2012 27th Annual ACM/IEEE Symposium on Logic in Computer Science

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**

I. 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

1043-6871/12 $26.00 © 2012 IEEE
DOI 10.1109/LICS.2012.54

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 assumptions, 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 [14]. 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 LPTS.

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 $s \in S$, δ_s is the Dirac distribution on s, *i.e.* $\delta_s(s) = 1$ and $\delta_s(t) = 0$ for all $t \neq s$. For $\mu \in$ *Dist*(S), the *support* of μ, denoted *Supp*(μ), is defined to be the set $\{s \in S | \mu(s) > 0\}$ and for $X \subseteq S$, $\mu(X)$ stands for $\sum_{s \in X} \mu(s)$. 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).

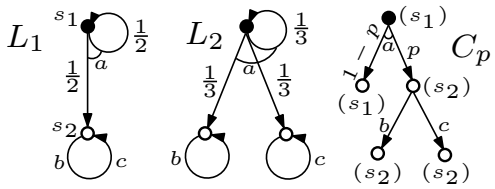

Fig. 1: Three reactive LPTSes. $p \in (0, 1)$ for C_p.

Definition 1 (LPTS)**.** *A* Labeled Probabilistic Transition System (*LPTS*) *is a tuple* $\langle S, s^0, \alpha, \tau \rangle$ *where* S *is a set of states,* $s^0 \in S$ *is a distinguished start state,* α *is a set of actions and* $\tau \subseteq S \times \alpha \times \text{Dist}(S)$ *is a probabilistic transition relation. For* $s \in S$*,* $a \in \alpha$ *and* $\mu \in \text{Dist}(S)$*, we denote* $(s, a, \mu) \in \tau$ *by* $s \xrightarrow{a} \mu$ *and say that* s *has a* transition *on* a *to* μ*.*

An LPTS is called reactive *if* τ *is a partial function from* $S \times \alpha$ *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 LTPSes. For example, Figure 1 shows three LPTSes. For $\mu = \{(s_1, \frac{1}{2}), (s_2, \frac{1}{2})\}$, L_1 has the transition $s_1 \xrightarrow{a} \mu$. 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 $\langle S_i, s_i^0, \alpha_i, \tau_i \rangle$ for an LPTS L_i and $\langle S_L, s_L^0, \alpha_L, \tau_L \rangle$ for an LPTS L.

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 (LTSes), a pair of states belonging to a strong simulation relation depends on whether certain other

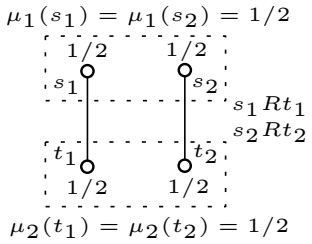

Fig. 2: A simple example where matching probabilities (solid edges) directly proves $\mu_1 \sqsubseteq_R \mu_2$.

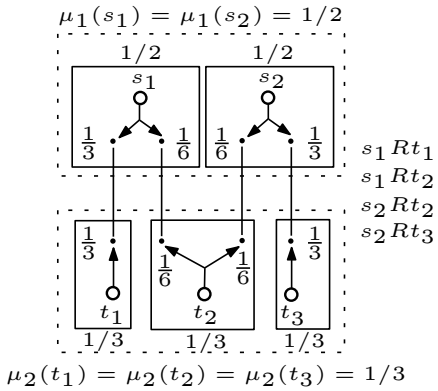

Fig. 3: An example where probabilities are split (arrows) before matching (solid edges) to prove $\mu_1 \sqsubseteq_R \mu_2$.

pairs of successor states also belong to the relation [22]. For LPTSes, one has successor *distributions* instead of successor states; a pair of states belonging to a strong simulation relation R 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, $\sqsubseteq_R$, which depends on the relation R between states. Intuitively, two distributions can be related if we can pair the states in their support sets, the pairs contained in R, *matching all* the probabilities under the distributions.

Consider an example with sRt and the transitions $s \xrightarrow{a} \mu_1$ and $t \xrightarrow{a} \mu_2$ with μ_1 and μ_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 $s_1 R t_1$ and $s_2 R t_2$ 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 $s_1 R t_1$, $s_1 R t_2$, $s_2 R t_2$ and $s_2 R t_3$ also hold, we say that $\mu_1 \sqsubseteq_R \mu_2$. Note that there can 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, $\mu_1 \in \mathit{Dist}(S_1)$, $\mu_2 \in \mathit{Dist}(S_2)$ and $R \subseteq S_1 \times S_2$.

Definition 2 ([28]). *$\mu_1 \sqsubseteq_R \mu_2$ 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 $s_1 R s_2$ 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 [4]. 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 μ_1 and μ_2. Considering *Supp*(μ_1) and *Supp*(μ_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 *Supp*(μ_1).

Lemma 1 ([30]). *$\mu_1 \sqsubseteq_R \mu_2$ iff for every $S \subseteq$ Supp(μ_1), $\mu_1(S) \leq \mu_2(R(S))$.*

It follows that when $\mu_1 \not\sqsubseteq_R \mu_2$, there exists a witness $S \subseteq \mathit{Supp}(\mu_1)$ 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 μ_1 cannot be matched and $S = \{s_2\}$ is a witness subset.

Definition 3 (Strong Simulation [28]). *R is a* strong simulation *iff for every $s_1 R s_2$ and $s_1 \xrightarrow{a} \mu_1^a$ there is a μ_2^a with $s_2 \xrightarrow{a} \mu_2^a$ and $\mu_1^a \sqsubseteq_R \mu_2^a$.*

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 $s_1 T s_2$. L_2 strongly simulates L_1, also denoted $L_1 \preceq L_2$, iff $s_1^0 \preceq s_2^0$. For the latter, alternatively, we say that simulation conformance *holds between L_1 and L_2.*

Definition 4 (Strong Simulation Equivalence). *The* strong simulation equivalence*, denoted $\simeq$, is defined as the kernel of strong simulation,* i.e. $\simeq = \preceq \cap \succeq$.

Definition 3 generalizes the one in the non-probabilistic setting [22] and has the following immediate consequence.

Lemma 2. *$\preceq \subseteq S_1 \times S_2$ is the coarsest strong simulation,* i.e. *$\preceq$ is a strong simulation and contains every strong simulation.*

Simulation conformance is decidable in polynomial time [4] and can be checked with a greatest fixed point algorithm that computes the coarsest simulation between L_1 and L_2. The algorithm uses a relation variable R initialized to $S_1 \times S_2$ and it checks the condition in Definition 3 for every pair in R, iteratively, removing any violating pairs from R. The algorithm terminates when a fixed point is reached showing $L_1 \preceq L_2$ or when the pair of start states is removed showing $L_1 \not\preceq L_2$. Several optimizations exist [30] but we do not consider them here, for simplicity.

Lemma 3 ([28]). *$\preceq$ is a preorder (i.e. reflexive and transitive).*

Finally, we find the following characterization of $\preceq$ useful in the algorithms we will discuss later on.

Lemma 4. *Let L_1 be a tree and $s_1 R s_2$ iff for every $s_1 \xrightarrow{a} \mu_1$, there exists $s_2 \xrightarrow{a} \mu_2$ with $\mu_1 \sqsubseteq_R \mu_2$. Then, $R = \preceq$.*

Proof Sketch: $R \subseteq \preceq$ by Def. 3. $\preceq \subseteq R$ can be proved by induction on the *height* of a state of L_1 using Lemma 2. ■

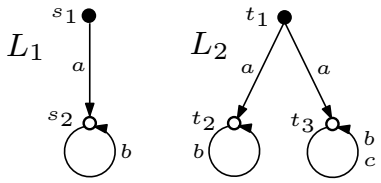

Fig. 4: An example showing that Lemma 4 does not hold, in general, if L_1 is not a tree. Let $R = \{(s_1, t_1), (s_2, t_2)\}$. Note that $\preceq = \{(s_1, t_1), (s_2, t_2), (s_2, t_3)\}$ and $R \subset \preceq$.

Note that the condition on R in the lemma is stronger than the one to make it a strong simulation (Definition 3). Also, if L_1 is not a tree, we can only conclude that $R \subseteq \preceq$, in general. See Figure 4 for an example where $R \subset \preceq$.

Counterexamples to $\preceq$. In the active learning problem we are interested in (Section IV), 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 $\{L' | L'$ is an LPTS and $L' \preceq L\}$.*

Lemma 5. *$L_1 \preceq L_2$ 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 $L_1 \preceq L_2$ is an LPTS C such that $C \in \mathcal{L}(L_1) \setminus \mathcal{L}(L_2)$, i.e. $C \preceq L_1$ but $C \not\preceq L_2$.*

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 ([19])**.** *If $L_1 \not\preceq L_2$, 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 1. ■

For example, C_p in Figure 1, for $p \in (0, \frac{1}{2}]$, is a counterexample to $L_1 \preceq L_2$. In another work, we showed that structures simpler than trees are not sufficient as counterexamples, even when one of the models is reactive [19].

We note an important feature of the algorithm used to prove the above theorem [19]. A counterexample C generated by the algorithm is essentially a finite *tree execution* of L_1. That is, there is a total mapping $M : S_C \to S_1$ 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 *Supp*(μ_c) is an injection and for every $c' \in$ *Supp*(μ_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 $p = \frac{1}{2}$ in Figure 1. While our algorithm always generates counterexamples with an *execution mapping*, it is possible to have a tree counterexample, as per Definition 6, without such a mapping. For example, C_p in Figure 1 for $p \in (0, \frac{1}{2})$ is also a counterexample with no such *execution mapping*. The condition we impose on a teacher in the active learning problem (Section IV) 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 $\mathcal{P} \subseteq \mathcal{L}(L)$ and $\mathcal{N} \cap \mathcal{L}(L) = \emptyset$, *i.e.* $P \preceq L$ for every $P \in \mathcal{P}$ and $N \preceq L$ 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 $L_{\mathcal{P}}$, trivially satisfies $P \preceq L_{\mathcal{P}}$ for every $P \in \mathcal{P}$. Now, if L is a consistent LPTS, it can be shown that $L_{\mathcal{P}} \preceq L$ and hence, by Lemma 3, $L_{\mathcal{P}}$ is also consistent. Thus, one can easily check, in polynomial time, if there exists a consistent LPTS by checking $N \preceq L_{\mathcal{P}}$ for every $N \in \mathcal{N}$. For this reason, we always assume the existence of a consistent LPTS. Clearly, the size of $L_{\mathcal{P}}$ 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 V). The algorithms we propose draw inspiration from the ones used to infer consistent non-probabilistic automata from counterexample traces [24], [16], [6], [18] which are based on partitioning the state space of the counterexamples. Let $S_{\mathcal{P}} = \bigcup_{P \in \mathcal{P}} S_P$ and $S_{\mathcal{N}} = \bigcup_{N \in \mathcal{N}} S_N$. First, we consider an algorithm based on the traditional state space partitioning of $S_{\mathcal{P}}$. 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 IV, 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 $S_{\mathcal{P}}$. For a partition Π of $S_{\mathcal{P}}$, let E_Π denote the set of equivalence classes under Π and for a state $s \in S_{\mathcal{P}}$, we let $[s]_\Pi$ denote the equivalence class of s (we drop the subscript Π when it is clear from the context). We always assume that $[s^0_P]_\Pi = [s^0_Q]_\Pi$

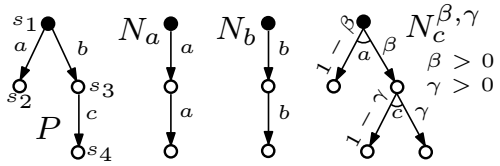

Fig. 5: Positive (P) and negative ($N_a, N_b, N_c^{\beta,\gamma}$) tree samples.

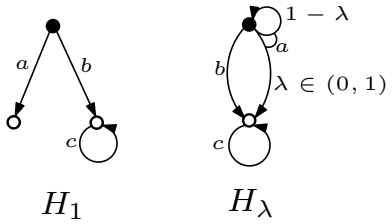

Fig. 6: Quotients for least size partition (H_1) and stochastic partition (H_λ) of P in Figure 5.

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* Π *of* $S_\mathcal{P}$*, define the* quotient LPTS*, denoted* $\mathcal{P}/\Pi$*, as the LPTS* $\langle E_\Pi, e^0, \alpha, \tau\rangle$ *where* $e^0 = [s_P^0]_\Pi$ *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* $\text{lift}(\mu_p)(e') = \sum_{s'\in e'} \mu_p(s')$ *for all* $e' \in E_\Pi$*.*

It can be easily shown that a quotient is always a well-defined LPTS. In the following, Π is a partition of $S_\mathcal{P}$.

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

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

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

Thus, we reduce the problem of finding a consistent LPTS to that of finding a consistent partition. As we show below, we can always find a consistent partition with a *bounded size*, where the *size* of Π is $|E_\Pi|$.

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

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 $s_P^0 R_P s_L^0$. 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 Π of $S_\mathcal{P}$ such that for $s_1, s_2 \in S_\mathcal{P}$, $[s_1]_\Pi = [s_2]_\Pi$ iff $R(s_1) = R(s_2)$. Note that $[s_P^0]_\Pi = [s_Q^0]_\Pi$ for $P, Q \in \mathcal{P}$. The size of Π is clearly bounded by 2^k. Now, we can show that $\{([s_p]_\Pi, s_l) | s_p R s_l\}$ 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 $S_\mathcal{P}$ of size at most exponential in $|S_L|$. While there may be a better bound, this way of partitioning $S_\mathcal{P}$ can not 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_λ, where λ is any value in $(0, 1)$, is another consistent LPTS with one less state.

Algorithm. A naïve algorithm for finding a *least-sized consistent partition* is to enumerate all the partitions of $S_\mathcal{P}$, 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 $1 \le i \le k$. For each i and state $s \in S_\mathcal{P}$, we introduce a new boolean variable, say $v_{[s]=i}$, to denote $[s] = e_i$. We add the constraint $\texttt{xor}(v_{[s]=1}, \ldots, v_{[s]=k})$ for every $s \in S_\mathcal{P}$ 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 $v_{[s_P^0]=1}$ 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_N^0, e_1) is not in the coarsest 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 $R_{s,i}$ to denote that $s \in S_\mathcal{N}$ is related to e_i by the coarsest strong 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 $1 \le i \le k$. Consider the expression $d_{\mu_n,\mu_p} \wedge v_{[s_p]=i}$, denoted σ_{t_n,i,t_p}. If d_{μ_n,μ_p} denotes $\mu_n \sqsubseteq_R \mathit{lift}(\mu_p)$, then this expression has the meaning that $[s_p] = e_i$ and the transition corresponding to t_p in the quotient, *viz.* $e_i \xrightarrow{a} \mathit{lift}(\mu_p)$, simulates t_n. If $X(s)$ denotes the set of all transitions outgoing from $s \in S_\mathcal{N}$, $Y(a)$ denotes the set of all transitions in $\mathcal{P}$ on action a and $\mathit{act}(t)$ denotes the action for the transition t, we add

$$R_{s,i} \iff \bigwedge_{t_n \in X(s)} \bigvee_{t_p \in Y(\mathit{act}(t_n))} \sigma_{t_n,i,t_p}$$

according to Lemma 4.

$\mathit{lift}(\mu_p)(e_i)$ can be encoded as $\sum_{s\in \mathit{Supp}(\mu_p)} l_{\mu_p,i,s}$ where $l_{\mu_p,i,s}$ denotes the *contribution* of s to the lifted probability of e_i under μ_p and satisfies

$$(v_{[s]=i} \implies l_{\mu_p,i,s} = \mu_p(s)) \wedge (\neg v_{[s]=i} \implies l_{\mu_p,i,s} = 0).$$

d_{μ_n,μ_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_{μ_n,μ_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 $\sqsubseteq_R$ holds between the distributions. We also introduce a variable for the witness subset $S \subseteq \mathit{Supp}(\mu_p)$ and encode the condition of Lemma 1 when $\sqsubseteq_R$ fails to hold. This variable for the witness subset can, in turn, be encoded using individual boolean variables for each $s \in \mathit{Supp}(\mu_p)$. 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 $\neg R_{s^0_N,1}$ for every $N \in \mathcal{N}$.

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 $S_\mathcal{P}$ 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 $S_\mathcal{P}$ 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 $S_\mathcal{P}$ (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 $S_\mathcal{P}$ to have the above one-to-one correspondence with S_L. Consider Figure 3 again and let μ_1 be outgoing from the root of some tree P in $\mathcal{P}$ and μ_2 appear in L. Let there be three groups (initially empty), one per state in $\mathit{Supp}(\mu_2)$, say G_{t_1}, G_{t_2} and G_{t_3}. As explained in Section II, having $\mu_1 \sqsubseteq_R \mu_2$, 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 $S_\mathcal{P}$. 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 $\mathit{Supp}(\mu_1)$ is matched with that of some split of exactly one state in $\mathit{Supp}(\mu_2)$, one can also think of the above grouping in the following alternative way. As the probability of $\frac{1}{2}$ for s_1 is split into $\frac{1}{3}$ and $\frac{1}{6}$, s_1 can be seen as being put in G_{t_1} with probability $\frac{1/3}{1/2} = \frac{2}{3}$ and in G_{t_2} with probability $\frac{1/6}{1/2} = \frac{1}{3}$. 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 $s_1[t_1]$. As the corresponding probability of $\frac{1}{3}$ is matched with that of some split of t_1 (implying $s_1 R t_1$), and as s_1 is not in the support of any distribution other than μ_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 $\sqsubseteq_R$). Similarly, for $s_1[t_2]$ and t_2. Now, if μ_3 is a distribution outgoing from s_1 in P, we may want to relate it to a distribution μ outgoing from t_1 (for $s_1[t_1]$) and another distribution μ' outgoing from t_2 (for $s_1[t_2]$). For a state $s_3 \in \mathit{Supp}(\mu_3)$, considering $\mu_3 \sqsubseteq_R \mu$ and $\mu_3 \sqsubseteq_R \mu'$ 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 $\mathit{par}(\cdot)$, when grouping a state in $S_\mathcal{P}$.

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

Definition 9 (Stochastic Partition)**.** *A* stochastic partition *of $S_\mathcal{P}$ is a tuple $(G, \{[s]\}_{s \in S_\mathcal{P}})$ where $G \subseteq 2^{S_\mathcal{P}}$ and $[s] : G \to \mathrm{Dist}(G)$ for every $s \in S_\mathcal{P}$, such that $\bigcup G = S_\mathcal{P}$ 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 $s \in S_\mathcal{P}$ and $g \in G$, $[s](g)$ is defined iff $[\mathrm{par}(s)](g')(g) > 0$ for some $g' \in G$.*

Furthermore, $s \in g$ iff $[s](g')(g) > 0$ for some $g' \in G$, for every $s \in S_\mathcal{P}$ and $g \in G$.

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

Here, G denotes the groups mentioned above and $[s]$ denotes the *stochastic* grouping of $s \in S_\mathcal{P}$ 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 $S_\mathcal{P}$, define the* quotient LPTS*, denoted $\mathcal{P}/\Pi$, as the LPTS $\langle G, g^0, \alpha, \tau\rangle$ where $g^0 \in G$ is such that $[s^0_P](g) = \delta_{g^0}$ for every $P \in \mathcal{P}$ and $g \in G$, $\alpha = \bigcup_{P \in \mathcal{P}} \alpha_P$ and $(g, a, \mu) \in \tau$ iff there exists $(s, a, \mu_p) \in \tau_P$, for some $P \in \mathcal{P}$ such that $s \in g$ and for every $g' \in G$,*

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

We denote this relation between μ and μ_p by $\mu = \text{lift}(\mu_p, g)$.

Thus, $(g, a, \mu) \in \tau$ iff there is a state $s \in g$ with $s \xrightarrow{a} \mu_p$ and μ is obtained by *lifting* μ_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. *$\mathcal{P}/\Pi$ is a well-defined LPTS.*

We have the following lemma analogous to classical partitions.

Lemma 9. *$\mathcal{P}/\Pi$ is consistent with $\mathcal{P}$ for all Π.*

Proof Sketch: One can show that $\{(s, g) | g \in G, s \in S_P \cap g\}$ is a strong simulation between P and $\mathcal{P}/\Pi$ for $P \in \mathcal{P}$. ■

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 $\mathcal{P}$, then there is a Π of size at most k with $\mathcal{P}/\Pi \preceq L$.*

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 $s_P^0 R_P s_L^0$. Let $R = \bigcup_{P \in \mathcal{P}} R_P$. Now, construct a stochastic partition with at most $|S_L|$ many groups following the intuitive explanation we gave when motivating stochastic partitions. For distributions $\mu_p \in \mathit{Dist}(S_\mathcal{P})$ and $\mu_l \in \mathit{Dist}(S_L)$, the stochastic groupings of a state $s \in \mathit{Supp}(\mu_p)$ is obtained by using a weight function showing $\mu_p \sqsubseteq_R \mu_l$. 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 μ_p and μ_l. Moreover, μ_l and this grouping depend on the group of $\mathit{par}(s)$. Once such a stochastic partition Π is built, we can show that $\{(g, s_l) | g$ is the group corresponding to $s_l\}$ is a strong simulation between $\mathcal{P}/\Pi$ and L. ■

Our main result follows as an immediate corollary, using Lemmas 3 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 $S_\mathcal{P}$ of the least size. For example, H_λ, $\lambda \in (0, 1)$, in Figure 6 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 λ 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 $s \in S_\mathcal{P}$, $1 \leq i, j \leq k$. For every i and $s \in S_\mathcal{P}$, add the constraint $\left(\sum_{1 \leq j \leq k} v_{[s](i),j} = 1\right) \vee \left(\sum_{1 \leq j \leq k} v_{[s](i),j} = 0\right)$ 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 $v_{[s_P^0](i),1} = 1$ for every i and $P \in \mathcal{P}$, making g_1 the start state of the quotient, and adding

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

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 σ_{t_n,i,t_p} (t_n, i and t_p are as before) which will now be

$$d_{\mu_n,\mu_p,i} \wedge \sum_{1 \leq j \leq k} v_{[s_p](j),i} > 0.$$

where $d_{\mu_n,\mu_p,i}$ denotes $\mu_n \sqsubseteq_R \mathit{lift}(\mu_p, g_i)$. Thus, we will check if there is a group of $\mathit{par}(s_p)$ (summation over $1 \leq j \leq k$) for which $s_p \in g_i$ and $\mu_n \sqsubseteq_R \mathit{lift}(\mu_p, g_i)$. For a j, $\mathit{lift}(\mu_p, g_i)(g_j)$ is encoded as $\sum_{s \in \mathit{Supp}(\mu_p)} v_{[s](i),j} \cdot \mu_p(s)$. 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 $\mathcal{P}$ 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 $L \simeq U$. 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 $H \simeq U$ holds and otherwise, return a counterexample. If it is a counterexample to $H \preceq U$ ($U \preceq H$), it is called a *negative* (*positive*) counterexample. Following Section II, 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_λ, which first performs an action a and goes to a state with (unknown) probability λ 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 λ as necessary to keep generating counterexamples. After choosing an initial value of λ, 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 λ 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 teacher to have an *execution mapping* to U (see Section II). 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 II. 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 4 no longer works because an update to λ 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 1. 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.

Algorithm 1 Active Learning Loop.

```
1: P = N = ∅
2: H ← single state LPTS with no transitions
3: repeat
4:     conjecture H to the teacher
5:     update P and N from returned counterexamples, or exit
6:     obtain a least sized consistent (stochastic) partition Π
7:     H ← P/Π
8: until false
```

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 1 terminates under Condition 1 on the teacher and using partitions at line 6 with the number of states of each intermediate hypothesis H 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 1 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 1 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 $N_c^{\beta,\gamma}$ in the figure, as necessary. One can show that there are always suitable values of β and γ whenever $N_c^{\beta,\gamma}$ needs to be returned and the learner always conjectures a two state LPTS. In fact, H_λ 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 [25] 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).

$$\frac{L_1 \parallel A \preceq P \qquad L_2 \preceq A}{L_1 \parallel L_2 \preceq P}$$

Several other assume-guarantee rules have been proposed, some of them involving symmetric [26] or circular reasoning [1], [26], [20]. 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 [26], [9], [15].

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 [28])**.** *The parallel composition of L_1 and L_2, denoted $L_1 \parallel L_2$, is defined as the LPTS $\langle S_1 \times S_2, (s_1^0, s_2^0), \alpha_1 \cup \alpha_2, \tau \rangle$ where $(s_1, s_2) \xrightarrow{a} \mu$ iff*

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

Here $\nu_1 \otimes \nu_2 \in \text{Dist}(S_1 \times S_2)$, such that $\nu_1 \otimes \nu_2 : (s_1, s_2) \mapsto \nu_1(s_1) \cdot \nu_2(s_2)$, for $\nu_1 \in \text{Dist}(S_1), \nu_2 \in \text{Dist}(S_2)$.

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 [19]. 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 [15] or algorithms which may not terminate [14].

Motivated by the success of existing applications of active learning to assume-guarantee reasoning [26], [9], [10], 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 *true*. If one of the premises fails, the teacher generates counterexamples with an *execution mapping* (Section II). 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 [19].

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 III). 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 [26]. 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 [19].

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 11. 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 $L_1 \parallel L_2 \preceq P$ directly is $O(poly(|L_1| \cdot |L_2|, |P|))$, where $|L|$ denotes $\max(|S_L|, |\tau_L|)$.

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^b-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 $\sum_{k=1}^{m} 2^{dk^{b+1}} = O(m2^{dm^{b+1}})$, 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 $O(poly(|A|, |\mathcal{N}|, l))$, 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 $O(poly(|A|, |\mathcal{N}|, l) \cdot m2^{dm^{b+1}})$. Furthermore, the complexity of checking the two premises of ASYM is $O(poly(|L_1| \cdot |A|, |P|) + poly(|L_2|, |P|))$ 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 LPTSes 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 LPTSes.

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. This research was sponsored by DARPA META II, GSRC, NSF, SRC, GM, ONR under contracts FA8650-10C-7079, 1041377 (Princeton University), CNS0926181/CNS0931985, 2005TJ1366, GMCMU-CRLNV301, N000141010188, respectively, and the CMU-Portugal Program.

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*, vol. 44(1-2), pp. 185–197, July 2001.

[8] R. Chadha 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 *ICGI*, 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. Garcia 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 *QEST*. 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 *SPIRE*. 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. Pnueli, "In Transition from Global to Modular Temporal Reasoning about Programs," in *LMCS*, ser. NATO ASI, vol. 13. Springer-Verlag, 1985, 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," *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.

[29] W.-G. Tzeng, "Learning Probabilistic Automata and Markov Chains via Queries," *Machine Learning*, vol. 8(2), pp. 151–166, March 1992.

[30] L. Zhang, "Decision Algorithms for Probabilistic Simulations," Ph.D. dissertation, Universität des Saarlandes, 2008.