let $E_{\Pi}$ denote the set of equivalence classes under $\Pi$ and for a state $s \in S_P$, we let $[s]_{\Pi}$ denote the equivalence class of $s$ (we drop the subscript $\Pi$ when it is clear from the context). We always assume that $[s^0_P]_{\Pi} = [s_Q]_{\Pi}$ for every $P, Q \in \mathcal{P}$, i.e., the start states of all the positive counterexamples are mapped to the same equivalence class.

**Definition 7 (Quotient LPTS).** Given a partition $\Pi$ of $S_P$, define the quotient LPTS, denoted $\mathcal{P}/\Pi$, as the LPTS $(\mathcal{E}_{\Pi}, e^0, a, \alpha)$ where $e^0 = [s^0_P]_{\Pi}$ for every $P \in \mathcal{P}$, $\alpha = \bigcup_{P \in \mathcal{P}} \alpha_P$ and $(e, a, \mu) \in \tau$ iff there exists $(s, a, \mu_P) \in \tau_P$ for some $P \in \mathcal{P}$ with $[s]_{\Pi} = e$ such that $\mu = \text{lift}(\mu_P)$ where $\text{lift}(\mu_P)(e') = \sum_{e' \in e'} \mu_P(e')$ for all $e' \in E_{\Pi}$.

It can be easily shown that a quotient is always a well-defined LPTS. In the following, $\Pi$ is a partition of $S_P$.

**Lemma 6.** $\mathcal{P}/\Pi$ is consistent with $\mathcal{P}$ for all $\Pi$.

**Proof Sketch:** One can show that $\{(s, [s]_{\Pi}) | s \in S_P\}$ is a strong simulation between $\mathcal{P}$ and $\mathcal{P}/\Pi$ for every $P \in \mathcal{P}$.

**Definition 8 (Consistent Partition).** $\Pi$ is defined to be consistent iff $\mathcal{P}/\Pi$ is consistent with $\mathcal{N}$, i.e., for every $N \in \mathcal{N}$, $N \not\sim \mathcal{P}/\Pi$.

Thus, we are able to find a consistent partition. As we show below, we can always construct a consistent partition with a bounded size, where the size of $\Pi$ is $|E_{\Pi}|$.

**Lemma 7.** If $L$ is an LPTS of $k$ states consistent with $\mathcal{P}$, then there is a $\Pi$ of size at most $2^k$ such that $\mathcal{P}/\Pi \leq L$.

**Proof Sketch:** Let $P \in \mathcal{P}$. As $P \preceq L$, there is a strong simulation $R_P \subseteq S_P \times S_L$ with $s^0_P R_P s_Q$. As $P$ is a tree, $s^0_P$ is not in the support of any distribution and hence, assume without loss of generality that $R_P(s^0_P) = \{s^0_Q\}$. Let $R = \bigcup_{P \in \mathcal{P}} R_P$. Now, $R$ induces a partition $\Pi$ of $S_P$ such that for $s_1, s_2 \in S_P$, $[s_1]_{\Pi} = [s_2]_{\Pi}$ iff $R(s_1) = R(s_2)$. Note that $[s^0]_{\Pi} = [s^0_Q]_{\Pi}$ for $P, Q \in \mathcal{P}$. The size of $\Pi$ is clearly bounded by $2^k$. Now, we can show that $\{(s_P)_{\Pi}, s_1\} R(s_1)$ is a strong simulation between $\mathcal{P}/\Pi$ and $L$.

Note that, if $L$ and every $P \in \mathcal{P}$ is an LTS, an upper bound of $k$ on the size can be shown by choosing $R_P$ in the proof to be a function. The following is now immediate, using Lemmas 3 and 6.

**Corollary 1.** For every consistent LPTS of $k$ states, there is a consistent partition of size at most $2^k$.

**Observation.** This shows that if $L$ is a minimum state consistent LPTS, there exists a consistent partition of $S_P$ of size at most exponential in $|S_L|$. While there may be a better bound, this way of partitioning $S_P$ cannot guarantee a minimum state consistent LPTS in general. For example, $H_1$ in Figure 6 is the quotient for a least sized consistent partition of $P$ for the trees in Figure 5 (obtained by merging $s_3$ and $s_4$). On the other hand, $H_\lambda$, where $\lambda$ is any value in $(0, 1)$, is another consistent LPTS with one less state.

**Algorithm.** A na"ive algorithm for finding a least-sized consistent partition is to enumerate all the partitions of $S_P$, with increasing size, and for each of them, check if the corresponding quotient simulates any tree in $\mathcal{N}$. Alternatively, we can cast it as an instance of the satisfiability problem over linear rational arithmetic, as shown below. In general, this is more efficient than the exhaustive search in the naïve algorithm, and also prepares the ground for an algorithm we discuss in the next subsection.

First, we describe the encoding to check if there is a consistent partition of size at most a given $k$. Let $e_i$ denote the equivalence class $i$ for $1 \leq i \leq k$. For each $i$ and state $s \in S_P$, we introduce a new boolean variable, say $v_{[s]=i}$, to denote $[s] = e_i$. We add the constraint $\land_{i=1}^k v_{[s]=i}$ for every $s \in S_P$ for the partition to be well-defined. Moreover, we fix $e_1$ to be the start state of the resulting quotient and have a constraint that $v_{[s_0]=1}$ for every $P \in \mathcal{P}$ as $e_1$ should now contain all the start states (Definition 7).

Now, to encode consistency, we want to say that no tree $N \in \mathcal{N}$ is simulated by the resulting quotient. We can avoid introducing a universal quantification over all possible strong simulations by finding a way to say that $(s^0_N, e_1)$ is not in the smallest strong simulation, for every $N \in \mathcal{N}$. Fortunately, we can make use of Lemma 4 to achieve exactly this. We introduce a boolean variable $R_{x,i}$ to denote that $s \in S_N$ is related to $e_i$ by the strongest simulation. Let $t_n = (s_n, a, \mu_n)$ and $t_p = (s_p, a, \mu_p)$ be a transition of $\mathcal{N}$ and $\mathcal{P}$, respectively, on the same action $a$, and $1 \leq i \leq k$. Consider the expression $d_{\mu_p, \mu} \land v_{[s]=i}$, denoted $\sigma_{t_n, i, t_p}$. If $d_{\mu_p, \mu}$ denotes $\mu_p \leq_R \text{lift}(\mu_p)$, then this expression has the meaning that $[s_p] = e_i$ and the transition corresponding to $t_p$ in the quotient, viz. $e_i, s_p \xrightarrow{a} \text{lift}(\mu_p)$, simulates $t_n$. If $X(s)$ denotes the set of all transitions outgoing from $s \in S_N$, $Y(a)$ denotes the set of all transitions in $\mathcal{P}$ on action $a$ and act(t) denotes the label of transition $t$.
denotes the action for the transition $t$, we add
\[ R_{t,n,i} \iff \bigwedge_{t_n\in X(s)} \bigvee_{t_p \in Y(\sigma(t_n))} \sigma_{t_n,i,t_p} \]
according to Lemma 4
\[ \text{lift}(\mu_p)(e_i) \] can be encoded as \[ \sum_{s\in \text{Supp}(\mu_p)} l_{\mu_p,i,s} \] where
\[ l_{\mu_p,i,s} \] denotes the contribution of $s$ to the lifted probability of $e_i$ under $\mu_p$ and satisfies
\[ (u[s]=i) \implies l_{\mu_p,i,s} = \mu_p(s) \land (\neg u[s]=i) \implies l_{\mu_p,i,s} = 0). \]
$d_{\mu_p,\mu_p}$ is encoded as follows. If we use Definition 2 alone, we need to introduce a nested existential quantifier for the weight function (to say that $d_{\mu_p,\mu_p}$ iff there is a weight function satisfying the conditions). To avoid this nested quantification, we also make use of Lemma 1. First, we introduce a variable for the weight function and encode the constraints of Definition 2 if $\subseteq_R$ holds between the distributions. We also introduce a variable for the witness subset $S \subseteq \text{Supp}(\mu_p)$ and encode the condition of Lemma 1 when $\subseteq_R$ fails to hold. This variable for the witness subset can, in turn, be encoded using individual boolean variables for each $s \in \text{Supp}(\mu_p)$. We also need boolean variables for the image of this witness subset under $R$. The details are straightforward and left to the reader. Finally, we encode consistency by having the constraint $\neg R_{\mu_p,i}$ for every $N \in N$.

It is not hard to show that the encoding is correct, i.e. the resulting encoding is satisfiable iff there is a consistent partition of size at most $k$. One can then obtain an algorithm to find a least-sized consistent partition by starting with $k = 0$ and incrementing it as long as the encoding for $k$ is unsatisfiable. As satisfiability over linear rational arithmetic is decidable, this is guaranteed to terminate from Corollary 1.

**Theorem 2.** The above described algorithm to find a least-sized consistent partition of $S_P$ terminates.

**B. Using Stochastic Partitions**

As noted above, the quotient of a least-sized consistent partition need not have the least number of states. We observe that the main reason for this is not being able to partition $S_P$ such that there is a one-to-one correspondence between the equivalence classes and $S_L$, instead of the current $2^S_L$ for a consistent LPTS $L$ (proof of Lemma 7). This suggests that we can learn a minimum state consistent LPTS if we can find a way to group the states of $S_P$ (groups need not be disjoint) with such a correspondence. This will then imply that if there is a minimum state consistent LPTS $L$, we can use this grouping to obtain an equally sized consistent LPTS. One can then automate the search for such a grouping using constraint solving.

Let $L$ be a consistent LPTS and let us see what we can do to group $S_P$ to have the above one-to-one correspondence with $S_L$. Consider Figure 3 again and let $\mu_1$ be outgoing from the root of some tree $P$ in $P$ and $\mu_2$ appear in $L$. Let there be three groups (initially empty), one per state in $\text{Supp}(\mu_2)$, say $G_{t_1}$, $G_{t_2}$, and $G_{t_3}$. As explained in Section 11, having $\mu_1 \subseteq_R \mu_2$, for some $R$, can be thought of as finding a way of splitting the probabilities in both the distributions and pairing states, already in $R$, to directly match the probabilities. We would like to use this matching to group the states of $S_P$. In particular, looking at the figure, we would like to place the two splits of $s_1$ ($s_2$ in $G_{t_1}$ and $G_{t_2}$ ($G_{t_2}$ and $G_{t_3}$), respectively.

As the probability of each split of a state in $\text{Supp}(\mu_1)$ is matched with that of some split of exactly one state in $\text{Supp}(\mu_2)$, one can also think of the above grouping in the following alternative way. As the probability of $1/3$ for $s_1$ is split into $1/2$ and $1/6$, such a grouping can be seen as being put in $G_{t_1}$ with probability $1/3$ and in $G_{t_2}$ with probability $1/6$. Thus, instead of putting $s_1$ deterministically into one group, it is put stochastically into multiple groups. Let these splits of $s_1$ put in $G_{t_1}$ and $G_{t_2}$ be $s_1[t_1]$ and $s_1[t_2]$, respectively.

Now, consider $s_3[t_3]$. As the corresponding probability of $1/3$ is matched with that of some split of $t_3$ (implying $s_1[t_1]$), and as $s_1$ is not in the support of any distribution other than $\mu_1$ (note that $P$ is a tree), we need not consider if $s_1$ is related, by $R$, to any other state in $L$, as far as $s_1[t_1]$ is concerned. And therefore, any distribution outgoing from this split of $s_1$ will only need to be related to some distribution outgoing from $t_1$ (by $\subseteq_R$). Similarly, for $s_1[t_2]$ and $t_2$. Now, if $\mu_3$ is a distribution outgoing from $s_1$ in $P$, we may want to relate it to a distribution $\mu$ outgoing from $t_1$ (for $s_1[t_1]$) and another distribution $\mu'$ outgoing from $t_2$ (for $s_1[t_2]$), respectively. For a state $s_3 \in \text{Supp}(\mu_3)$, considering $\mu_3 \subseteq_R \mu$ and $\mu_3 \subseteq_R \mu'$ both hold, following the above described stochastic grouping may result in two different ways of grouping $s_3$. Thus, we need to remember the group of its parent, denoted by $\text{par}(\cdot)$, when grouping a state in $S_P$.

This is the main motivation behind a stochastic partition, which is defined below.

**Definition 9 (Stochastic Partition).** A stochastic partition of $S_P$ is a tuple $(G, \{[s]\}_{s \in S_P})$ where $G \subseteq 2^S_P$ and $[s] : G \rightarrow \text{Dist}(G)$ for every $s \in S_P$, such that $\bigcup G = S_P$ and
1) there is a $g^0 \in G$ such that for every $P \in \mathcal{P}$ and $g \in G$,
   \[ [s^0_P](g) = \delta_{g^0} \] and
2) for every non-root state $s \in S_P$ and $g \in G$, $[s](g)$ is defined if $\text{par}(s)(g')(g) > 0$ for some $g' \in G$.

Furthermore, $s \in g$ iff $[s](g')(g) > 0$ for some $g' \in G$, for every $s \in S_P$ and $g \in G$.

We use $(G_{\Pi}, \{[s]_{\Pi}\})$ for a stochastic partition $\Pi$ and when $\Pi$ is clear, we drop the subscripts.

Here, $G$ denotes the groups mentioned above and $[s]$ denotes the stochastic grouping of $s \in S_P$ given a group of its parent. Point 1 above says that the start states of all trees in $\mathcal{P}$ go deterministically to a designated group. Note that the start states have no parents and the dependence of $[s^0_P]$ on an argument is just a notational convenience. And point 2 says that for every non-root state $s$; $[s]$ is only defined for a valid group of its parent. We implicitly assume that $[s](g')(g) = 0$ for every $g \in G$ if $[s]$ is not defined at $g'$. Now, we define the quotient of a stochastic partition in the following way.
Definition 10 (Quotient LPTS). Given a stochastic partition \( \Pi = (G, \{[s]\}_s) \) of \( S_P \), define the quotient LPTS, denoted \( \overline{\Pi} \), as the LPTS \( (G, g^0, \alpha, \tau) \) where \( g^0 \in G \) is such that \([s'_P]_P(g) = \delta_{g^0} \) for every \( P \in \overline{P} \) and \( g \in G \), \( \alpha = \bigcup_{P \in \overline{P}} \alpha_P \) and \( (g, \alpha, \mu) \in \tau \) iff there exists \((s, \alpha, \mu_p) \in \tau_P \), for some \( P \in \overline{P} \) such that \( s \in g \) and for every \( g' \in G \),

\[
\mu(g') = \sum_{s' \in g'} [s']_{[s]}(g')(g') \cdot \mu_P(s').
\]

We denote this relation between \( \mu \) and \( \mu_p \) by \( \mu = \lift(\mu_p, g) \).

Thus, \((g, \alpha, \mu) \in \tau \) iff there is a state \( s \in g \) with \( s \to \mu_p \) and \( \mu \) is obtained by lifting \( \mu_p \), given that \( s \in g \). For this to make sense, we need to show that the lifting is a valid distribution. In the following, \( \Pi = (G, \{[s]\}_s) \) is a stochastic partition.

Lemma 8. \( \overline{\Pi} \) is a well-defined LPTS.

We have the following lemma analogous to classical partitions.

Lemma 9. \( \overline{\Pi} \) is consistent with \( \overline{P} \) for all \( \overline{P} \).

Proof Sketch: One can show that \( \{([s], g) \in G, s \in S_P \cap g \} \) is a strong simulation between \( P \) and \( \overline{\Pi} \) for \( P \in \overline{P} \).

Consistency of a stochastic partition is defined in the same way as Definition 8. Thus, we reduce the problem of finding a minimum state consistent LPTS to that of finding a least-sized consistent stochastic partition where the size of a stochastic partition is its number of groups.

Lemma 10. If \( L \) is an LPTS of \( k \) states consistent with \( \overline{\Pi} \), then there is a \( \Pi \) of size at most \( k \) with \( \overline{\Pi} \) \( \leq \) \( L \).

Proof Sketch: Let \( P \in \overline{P} \). As \( P \leq L \), there is a strong simulation \( R_P \subseteq S_P \times S_L \) with \( s^0_P, R_P s^0_P \). Let \( R = \bigcup_{P \in \overline{P}} R_P \). Now, construct a stochastic partition with at most \( |S_L| \) many groups following the intuitive explanation we gave when motivating stochastic partitions. For distributions \( \mu_p \in \Dist(S_P) \) and \( \mu \in \Dist(S_L) \), the stochastic groupings of a state \( s \in \Supp(\mu_p) \) is obtained by using a weight function showing \( \mu_p \subseteq_R \mu \). In particular, \( s \) is put in the group corresponding to \( s_l \in S_L \) with probability \( w(s, s_l) / \mu_p(s) \) where \( w \) is the weight function which is uniquely chosen given \( \mu_p \) and \( \mu_l \). Moreover, \( \mu_l \) and this grouping depend on the group of \( \par(s) \). Once such a stochastic partition \( \Pi \) is built, we can show that \( \{(g, s) \mid g \in \Par \} \) is a strong simulation between \( \overline{\Pi} \) and \( L \).

Our main result follows as an immediate corollary, using Lemmas 8 and 9.

Corollary 2. For every consistent LPTS of \( k \) states, there is a consistent stochastic partition of size at most \( k \).

So, we can obtain a minimum state consistent LPTS by constructing the quotient for a consistent stochastic partition of \( S_P \) of the least size. For example, \( H_\alpha, \lambda \in (0, 1) \), in Figure 5 is the quotient for a least sized consistent stochastic partition for the trees in Figure 5 (where \( s_1 \) goes to group 1, \( s_2 \) goes to group 2 with probability \( \lambda \) and to group 1 with \( 1 - \lambda \) and \( s_3 \) and \( s_4 \) go to group 2). We describe an algorithm to find a least-sized consistent stochastic partition by casting it as an instance of the satisfiability problem over linear rational arithmetic.

Algorithm. The encoding is similar to the case of partitions in the previous subsection. To find a stochastic partition of size at most a given \( k \), let \( g_i \) denote the group \( i \) for \( 1 \leq i \leq k \). Introduce a non-negative rational variable \( v(s)(i, j) \) to denote \( s(g_i)(g_j) \) for every \( s \in S_P \), \( 1 \leq i, j \leq k \). For every \( i \) and \( s \in S_P \), add the constraint \( \bigvee_{1 \leq j \leq k} v(s)(i, j) = 1 \) \( \forall \sum_{1 \leq j \leq k} v(s)(i, j) = 0 \) to denote that \( s(g_i) \) is a distribution or is undefined. Then, we encode points 1 and 2 of Definition 9 by adding the constraint \( v(s)(i, j) = 1 \) for every \( i \) and \( P \in \overline{P} \), making \( g_1 \) the start state of the quotient, and adding

\[
\sum_{1 \leq j \leq k} v(s)(i, j) = 1 \iff \sum_{1 \leq i \leq k} v(\par(s))(i, i) > 0
\]

for every non-root state \( s \) and \( i \). This ensures that the stochastic partition obtained is well-defined.

Encoding consistency is the same as before except for \( \sigma_{t_n, i, t_p} \) \((t_n, i \) and \( t_p \) are as before\) which will now be

\[
d_{\mu_n, \mu_p, i} \land \sum_{1 \leq j \leq k} v(s)(j, i) > 0.
\]

where \( d_{\mu_n, \mu_p, i} \), denotes \( \mu_n \subseteq_R \lift(\mu_p, g_i) \). Thus, we will check if there is a group of \( \par(s_p) \) (summation over \( 1 \leq j \leq k \)) for which \( s_p \in g_i \) and \( \mu_n \subseteq_R \lift(\mu_p, g_i) \). For a \( j \), \( \lift(\mu_p, g_i)(g_j) \) is encoded as \( \sum_{s \in \Supp(\mu_p)} v(s)(i, j) \cdot \mu_p(s) \). Rest of the encoding is similar.

We can similarly show the correctness of the encoding and the termination of the algorithm follows from Corollary 2.

Theorem 3. The problem of learning a minimum state consistent LPTS with \( \overline{P} \) and \( \mathcal{N} \) is decidable.

IV. ACTIVE LEARNING FOR LPTSES

We now consider the problem of learning the language of an LPTS, i.e., learning an LPTS up to simulation equivalence (following Lemma 5), in the framework of active learning. Let \( U \) be an unknown target LPTS. The learning framework has a learner and a teacher. The goal of the learner is to learn an LPTS \( L \) such that \( L \approx U \). To that effect, the learner maintains a hypothesis LPTS \( H \). The process of learning proceeds in rounds where in each round, the learner makes a query to the teacher and updates \( H \) based on the response. For reasons mentioned in the introduction, we only consider a single type of queries in this paper where the learner conjectures \( H \) as (simulation) equivalent to \( U \). In response to such a query, the teacher is expected to check whether \( H \approx U \) holds and otherwise, return a counterexample. If it is a counterexample to \( H \not\approx U \) \((U \not\approx H) \), it is called a negative (positive) counterexample. Following Section 11 we assume that the counterexamples are always trees. Furthermore, there should always exist an LPTS consistent with all of the counterexamples, i.e., simulating all the positive counterexamples and
none of the negative counterexamples, received by the learner so far. Also, every conjecture \( H \) made by the learner should be consistent with the counterexamples received so far, in the above sense.

Unfortunately, the framework, as described above, is too general to be useful, as the following lemma shows.

**Theorem 4.** The problem of learning an unknown LPTS \( U \) is undecidable in the active learning framework.

**Proof Sketch:** We show that there is no algorithm to learn the unknown target \( U_\lambda \), which first performs an action \( a \) and goes to a state with (unknown) probability \( \lambda \) to loop on action \( b \) or goes to another state with the remaining probability to deadlock, by describing an adversarial teacher which manipulates the value of \( \lambda \) as necessary to keep generating counterexamples. After choosing an initial value of \( \lambda \), the teacher returns a counterexample as long as the hypothesis is not simulation equivalent to the target. If a hypothesis simulation equivalent to the target is conjectured, the teacher increases the value of \( \lambda \) just enough to have the new target not simulated by the hypothesis, while still being consistent with all the previously generated counterexamples, and a new (positive) counterexample can then be generated.

The main reason behind the theorem is that it is not necessary for the positive tree counterexamples returned by the learner to have an execution mapping to \( U \) (see Section \[I\]). Such a teacher can be seen as an adversary which can choose the probability values in the counterexamples returned, which are infinitely many, to make the learner never converge to the desired probabilities.

But, in practice, to be able to apply the learning framework in a given setting, one needs to implement the teacher’s algorithm and we are not aware of any algorithm to generate counterexamples other than the one discussed in Section \[I\]. As mentioned before, this algorithm has an interesting property that the generated counterexamples have an execution mapping to \( L_1 \) when \( L_1 \preceq L_2 \) fails. This suggests us to impose the following friendliness condition on a teacher.

**Condition 1** (Friendly Teacher). Every positive (negative) counterexample returned by the teacher should have an execution mapping to \( U \) \((H)\).

First of all, we observe that the proof of Theorem \[I\] no longer works because an update to \( \lambda \) may violate Condition 1 on any positive counterexample already returned. In fact, as we show below, the problem becomes decidable. Let \( \mathcal{P} \) and \( \mathcal{N} \) denote the sets of positive and negative counterexamples, returned by the teacher so far, respectively. First, consider the pseudo-code in Algorithm \[I\] It suggests a method of using the algorithms described in Section \[III\] by treating \( \mathcal{P} \) and \( \mathcal{N} \) as the tree samples. There is a choice at line 6 to use partitions or stochastic partitions.

First, we show that using traditional partitions at line 6 makes the problem of learning a target decidable.

**Lemma 11.** The active learning loop of Algorithm \[I\] terminates under Condition \[4\] on the teacher and using partitions at line 6 with the number of states of each intermediate hypothesis \( U \) bounded by that of \( U \).

**Proof Sketch:** Consider an arbitrary iteration of the learning loop. First of all, due to Condition \[1\] the quotient of the partition induced by the execution mappings from the positive counterexamples to \( U \) is a sub-structure of \( U \) and hence, is trivially simulated by \( U \) and is a consistent LPTS. As the algorithm finds a least-sized consistent partition, its size is bounded by \( |S_U| \).

Then, notice that every future hypothesis is consistent with any new counterexample returned, and hence, is distinct from the current one. Moreover, due again to Condition \[1\] and as lift only adds probabilities, one can show that there are only finitely many possible distributions for a given partition size.

We conclude that the algorithm terminates.

Thus, we have the following result.

**Theorem 5.** The problem of learning an unknown LPTS is decidable in the active learning framework, with Condition \[7\] on the teacher.

It is sometimes desirable to learn an LPTS with the least number of states. While the algorithm described above learns an LPTS, it is not guaranteed to output a minimum state LPTS simply because each hypothesis need not have the least number of states (see Section \[III-A\]). This suggests us to impose the following condition on the learner.

**Condition 2** (Learner). Every hypothesis \( H \) made by the learner is a minimum state LPTS consistent with \( \mathcal{P} \) and \( \mathcal{N} \).

If there is a learning algorithm under Conditions \[1\] and \[2\] then it is guaranteed to output a minimum state LPTS which is (simulation) equivalent to \( U \). But, there is no such algorithm as we show below.

**Theorem 6.** The problem of learning an unknown LPTS \( U \) is undecidable in the active learning framework, with both Condition \[7\] on the teacher and Condition \[2\] on the learner.

**Proof Sketch:** We show that there is no algorithm to learn (unknown) \( H_1 \) in Figure \[6\] by describing an adversarial teacher which can return a counterexample for any conjectured hypothesis. Initially, the teacher keeps returning negative counterexamples, if there are transitions on actions other than \( a, b \) and \( c \) in the hypothesis, or the positive counterexample \( P \) in Figure \[5\] until the learner conjectures a single-state LPTS with
self-loops on these three actions. Thereafter, if a conjectured hypothesis has transitions on only $a$, $b$, and $c$, and simulates $P$, the teacher returns $N_a$ to force the future hypotheses to have at least two states and in every future round, returns $N_b$ or $N_c^2$ in the figure, as necessary. One can show that there are always suitable values of $\beta$ and $\gamma$ whenever $N_c^2$ needs to be returned and the learner always conjectures a two state LPTS. In fact, $H_3$ is always a consistent LPTS for a suitable $\lambda \in (0, 1)$.

However, we obtain a semi-algorithm to the problem by using stochastic partitions at line 6 of Algorithm $1$. That is, if the algorithm terminates, it is guaranteed to learn the target with the least number of states. Correctness is immediate from Theorem $3$.

V. LEARNING ASSUMPTIONS FOR
COMPOSITIONAL REASONING

As mentioned in the introduction, the original motivation for this work was to automate assume-guarantee style reasoning for simulation conformance. Assume-guarantee reasoning is a compositional technique that breaks up the verification of large systems into that of its components for increased scalability. When checking individual components, the method uses assumptions about their environments and discharges them on the rest of the system. For a system of two components, such reasoning is captured by the following simple assume-guarantee rule (ASYM).

\[
L_1 \parallel A \preceq P \quad L_2 \preceq A \quad L_1 \parallel L_2 \preceq P
\]

Several other assume-guarantee rules have been proposed, some of them involving symmetric or circular reasoning. Despite its simplicity, rule ASYM has been proven most effective in practice and has been studied extensively mainly in a non-probabilistic setting, for different notions of conformance.

In our case, $L_1, L_2, A$, and $P$ are LPTSes with $P$ standing for the specification which the composition $L_1 \parallel L_2$ should conform to, where $\parallel$ is defined below.

Definition 11 (Composition $2^8$). The parallel composition of $L_1$ and $L_2$, denoted $L_1 \parallel L_2$, is defined as the LPTS $(S_1 \times S_2, (s_1^0, s_2^0), \alpha_1 \cup \alpha_2, \tau)$ where $(s_1, s_2) \rightarrow \mu$ iff

1) $s_1 \rightarrow_1 \mu_1$, $s_2 \rightarrow_2 \mu_2$ and $\mu = \mu_1 \otimes \mu_2$, or
2) $s_1 \rightarrow_1 \mu_1$, $a \notin \alpha_2$ and $\mu = \mu_1 \otimes \delta_{s_2}$, or
3) $a \notin \alpha_1$, $s_2 \rightarrow_2 \mu_2$ and $\mu = \delta_{s_1} \otimes \mu_2$.

Here $v_1 \otimes v_2 \in \text{Dist}(S_1 \times S_2)$, such that $v_1 \otimes v_2 : (s_1, s_2) \mapsto v_1(s_1) \cdot v_2(s_2)$, for $v_1 \in \text{Dist}(S_1), v_2 \in \text{Dist}(S_2)$.

The main challenge in using assume-guarantee reasoning is to automatically come up with a small assumption $A$ satisfying the premises. We first note that the proposed rule is sound and complete. Completeness, obtained trivially by replacing $A$ with $L_2$, is essential to guarantee termination of our proposed algorithm. Previous attempts at automating assume-guarantee reasoning using learning in a probabilistic setting have been restricted to checking probabilistic reachability properties using either an incomplete rule or algorithms which may not terminate.

Motivated by the success of existing applications of active learning to assume-guarantee reasoning, we propose to use the active learning framework presented in Section IV to learn an intermediate assumption $A$ in the rule ASYM. We describe an algorithm for the problem using learning and show termination below.

Teacher. The teacher is implemented by two conformance checks corresponding to the two premises of the rule, checked in any order.

- **Premise 1** guides the learner towards a conjecture that makes $L_1 \parallel A \preceq P$ true.
- **Premise 2** guides the learner towards a conjecture that is discharged on $L_2$, i.e. that makes $L_2 \preceq A$ true.

If the conjectured $A$ satisfies both the premises, soundness of ASYM implies $L_1 \parallel L_2 \preceq P$ holds, and the teacher returns $\text{true}$. If one of the premises fails, the teacher generates counterexamples with an execution mapping (Section I). Thus, the teacher satisfies Condition $1$. When premise 2 fails, a positive counterexample is returned to the learner. When premise 1 fails, the obtained counterexample is first projected onto $A$ and then returned as a negative counterexample. As a counterexample $C$ to premise 1 has an execution mapping to $L_1 \parallel A$, the projection onto $A$ is simply the contribution of $A$ towards $C$ in the composition. To enable this, additional information regarding individual distributions is maintained during composition.

Spuriousness Check. Note that if $L_1 \parallel L_2 \not\preceq P$, no assumption satisfies both the premises of ASYM (violating the assumption on the existence of a consistent LPTS in Section I). To detect this, the learner needs to check if a counterexample returned by the teacher exposes the failure of the conclusion of ASYM. A real counterexample would imply that the specification will not hold of the original system while a spurious one would need the learner to revise its hypothesis for the assumption. We restrict spuriousness check to negative counterexamples following previous approaches.

A simple way is to check $N \preceq L_2$ for a negative counterexample $N$. $N$ is real if the check succeeds and spurious, otherwise. A slightly more involved, but practical, way is described elsewhere.

Algorithm. Now, the learner can simply use Algorithm $1$ using partitions, to learn an intermediate assumption. As the positive (negative) counterexamples have execution mapping to $L_2$ ($A$), it is as if the unknown target is $L_2$. Note that if $P$ holds of the system, $L_2$ is clearly an assumption satisfying the premises. However, the algorithm is expected to terminate with a smaller assumption in practice, which also satisfies the premises. If $P$ does not hold, the algorithm terminates with a real counterexample. Termination is guaranteed by Lemma $1$. If we also impose Condition $2$, the learner uses stochastic partitions in Algorithm $1$ giving a semi-algorithm.

Complexity Analysis. Let us now analyze the complexity of assume-guarantee reasoning using the learning algorithm described above (with partitions). The complexity of checking...
Let $d = |\tau_2|$ and $b$ be the maximum size of the support of a distribution in $L_2$. Given a state of a candidate assumption of size $k$ and a distribution of $L_2$, there can be at most $k^d$-many corresponding distributions (due to non-determinism) from that state. For $k$ states and $d$ distributions, this gives a total of $dk^{b+1}$. Therefore, there are $2^{dk^{b+1}}$ different possible candidates of size $k$ to consider. The total number of iterations of the learning algorithm is then bounded by $\sum_{k=1}^{m} 2^{dk^{b+1}} = \mathcal{O}(m2^{dm^{b+1}})$, where $m$ is the number of states in the final assumption output by the algorithm.

At each iteration, in the worst-case, the algorithm enumerates all the candidate assumptions of the current size $k$ and performs simulation checks with all the negative counterexamples. These checks have a complexity of $\mathcal{O}(\text{poly}(|A|, |\mathcal{N}|, l))$, where $A$ is the final assumption, $\mathcal{N}$ is the final set of negative counterexamples and $l$ is the largest $|N|$, for any $N \in \mathcal{N}$. Thus, the total worst-case complexity of the learning algorithm for computing the final assumption is $\mathcal{O}(\text{poly}(|A|, |\mathcal{N}|, l) \cdot m2^{dm^{b+1}})$. Furthermore, the complexity of checking the two premises of ASYM is $\mathcal{O}(\text{poly}(|L_1| \cdot |A| \cdot |P|) + \text{poly}(|L_2|, |P|))$ at every iteration. We observe that in practice, if the assumption is small (i.e., $|A| \ll |L_2|$) this approach can be better than checking $L_1 \parallel L_2$ directly. In other cases, however, we would need better algorithms to address the problem. We leave this for future work.

VI. CONCLUSION

We have presented algorithms and decidability results for the problem of learning non-deterministic LPTes from stochastic tree samples, using traditional and stochastic state-space partitioning. We have also described the application of the algorithms to automating the discovery of assumptions for the compositional verification of LPTes.

In the future, we would like to investigate further conditions on the teacher that will make the active learning problem with stochastic partitions decidable. We also plan to investigate the use of weak simulation for the conformance relation, as this will result in smaller assumptions for compositional verification. However, algorithms for checking weak simulation are not currently known. Finally we plan to investigate new applications for our algorithms in learning abstractions or active model checking and in domains other than verification.

ACKNOWLEDGMENTS

We thank Christel Baier, Rohit Chadha, Sagar Chaki, Lu Feng, Holger Hermanns, Marta Kwiatkowska, Joel Ouaknine, David Parker, Nishant Sinha, Frits Vaandrager, Mahesh Viswanathan, James Worrell and Lijun Zhang for answering our questions related to this research and the reviewers for their suggestions.

REFERENCES

[1] L. d. Alfaro, T. A. Henzinger, and R. Jhala, “Compositional Methods for Probabilistic Systems,” in CONCUR, ser. LNCS, vol. 2154. London, UK: Springer-Verlag, 2001, pp. 351–365.

[2] D. Angluin, “Learning Regular Sets from Queries and Counterexamples,” Information and Computation, vol. 75(2), pp. 87–106, Nov. 1987.

[3] D. Angluin and C. H. Smith, “Inductive Inference: Theory and Methods,” ACM Comp. Surv., vol. 15(3), pp. 237–269, September 1983.

[4] C. Baier, B. Engelen, and M. Majster-Cederbaum, “Deciding Bisimilarity and Similarity for Probabilistic Processes,” J. Comput. Syst. Sci., vol. 60(1), pp. 187–231, Feb 2000.

[5] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio, “Learning Functions Represented as Multiplicity Automata,” J. ACM, vol. 47(3), pp. 506–530, May 2000.

[6] R. C. Carrasco and J. Oncina, “Learning Deterministic Regular Grammars From Stochastic Samples in Polynomial Time,” RAIRO, vol. 33, pp. 1–20, 1999.

[7] R. C. Carrasco, J. Oncina, and J. Calera-Rubio, “Stochastic Inference of Regular Tree Languages,” Machine Learning, 44(1-2), pp. 185–197, July 2001.

[8] R. Chada and M. Viswanathan, “A Counterexample-Guided Abstraction-Refinement Framework for Markov Decision Processes,” TOCL, vol. 12(1), pp. 1–49, November 2010.

[9] S. Chaki, E. M. Clarke, N. Sinha, and P. Thati, “Automated Assume-Guarantee Reasoning for Simulation Conformance,” in CAV, ser. LNCS, vol. 3576. Springer-Verlag, 2005, pp. 534–547.

[10] Y.-F. Chen, A. Farzan, E. M. Clarke, Y.-K. Tsay, and B.-Y. Wang, “Learning Minimal Separating DFA’s for Compositional Verification,” in TACAS, ser. LNCS, vol. 5505. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 31–45.

[11] E. Clarke, D. Long, and K. McMillan, “Compositional Model Checking,” in LICS. Piscataway, NJ, USA: IEEE Press, 1989, pp. 353–362.

[12] E. M. Clarke, O. Grumberg, and D. A. Peled, Model Checking. Cambridge, MA, USA: MIT Press, 2000.

[13] C. de la Higuera and J. Oncina, “Learning Stochastic Finite Automata,” in ICGL, ser. LNCS, vol. 3264, Springer-Verlag, 2004, pp. 175–186.

[14] L. Feng, T. Han, M. Kwiatkowska, and D. Parker, “Learning-based Compositional Verification for Synchronous Probabilistic Systems,” in ATVA, ser. LNCS, vol. 6996. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 511–521.

[15] L. Feng, M. Kwiatkowska, and D. Parker, “Automated Learning of Probabilistic Assumptions for Compositional Reasoning,” in FASE, ser. LNCS, vol. 6603. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 2–17.

[16] P. García and J. Oncina, “Inference of Recognizable Tree Sets,” Universidad Politecnica de, Research Report DSIC - II/47/93, 1993.

[17] E. M. Gold, “Complexity of Automaton Identification from Given Data,” Information and Control, vol. 37(3), pp. 302–320, 1978.

[18] A. Gupta, K. L. McMillan, and Z. Fu, “Automated Assumption Generation for Compositional Verification,” FMSD, vol. 32(3), pp. 285–301, June 2008.

[19] A. Komuravelli, C. S. Păsăreanu, and E. M. Clarke, “Assume-Guarantee Abstraction Refinement for Probabilistic Systems,” in CAV, 2012, (to appear).

[20] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu, “Assume-Guarantee Verification for Probabilistic Systems,” in TACAS, ser. LNCS, vol. 6015. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 23–37.

[21] H. Mao et al., “Learning Probabilistic Automata for Model Checking,” in IJEST. Washington, DC, USA: IEEE Computer Society, 2011, pp. 111–120.

[22] R. Milner, “An Algebraic Definition of Simulation between Programs,” Stanford, CA, USA, Tech. Rep., 1971.

[23] A. L. Oliveira and J. P. Marques-Silva, “Efficient Search Techniques for the Inference of Minimum Size Finite Automata,” in SPRIE. IEEE Computer Society Press, 1998, pp. 81–89.

[24] J. Oncina and P. Garcia, “Identifying Regular Languages In Polynomial Time,” in ASSPR, vol. 5. World Scientific, 1992, pp. 99–108.

[25] A. Puget, “In Transition from Global to Modular Temporal Reasoning about Programs,” in LMCS, ser. NATO ASI, vol. 13. Springer-Verlag, 1998, pp. 123–144.

[26] C. S. Păsăreanu, D. Giannakopoulou, M. G. Bobaru, J. M. Cobleigh, and H. Barringer, “Learning to Divide and Conquer: Applying the L* Algorithm to Automate Assume-Guarantee Reasoning,” in FMSD, vol. 32(3), pp. 175–205, June 2008.

[27] M. O. Rabin, “Probabilistic Automata,” Information and Control, vol. 6(3), pp. 230–245, 1963.

[28] R. Segala and N. Lynch, “Probabilistic Simulations for Probabilistic Processes,” Nordic J. of Computing, vol. 2(2), pp. 250–273, June 1995.
Appendix

A. Proof of Lemma 2

By Definition 3, $\preceq$ is the union of all strong simulations. It can easily be shown that union of two strong simulations is a strong simulation and hence $\preceq$ is a strong simulation. It is also the coarsest as it includes any strong simulation.

B. Proof of Lemma 4

It suffices to show that $R \subseteq \preceq$ and $\preceq \subseteq R$.

$R$ is clearly a strong simulation which, by Lemma 2, implies $R \subseteq \preceq$.

To prove the other direction, let $s_1 \preceq s_2$. We show that $s_1R_s_2$ by induction on the height of $s_1$ in the tree $L_1$, where the height of a leaf state is defined to be 0 and the height of any other state is defined to be one plus the maximum height of any state in the support of any outgoing distribution from that state.

For the base case, let $s_1$ be any leaf state. As $s_1$ has no outgoing transitions, $s_1R_s_2$ trivially holds by the assumption on $R$.

For the inductive case, let the height of $s_1$ be non-zero and let $s_1 \xrightarrow{a} \mu_1$. Then, as $\preceq$ is a strong simulation (Lemma 2), there exists $\mu_2$ with $s_2 \xrightarrow{a} \mu_2$ such that $\mu_1 \preceq \mu_2$. Let $S \subseteq \text{Supp}(\mu_1)$. We then have $\mu_1(S) \leq \mu_2(\preceq (S))$. As every state in $\text{Supp}(\mu_1)$, and hence in $S$, has a smaller height than that of $s_1$, by induction hypothesis, $\preceq (S) \subseteq R(S)$ and therefore, $\mu_1(S) \leq \mu_2(R(S))$. As $S$ is arbitrary, we conclude that $\mu_1 \subseteq_R \mu_2$. By the assumption on $R$, we conclude that $s_1R_s_2$.

Thus, by induction, we conclude that $\preceq \subseteq R$.

C. Proof of Lemma 7

Let $P \in \mathcal{P}$. As $P \preceq L$, there is a strong simulation $R_P \subseteq S_P \times S_L$ with $s_P^0RPs_L^0$. As $P$ is a tree, $s_P^0$ is not in the support of any distribution and hence, assume without loss of generality that $R_P(s_P^0) = \{s_L^0\}$. Let $R = \bigcup_{P \in \mathcal{P}} R_P$. Now, $R$ induces a partition $\Pi$ of $S_P$ such that for $s_1, s_2 \in S_P$, $[s_1]_{\Pi} = [s_2]_{\Pi}$ iff $R(s_1) = R(s_2)$. Note that $[s_P^0]_{\Pi} = [s_Q^0]_{\Pi}$ for $P, Q \in \mathcal{P}$, satisfying the assumption on $\Pi$ in Definition 7.

The size of $\Pi$ is clearly bounded by $2^k$.

We first show that the relation $\ell' = \{(\Pi_P, \Pi_s)|s_PR_s|\}$ is a strong simulation. Let $eR\ell_1$ and $e \xrightarrow{a} \mu$. By Definition 7, there exists $s_p \in S_P$ and $\mu_p \in \text{Dist}(S_P)$ with $[s_p]_{\Pi} = e$, $s_p \xrightarrow{a} \mu_p$ and $\mu(e') = \sum_{s' \in e'} \mu_p(s')$ for all $e' \in E$. By the definition of $R'$ and $\Pi$, $R(s_1) = R(s_2)$ for all $s_1, s_2 \in e$ and hence, $s_PR_s$. As $\ell$ is the disjoint union of strong simulations, there exists $\mu_l \in \text{Dist}(S_L)$ such that $s_l \xrightarrow{a} \mu_l$ and $\mu_P \preceq_R \mu_l$. Let $E' \subseteq \text{Supp}(\mu)$. Now, $\mu(E')$
= \sum_{e' \in E'} \mu(e') \\
= \sum_{e' \in E'} \mu_p(\{s \in S_p \mid [s]_{\Pi} = e'\}) \quad \text{(choice of } \mu) \\
= \mu_p(\{s \in S_p \mid [s]_{\Pi} \in E'\}) \\
\leq \mu_l(\bigcup_{e' \in E'} R(\{s \in S_p \mid [s]_{\Pi} = e'\})) \quad \{\mu_p \sqsubseteq_R \mu_l\} \\
= \mu_l(\bigcup_{e' \in E'} R'(e')) \quad \text{(Def. of } R') \\
= \mu_l(R'(E')). \\

So, by Lemma 1, \mu \sqsubseteq_R \mu_l. We conclude that \( R' \) is a strong simulation. For an arbitrary \( P \in \mathcal{P} \), as \( s_0^0, Rs_0^0 \) and as \( s_0^0 \sqsubseteq_{\Pi} [s_0^0]_{\Pi} \) (Definition 7), \( s_0^0 R_s^0 s_L^0 \). Therefore, \( \mathcal{P} / \Pi \leq L \). ■

D. Proof of Lemma 8

Let \((g, a, \mu) \in \tau_{\mathcal{P} / \Pi}\) be arbitrary. It suffices to show that \( \mu \in \text{Dist}(G) \). This immediately implies that \( \mathcal{P} / \Pi \) is an LPTS, according to Definition 1. Let \((s, a, \mu_p) \in \tau_{\mathcal{P}}\) for some \( P \in \mathcal{P} \) such that \( s \in g \) and \( \mu = \text{lift}(\mu_p, g) \) as in Definition 10. Now, \( \sum_{g' \in G} \mu_g(g') = \)

\[ = \sum_{g' \in G} \sum_{s' \in g'} \mu_g(s') \cdot \mu_p(s') \\
= \sum_{s' \in S_p} \left( \mu_g(s') \cdot \sum_{g' : s' \in g'} \mu_g(s') \right) \\
= \sum_{s' \in S_p} \mu_g(s') \cdot \sum_{g' : s' \in g', (g') > 0} [s'](g') \cdot \mu_p(s') \\
= \sum_{s' \in S_p} \mu_g(s') \quad \{[s'](g') \in \text{Dist}(G)\} \\
= 1 \quad \{\mu_p \in \text{Dist}(S_p)\}. \\
■

E. Proof of Lemma 9

Let \( P \in \mathcal{P} \). We first show that the relation \( R = \{(s, g) \mid g \in G, s \in S_p \cap g\} \) is a strong simulation.

Let \( sRg \) and \( s \xrightarrow{a} \mu_p \). As \( s \in g \), by Definition 10 \( g \xrightarrow{a} \mu \) where for every \( g' \in G', \)

\[ \mu(g') = \sum_{s' \in g'} [s'](g')(g') \cdot \mu_p(s'). \]

It suffices to show that \( \mu_p \sqsubseteq_R \mu \). Let \( S \subseteq \text{Supp}(\mu_p) \). Now, \( \mu_p(S) \)

\[ = \sum_{s' \in S} \mu_p(s') \]

\[ = \sum_{s' \in S} \sum_{g : s' \in g'} [s'](g')(g') \cdot \mu_p(s') \quad \{[s'](g') \in \text{Dist}(G)\} \\
= \sum_{g' \in G} \sum_{s' : s' \in g'} [s'](g')(g') \cdot \mu_p(s') \\
= \sum_{g' \in G} \sum_{s' : s' \in g'} [s'](g')(g') \cdot \mu_p(s') \quad \{\text{definition of } R\} \\
\leq \sum_{g' \in G} \sum_{s' : s' \in g'} [s'](g')(g') \cdot \mu_p(s') \\
= \sum_{g' \in G} \mu(g') \quad \{\text{choice of } \mu\} \\
= \mu(R(S)) \\
= \mu(R) \]

So, by Lemma 1, \( \mu_p \sqsubseteq_R \mu \). We conclude that \( R \) is a strong simulation. From Definitions 9 and 10 \( s_0^0 \in s_0^0 \mathcal{P} / \Pi \) and hence, \( s_0^0 R_s^0 s_L^0 \). Therefore, \( P \leq \mathcal{P} / \Pi \). ■

F. Proof of Lemma 10

Let \( P \in \mathcal{P} \). As \( P \leq L \), there is a strong simulation \( R_p \subseteq S_p \times S_L \) with \( s_0^0, l \mathcal{P} / \Pi \). Let \( R = \bigcup_{P \in \mathcal{P}} R_p \). For \( sR_S \) and \( s \xrightarrow{a} \mu_p \), there can be one or more transitions \( s \xrightarrow{a} \mu_l \) with \( \mu_p \sqsubseteq_R \mu_l \). We assume that we can always choose a unique \( s \xrightarrow{a} \mu_l \) with \( \mu_p \sqsubseteq_R \mu_l \) (say, by ordering the possible transitions in some way and choosing the first) and also that we can always choose a unique weight function \( w \) satisfying the conditions of Definition 2 for \( \mu_p \sqsubseteq_R \mu_l \).

Create a group of states of \( S_p \) for each \( s \in S_L \), say \( \gamma(s_l) \), initialized to \( \emptyset \) and let \( \Gamma \) be the set of all these groups. We will populate these groups by induction on the depth of a state in \( S_p \) with \( s \in \gamma(s_l) \) implying \( sR_{\mathcal{P}} \). We will also define \( \varphi(s) : \Gamma \rightarrow \text{Dist}(\Gamma) \) for each \( s \in S_p \) by the same induction. Let \( s \in S_p \) be arbitrary. We proceed by induction on \( d(s) \), the depth of \( s \).

The base case is when \( d(s) = 0 \) implying \( s \) is a start state. \( s \) is added to \( \gamma(s^0_L) \) and \( \varphi(s)(g) \) maps every \( g \in \Gamma \) to \( \delta_{\gamma(s^0_L)}(g) \). Clearly, \( sR_{\mathcal{P}}^0 \) and \( \varphi(s)(g)(\gamma(s^0_L)) > 0 \) for every \( g \in \Gamma \).

For the inductive step, \( d(s) > 0 \) and let \( g \in \Gamma \). If \( \text{par}(s) \notin g \), \( \varphi(s)(g) \) is undefined. Otherwise, let \( s \in S_L \) be the unique state satisfying \( g = \gamma(s_l) \). Thus, \( \text{par}(s) \in \gamma(s_l) \) and by induction hypothesis, \( \varphi(s) \text{par}(s)R_{\mathcal{P}} \). Let \( s \xrightarrow{a} \mu_p \) be the unique transition with \( s \in \text{Supp}(\mu_p) \) (as \( \text{par}(s) \) is unique). As \( R \) is the disjoint union of strong simulatons, choose \( s \xrightarrow{a} \mu_p \) with \( \mu_p \sqsubseteq_R \mu_l \) as mentioned in the beginning in a unique way. Furthermore, let \( w \) be the uniquely chosen weight function satisfying the conditions in Definition 2 for \( \mu_p \sqsubseteq_R \mu_l \). For every \( s' \in S_L \) with \( w(s, s_l') > 0 \), define \( \varphi(s)(g)(\gamma(s^0_L)) = w(s', s_l')/\mu_p(s) \) and add \( s \) to \( \gamma(s_l) \). Now, \( w(s, s_l') > 0 \) implies \( sR_{\mathcal{P}}^0 \) by Definition 2. The definition
also says that $\sum_{s' \in S_L} w(s, s') = \mu_p(s)$ which implies that $\varphi(s)(g) \in \text{Dist}(\Gamma)$. Clearly, $\varphi(s)(g)(\gamma(s')) > 0$.

That completes populating $\Gamma$ and defining $\varphi(s)$ for every state $s \in S_P$. Note that if $\varphi(s)(g)$ is defined, then $\text{par}(s) \in g$ from the above construction and hence, $g$ is non-empty. Furthermore, every group $g$ in $\text{Supp}(\varphi(s)(g))$ contains $s$, again from the construction above, and hence, is non-empty.

Now, define a stochastic partition $\Pi = (G, \{[s]\}_{s \in S_P})$ with $G$ containing all the non-empty groups of $\Gamma$ and $[s]$ given by $\varphi(s)$. It is not difficult to see that $\Pi$ is well-defined according to Definition 2. First of all, one can easily show, using the same induction above, that every state is added to some group and hence $\bigcup G = S_P$. Then, as discussed above, $\varphi(s)$ is only defined for groups in $G$ and the support of any distribution in the range set of $\varphi(s)$ is contained in $G$ and hence, $\varphi(s) : G \rightarrow \text{Dist}(G)$. $(s'_L)$ is the $g^0$ in Definition 9. Also, from the way we populated groups in $G$, the condition that $s \in g$ iff there exists $g' \in G$ such that $[s](g')(g) > 0$ follows for every $s \in S_P$ and $g \in G$.

We will now show that $\mathcal{P}/\Pi \leq L$ by first proving that $R' = \{(g, s)g \in G, g = \gamma(s)\}$ is a strong simulation. Let $gR's_I$ and $g \triangleleft \mu$. By Definition 10 there exists $s \triangleleft \mu_p$ in some $P \in \mathcal{P}$ with $s \in g$ such that for every $g' \in G$,

$$\mu(g') = \sum_{s' \in g'} [s'](g')(g') \cdot \mu_p(s')$$

By definition of $R'$, $g = \gamma(s_I)$ and hence, $s \in \gamma(s_I)$. From the above construction of $\Pi$, we can then infer $sR's_I$. Now, choose $s_I \rightarrow \mu_1$ with $\mu_p \sqsubseteq_R \mu_1$ as mentioned in the beginning in a unique way. It suffices to show that $\mu \sqsubseteq_R \mu_1$. Let $w$ be the uniquely chosen weight function to show that $\mu_p \sqsubseteq_R \mu_1$.

Let $\gamma(s'_I) \in \text{Supp}(\mu)$. Then, $\mu(\gamma(s'_I))$

$$= \sum_{s' \in \gamma(s'_I)} [s'](g)(\gamma(s'_I)) \cdot \mu_p(s')$$

{choice of $\mu$ above}

$$= \sum_{s' \in \text{Supp}(\mu_p) \cap \gamma(s'_I)} [s'](g)(\gamma(s'_I)) \cdot \mu_p(s')$$

$$= \sum_{s' \in \text{Supp}(\mu_p)} w(s', s_I)$$

{from the above construction of $[s']$}

$$= \mu(s_I)$$

{Definition 2}

So, $\mu(g') = \mu_1(R'(g'))$ for every $g' \in \text{Supp}(\mu)$. As $R'$ maps distinct groups in $G$ to distinct states of $S_L$, it follows that $\mu \sqsubseteq_R \mu_1$ (by exhibiting the trivial weight function). We conclude that $R'$ is a strong simulation. Clearly, $s'_L = [s'_L] R' s'_L$. Therefore, $\mathcal{P}/\Pi \leq L$. Also, $|G| \leq |S_L| = k$. ■

G. Proof of Theorem 2

We give an example where it is impossible for the learner to converge to the unknown target, up to $\approx$, in presence of an adversarial teacher.

Consider $U_\lambda$ in Figure 7 where $\lambda \in (0, 1)$. For a fixed $\lambda$, $U_\lambda$ is an LPTS with the alphabet $\{a, b\}$. The strategy for an adversarial teacher is described in Algorithm 2 which is briefly summarized in words below. Let $U_\lambda$, for some unknown $\lambda$, be the unknown target and $H_n$ be the hypothesis at the beginning of every round $n \geq 1$ of the active learning loop (we count rounds beginning with 1). The teacher acts as an adversary by manipulating the value of $\lambda$ as necessary and it suffices to show that there is some LPTS consistent with all the counterexamples generated so far. So, let $\lambda_n$ be the value of $\lambda$ at the beginning of round $n$ and let $\mu_n$, be the corresponding distribution on $a$.

In every round $n$, the teacher first checks $H_n \leq U_\lambda$, returning a negative counterexample if it fails, and then checks $U_\lambda \leq H_n$, returning a positive counterexample if it fails. If both checks succeed (i.e. $U_\lambda \approx H_n$), the teacher modifies the value of $\lambda$ such that $U_{\lambda_n} \leq U_\lambda$ but not the other way around. This is achieved by incrementing its value at line 15, where $\text{Dist}_a[N]$ is the set of distributions labeled by $a$ in $N$. First, it computes $\lambda^+$ which is the least of all $\lambda^+\lambda$'s, greater than $\lambda$, and 1 where $\mu$ is any distribution appearing in a transition of any negative counterexample labeled by $a$ and $\lambda^+$ is the measure, under $\mu$, of all the states having a transition on $b$. It then updates $\lambda$ to the mean of $\lambda$ and $\lambda^+$, i.e. $\lambda_{n+1} = (\lambda_n + \lambda^+)/2$. After this update, as $\lambda > \lambda_n$, $U_{\lambda_n} \leq U_\lambda$ holds but $U_\lambda \not\leq U_{\lambda_n}$ and hence, $U_{\lambda_n} \not\leq H_n$. This ensures that a positive counterexample $P$ always exists, justifying line 16.

Now, it is easy to see that $\lambda^+$ at line 14 is well-defined and always exists. Thus, the teacher can return a counterexample for every hypothesis made by the learner.

We will now show that $U_{\lambda_n}$ is consistent with $\mathcal{P}$ and $\mathcal{N}$ at the beginning of each round $n \geq 1$ by induction on $n$, where $\mathcal{P}$ and $\mathcal{N}$ are the sets of positive and negative counterexamples, respectively. For $n = 1$, $\mathcal{P} \cup \mathcal{N} = \emptyset$ and hence, $U_{\lambda_1}$ is consistent.

Assume that $U_{\lambda_n}$ is consistent with $\mathcal{P} \cup \mathcal{N}$ for some $m \geq 1$. If a negative (positive) counterexample $N$ ($P$) is added to $\mathcal{N}$ ($\mathcal{P}$) at line 7 (11), $N \not\leq U_{\lambda_n} (P \not\subseteq U_{\lambda_n})$ by Definition 6. As $U_{\lambda_m} = U_{\lambda_{m+1}}, U_{\lambda_{m+1}}$ is consistent with $\mathcal{P}$ and $\mathcal{N}$. Now, let $P$ be a positive counterexample added to $\mathcal{P}$ at line 17. Clearly, $P \not\subseteq U_{\lambda_{m+1}}$ by Definition 6. Also, by induction hypothesis, for every $P' \in \mathcal{P} \setminus \{P\}$, $P' \not\subseteq U_{\lambda_{m+1}}$ (from above), we obtain $P^0 \not\subseteq U_{\lambda_{m+1}}$. From Lemma 6. Let $\mathcal{N} \in \mathcal{N}$. By induction hypothesis, $N \not\leq U_{\lambda_n}$ and we need to show that $N \not\leq U_{\lambda_n}$. For the sake of contradiction, assume that $N \leq U_{\lambda_{m+1}}$. 

![Fig. 7: There is no learner for the target $U_\lambda$, in presence of an unrestricted teacher.](image-url)
Now, every transition outgoing from $s_0$ is labeled by $a$, as the only transition outgoing from the start state of $U_{\lambda_{m+1}}$ is labeled by $a$. Let $s_0 \xrightarrow{a} \nu$. So, $\nu \subseteq \mu_{m+1}$. No state in $\text{Supp}(\nu)$ has a transition labeled by an action other than $b$, as otherwise, $\nu \not\subseteq \mu_{m+1}$. That is, every state $s$ in $\text{Supp}(\nu)$ either has no outgoing transition or has a transition labeled by $b$. One can easily argue that this is also the case for any transition outgoing from $s$ and so on. Consider $p_b'$, the measure of all the states having a transition on $b$ under $\nu$. We have that $p_b' \leq \lambda_{m+1}$, as otherwise, $\nu \not\subseteq \mu_{m+1}$.

If $p_b' \leq \lambda_m$, clearly $N \not\subseteq U_{\lambda_m}$ which leads to a contradiction. So, $p_b' > \lambda_m$. But then, by construction of $U_{\lambda_{m+1}}$ (line 14 of Algorithm 2), $\lambda_{m+1} < p_b'$ leading to a contradiction.

We conclude that $N \not\subseteq U_{\lambda_{m+1}}$. This completes the inductive step. Intuitively, whenever $\lambda$ is updated at line 15, it is as if the unknown target is $U_\lambda$ from the beginning and no inconsistencies arise.

Hence, the learner keeps receiving counterexamples and will never converge to the unknown target.

Algorithm 2 An adversarial teacher in the proof of Theorem 4

1: $n \leftarrow 1$
2: $\lambda \leftarrow$ arbitrary rational in $(0,1)$
3: $\mathcal{N} \leftarrow \emptyset$, $P \leftarrow \emptyset$
4: repeat
5: if $H_n \not\subseteq U_\lambda$ then
6: let $N$ be a tree counterexample (Def. 5)
7: $\mathcal{N} \leftarrow \mathcal{N} \cup \{N\}$
8: return $N$ to the learner as a negative counterexample
9: else if $U_\lambda \not\subseteq H_n$ then
10: let $P$ be a tree counterexample (Def. 5)
11: $P \leftarrow P \cup \{P\}$
12: return $P$ to the learner as a positive counterexample
13: else
14: $\lambda^+ = \min \{p_b' > \lambda \mid \mu \in \text{Dist}_b[N]\} \cup \{1\}$
15: $\lambda \leftarrow (\lambda^+ + \lambda)/2$
16: let $P$ be a tree counterexample to $U_\lambda \not\subseteq H_n$ (Def. 5)
17: $P \leftarrow P \cup \{P\}$
18: return $P$ to the learner as a positive counterexample
19: end if
20: $n \leftarrow n + 1$
21: until false

H. Proof of Theorem 4

We give an example where it is impossible for the learner to converge to the target, up to $\simeq$, in presence of an adversarial teacher.

Consider $H_1$ in Figure 6 as the unknown target $U$ and let $H_n$ be the hypothesis at the beginning of each round $n \geq 1$ (we count rounds beginning with 1) of the active learning loop. We describe a strategy of a teacher below to keep generating counterexamples no matter what the conjectured hypothesis is.

By Condition 2, $H_1$ is an LPTS with a single state, which is also the start state. Initially, in every round $n \geq 1$, the teacher first checks if $H_n$ has a transition on an action other than $a$, $b$ or $c$ in which case, clearly, $H_n \not\subseteq U$ and a negative tree counterexample is returned using the algorithm sketched in Section 1. Then, the teacher checks $P \simeq H_n$ and returns $P$ as a positive tree counterexample if it fails where $P$ is in Figure 5. Note that $P$ has an execution mapping to $U$ and hence, the teacher satisfies Condition 1. According to this strategy, the learner keeps receiving negative counterexamples for transitions on actions other than $a$, $b$ and $c$ or the positive counterexample $P$ which can go on forever, in which case we are done, or its hypothesis converges to the LPTS $H^*$ (disallowing duplicate transitions) with a single state and Dirac self-loops on $a$, $b$ and $c$. We will assume the latter, i.e. the learner conjectures $H^*$ after some finite number of rounds. Note that it is possible that $P$ has not yet been returned as a positive counterexample to the learner.

At this point, the teacher returns $N_a$ in Figure 5 as a negative counterexample. This forces every future hypothesis to have at least two states. In fact, the LPTS $H_2$ with two states in Figure 6 for any $0 < \lambda < 1$ is a consistent hypothesis. By Condition 2, the next hypothesis has only two states. Now, we describe the teacher’s strategy for future rounds. For this strategy, we show that a consistent LPTS of two states exists and that a counterexample can be returned, in every round. So, let $s_1$ and $s_2$ be the two states of the hypothesis with $s_1$ being the start state. Furthermore, let $\Delta_1^a$, $\Delta_1^b$, and $\Delta_1^c$ be the sets of distributions outgoing from $s_i$, $i = 1, 2$, on actions $a$, $b$ and $c$, respectively. The teacher’s strategy proceeds in every future round is as follows.

1) As in the initial strategy, it first checks if there is a reachable state in the hypothesis with a transition on an action other than $a$, $b$ and $c$ and returns a negative counterexample (see Section 1) if there is one.
2) Then, it checks $P \simeq H_n$ and returns $P$ as a positive counterexample if it fails.
3) At this point, $P \simeq H_n$ and $N_a \not\subseteq H_n$ hold ($H_n$ is consistent with them) and we infer the following.

(i) $\Delta_1^a \neq \emptyset$ and for every $\mu_a \in \Delta_1^a$, $H_a(s_1) < 1$ and
(ii) $\Delta_1^b \neq \emptyset$ and for every $\mu_b \in \Delta_1^b$ and every $s_i \in \text{Supp}(\mu_b)$, $\Delta_1^b \neq \emptyset$.

The teacher, therefore, does the following.

a) If there is a $\mu_b \in \Delta_1^b$ with $\mu_b(s_1) = 1$, it returns $N_b$ in Figure 5 as a negative counterexample. Clearly, $N_b$ has an execution mapping to $H_n$.

b) Otherwise, there exists a $\mu_b \in \Delta_1^b$ with $\mu_b(s_2) > 0$, implying $\Delta_1^c \neq \emptyset$ and $N_2^b \gamma$ in Figure 5 is returned as a negative counterexample, where $\beta = \mu_b(s_2)$ for some $\mu_a \in \Delta_1^a$ and $\gamma = \mu_c(s_2)$ for some $\mu_c \in \Delta_2^c$. Again, $N_2^c \gamma$ has an execution mapping to $H_n$.

Clearly, except for a counterexample generated in case 3(b) above, $H_\lambda$ is a consistent hypothesis for any $\lambda \in (0,1)$. For case 3(b), $H_\lambda$ with $0 < \lambda < \beta$ is consistent. So, after any round, $H_\lambda$ with $\lambda$ set to a value smaller than the least $\beta$ of any $N_2^b \gamma$ returned is consistent and such a $\lambda$ always exists as there are infinite rationals in $(0,1)$. Thus, Condition 2 forces the learner to always conjecture a two state LPTS and hence, it keeps receiving counterexamples and will never converge to $U$. ■