Leveraging Weighted Automata in Compositional Reasoning about Concurrent Probabilistic Systems

Fei He[1,2,3] Xiaowei Gao[1,2,3] Bow-Yaw Wang[4] Lijun Zhang[5]

[1] Tsinghua National Laboratory for Information Science and Technology (TNList)
[2] School of Software, Tsinghua University
[3] Key Laboratory for Information System Security, Ministry of Education, China
[4] Academia Sinica, Taiwan
[5] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
hefei@tsinghua.edu.cn, syczgxw@163.com, bywang@iis.sinica.edu.tw, zhanglj@ios.ac.cn

Abstract

We propose the first sound and complete learning-based compositional verification technique for probabilistic safety properties on concurrent systems where each component is an Markov decision process. Different from previous works, weighted assumptions are introduced to attain completeness of our framework. Since weighted assumptions can be implicitly represented by multi-terminal binary decision diagrams (MTBDD's), we give an L^*-based learning algorithm for MTBDD's to infer weighted assumptions. Experimental results suggest promising outlooks for our compositional technique.

Categories and Subject Descriptors D.2.4 [*Software/Program Verification*]: Model Checking

General Terms Theory, Verification

Keywords Compositional verification; probabilistic model checking; algorithmic learning

1. Introduction

Probabilistic programs are widely deployed in various systems. For problems requiring substantial computation resources, their solutions can be too costly for practice purposes. For many such problems, probabilistic algorithms may attain better expected worst-case running time than the worst-case running time of any classical algorithm [36]. Probabilistic methods hence become a viable technique to solve hard problems in practice. Indeed, the IEEE 802.11 standard has employed probabilistic methods to avoid the transmission collision in wireless networks [2].

Similar to classical systems, probabilistic systems are not always free of errors. In order to ensure their correctness, verification techniques have been developed to analyze probabilistic systems. Just like classical systems, a probabilistic system may consist of several concurrent components. The number of system states also increases exponentially in the number of concurrent components. Addressing the state explosion problem is crucial to probabilistic verification techniques.

For classical systems, compositional verification aims to mitigate the state explosion problem by divide and conquer. Suppose a classical system $M_0 \| M_1$ composed of two concurrent components M_0, M_1, and P an intended property about the system. Consider the assume-guarantee reasoning proof rule for classical systems [13]:

$$\frac{M_0 \preceq A \qquad A \| M_1 \models P}{M_0 \| M_1 \models P} \tag{1}$$

The notation $M_0 \preceq A$ means that A simulates all behaviors of M_0. Informally, the rule says that to show the composed system satisfying P, it suffices to find a classical assumption A such that A simulates M_0, and A composed with M_1 satisfies P as well.

A useful assumption needs to be small (at least smaller than M_0) and able to establish the intended property. Finding useful assumptions in assume-guarantee reasoning appears to require ingenuity. Although heuristics have been proposed to construct such assumptions automatically, they are not always applicable. Oftentimes, verifiers have to provide assumptions manually. Such laborious tasks are very time consuming and can be extremely difficult to carry out on large systems.

Interestingly, the problem of finding useful classical assumptions can be solved by active machine learning [13]. In active machine learning [3], a learning algorithm infers a representation of an unknown target by making queries to a teacher. The learning-based framework thus devises a mechanical teacher to answer such queries. Together with a learning algorithm, the framework is able to find assumptions automatically. For classical systems, the L^* learning algorithm for regular languages [3] suffices to infer classical finite automata as classical assumptions [13]. Other techniques have also been developed to find useful assumptions for compositional verification of classical systems [11, 20, 22, 25].

From the classical learning-based framework, one gathers that two ingredients are essential to finding probabilistic assumptions. First, a sound and invertible assume-guarantee reasoning proof rule for probabilistic systems is needed. A sound proof rule allows us to analyze compositionally by finding probabilistic assumptions. An invertible proof rule additionally guarantees the existence of such probabilistic assumptions when probabilistic systems satisfy intended properties. Second, a learning algorithm for probabilistic assumptions is also needed. With a carefully designed mechanical teacher, probabilistic assumptions can then be inferred by the learning-based framework for probabilistic systems.

POPL '15, January 15–17, 2015, Mumbai, India.

ACM 978-1-4503-3300-9/15/01…$15.00.
http://dx.doi.org/10.1145/2676726.2676998

Finding a sound and invertible assume-guarantee reasoning proof rule does not appear to be a problem. Indeed, the classical proof rule (1) can be extended to probabilistic systems via probabilistic simulation [39]. Learning probabilistic assumptions however is more difficult. To the best of our knowledge, an active learning algorithm for probabilistic systems is yet to be found. In fact, it is undecidable to infer labeled probabilistic transition systems under a version of Angluin's learning model [30]. Learning algorithms for general probabilistic systems may not exist after all.

Given the absence of learning algorithms for probabilistic systems, some authors propose restricted proof rules with only classical assumptions [16, 32]. Since classical assumptions can be represented by classical finite automata, the L^* algorithm is employed to infer such assumptions in restricted probabilistic assume-guarantee reasoning proof rules. Yet classical assumptions are incapable of expressing general probabilistic behaviors. Such restricted proof rules are not invertible. Subsequently, existing probabilistic assume-guarantee reasoning frameworks are sound but incomplete.

We propose a sound and complete assume-guarantee reasoning framework for verifying probabilistic safety properties on Markov decision processes (MDP's). Let M_0 and M_1 be MDP's, and $P_{\leq p}[\psi]$ a probabilistic safety property. Our most ingenious idea is to consider *weighted* assumptions in our new assume-guarantee reasoning proof rule:

$$\frac{M_0 \preceq_e A \qquad A \| M_1 \models P_{\leq p}[\psi]}{M_0 \| M_1 \models P_{\leq p}[\psi]} \tag{2}$$

where A is a weighted automaton. Intuitively, $M_0 \preceq_e A$ means that every transition of A has a weight not less than the probability of the corresponding transition in M_0. Compared to the proof rules in [16, 32], ours relaxes but does not restrict the expressive power of assumptions. More precisely, we consider $0/1$-weighted automata whose weights are between 0 and 1 inclusively as weighted assumptions. Since transition functions of $0/1$-weighted automata can be probability distributions, the class of $0/1$-weighted automata subsumes MDP's. Our assume-guarantee reasoning proof rule is trivially invertible.

In order to find weighted assumptions in our learning-based framework, we also need a learning algorithm for such assumptions. Although active learning algorithms for probabilistic systems are still unknown, weighted assumptions on the other hand are learnable due to the relaxation on transition functions. Our second innovation is to adopt a well-known representation that enables a simple L^*-based learning algorithm for weighted assumptions. Observe that weighted automata can be implicitly represented by multi-terminal binary decision diagrams (MTBDD's) [6, 15, 18]. We hence develop an L^*-based learning algorithm for MTBDD's and deploy it to infer implicitly represented weighted assumptions. With the two ingredients, a mechanical teacher is designed to guide our learning algorithm to find weighted assumptions for probabilistic safety properties. We successfully develop a sound and complete learning-based assume-guarantee reasoning framework by circumventing the unsolved problem of learning probabilistic systems.

In addition to completeness and learnability, adopting weighted assumptions can also be very efficient. Note that assumptions are not unique oftentimes. If a probabilistic assumption establishes a probabilistic property, a slightly different weighted (but not necessarily probabilistic) assumption most likely will establish the property as well. Since there are more useful weighted assumptions, our new framework can be more effective in finding one of them. Additionally, inferring weighted assumptions implicitly allows us to better integrate the learning-based framework with symbolic probabilistic model checking. Indeed, experimental results from realistic test cases such as IEEE 802.11 and 1394 standards are promising.

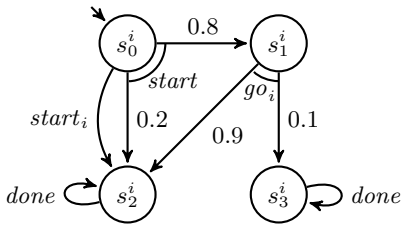


Figure 1: node_i

Compositional verification can alleviate the state explosion problem even for probabilistic programs.

Our technical contributions are summarized as follows.

- We propose the first sound and invertible assume-guarantee reasoning proof rule with weighted assumptions for probabilistic safety properties on MDP's.
- We give an MTBDD learning algorithm under Angluin's active learning model. It uses a polynomial number of queries in the sizes of target MTBDD's and variable sets.
- With our new proof rule and learning algorithm, we give the first sound and complete learning-based assume-guarantee reasoning framework for probabilistic safety properties on MDP's.
- We compare our new technique with the monolithic probabilistic model checker PRISM [38]. Experimental results suggest promising outlooks for our compositional technique.

This paper is organized as follows. In Section 2, we illustrate our learning-based compositional verification technique by a small example. In Section 3, backgrounds of probabilistic systems and probabilistic model checking are provided. Section 4 presents our sound and invertible assume-guarantee reasoning proof rule. The MTBDD learning algorithm is described in Section 5. Our learning-based assume-guarantee reasoning framework is given in Section 6. Section 7 reports the experimental results on parameterized test cases. Finally, Section 9 concludes this paper.

2. A Motivating Example

Consider the probabilistic system $\mathsf{node}_1 \| \mathsf{node}_2$ composed of two MDP's node_i $(i = 1, 2)$ in Figure 1. The process node_i has four states: the initial state (s_0^i), the ready state (s_1^i), the succeeded state (s_2^i), and the failed state (s_3^i). Initially, both node_1 and node_2 begin at their respective initial states s_0^1 and s_0^2. The system may start up all nodes (by the $start$ action), or choose one node to start (by either the $start_1$ or $start_2$ action). The two processes node_1 and node_2 synchronize on shared actions. When the system starts up all nodes by the $start$ action, node_1 transits to its ready state s_1^1 with probability 0.8, or to its succeeded state s_2^1 with the probability 0.2. Simultaneously, node_2 transits to its ready and succeeded states s_1^2 and s_2^2 with probabilities 0.8 and 0.2 respectively. Note that the sum of probabilities on each action is 1. Each action hence gives a probabilistic distribution over states. For non-shared actions, only the acting process moves; other processes stay. Hence node_1 transits to its succeeded state s_2^1 while node_2 remains in its initial state s_0^2 when the system chooses to start up node_1 with the action $start_1$. Similarly, when the process node_i is at its ready state s_1^i, it transits to its succeeded state s_2^i with probability 0.9, or to its failed state s_3^i with probability 0.1 on the action go_i. Observe that the probability of a transition is not shown when it is 1. Hence node_i transits from s_0^i to s_2^i with probability 1 on the action $start_i$.

In the system $\mathsf{node}_1 \| \mathsf{node}_2$, the system state $s_3^1 s_3^2$ is the failed state. The system is designed so that the probability of reaching the failed state is no more than 0.01. Formally, the intended property is

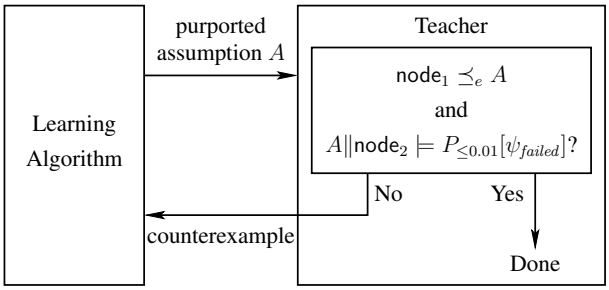


Figure 2: Overview

specified by the probabilistic computation tree logic formula

$$P_{\leq 0.01}[\psi_{failed}]$$

where ψ_{failed} stands for $F\ \langle s_3^1 s_3^2 \rangle$ and F is the "in the future" temporal operator. We would like to check whether the system satisfies the probabilistic property by compositional verification.

2.1 Compositional Reasoning

With the proof rule (2), to show $\mathsf{node}_1 \| \mathsf{node}_2 \models P_{\leq 0.01}[\psi_{failed}]$, it suffices to find a weighted assumption A that

- $\mathsf{node}_1 \preceq_e A$, equivalently, A performs every transition of node_1 with no less probability; and
- $A \| \mathsf{node}_2 \models P_{\leq 0.01}[\psi_{failed}]$, equivalently, the system $A \| \mathsf{node}_2$ satisfies the probabilistic property.

Clearly, one could choose A to be node_1 if the system satisfies the intended probabilistic property. But then the premise $A \| \mathsf{node}_2 \models P_{\leq 0.01}[\psi_{failed}]$ is precisely the conclusion $\mathsf{node}_1 \| \mathsf{node}_2 \models P_{\leq 0.01}[\psi_{failed}]$. Verifiers would not benefit from compositional verification by choosing node_1 as a weighted assumption.

2.2 Overview

We follow the learning-based framework to infer a weighted assumption satisfying the two conditions in the last subsection [11, 13, 25, 32]. In the framework, a learning algorithm is deployed to infer weighted assumptions with the help of a mechanical teacher. The learning algorithm presents purported assumptions to the teacher. The teacher checks if a purported weighted assumption fulfills the premises in the assume-guarantee reasoning proof rule (2). If not, the mechanical teacher will help the learning algorithm refine purported assumptions by counterexamples.

Figure 2 gives an overview of the learning-based framework. On a purported weighted assumption A, the teacher checks $\mathsf{node}_1 \preceq_e A$ and invokes a model checker to verify $A \| \mathsf{node}_2 \models P_{\leq 0.01}[\psi_{failed}]$. If both premises are fulfilled, we are done. Otherwise, the teacher provides a counterexample to the learning algorithm. The learning algorithm then modifies the purported weighted assumption A accordingly. We illustrate the framework with concrete examples.

2.3 A Purported Assumption

Consider a purported weighted assumption A in Figure 3. On the actions $start$, $start_1$, go_1, and $done$, the assumption A can transit from a state to any state. Similar to MDP's, the weight of a transition is not shown when it is 1. For instance, A transits from the state s_1^1 to the state s_j^1 on the action go_1 with weight 1 for every $0 \leq j \leq 3$. In comparison, the process node_1 moves from the state s_1^1 to the states $s_0^1, s_1^1, s_2^1, s_3^1$ on the action go_1 with probabilities $0, 0, 0.9, 0.1$ respectively (Figure 1). Observe that A is not an MDP since the sum of weights from the state s_1^1 on the action go_1 is 4.

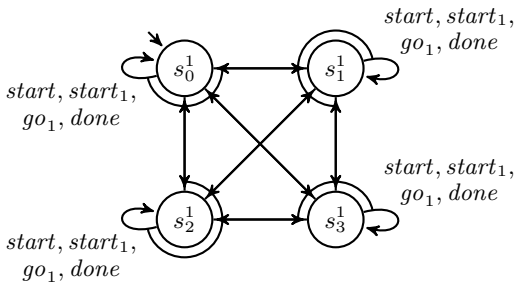


Figure 3: Weighted Assumption A

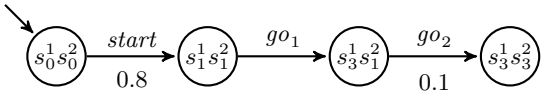


Figure 4: Witness to $A \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$

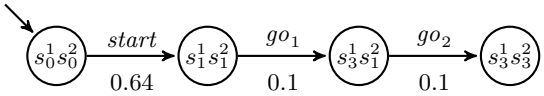


Figure 5: Corresponding Path in $\mathsf{node}_1 \| \mathsf{node}_2$

On receiving the weighted assumption A, the mechanical teacher decides whether the assumption A fulfills both premises in our probabilistic compositional verification proof rule. It first checks if the assumption A performs every transition of node_1 with a weight not less than the probability in node_1. This is clearly the case. Consider, for instance, the transitions from s_1^1 to $s_0^1, s_1^1, s_2^1, s_3^1$ on the action go_1. The weights associated with these transitions of A are all equal to 1. They are not less than the probabilities $0, 0, 0.9, 0.1$ associated with the corresponding transitions of node_1 respectively. The premise $\mathsf{node}_1 \preceq_e A$ is fulfilled. The mechanical teacher then checks the other premise by model checking.

2.4 Model Checking

Technically, a probabilistic model checker does not take weighted assumptions as inputs. Since A is a weighted assumption, $A \| \mathsf{node}_2$ need not be an MDP. A probabilistic model checker can not verify whether $A \| \mathsf{node}_2 \models P_{\leq 0.01}[\psi_{failed}]$ directly. We need to lift the probabilistic model checking algorithm to weighted assumptions.

After model checking, we find that the property $P_{\leq 0.01}[\psi_{failed}]$ does not hold on $A \| \mathsf{node}_2$. A witness to $A \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$ is shown in Figure 4. The witness has only one path from the initial state $s_0^1 s_0^2$ to the failed state $s_3^1 s_3^2$. Its weight is $0.8 \times 1 \times 0.1 = 0.08 > 0.01$. $P_{\leq 0.01}[\psi_{failed}]$ is not satisfied on $A \| \mathsf{node}_2$.

2.5 Witness Checking

Since $A \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$, the mechanical teacher concludes that the weighted assumption A does not establish the intended probabilistic property. On the other hand, the mechanical teacher cannot conclude that the system $\mathsf{node}_1 \| \mathsf{node}_2$ does not satisfy the property either. Since A has larger weights than node_1, a weighted witness to $A \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$ is not necessarily a witness to $\mathsf{node}_1 \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$. Before revising the weighted assumption A, the mechanical teacher checks if the witness to $A \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$ is spurious or not.

Recall that the weighted assumption A contains all transitions in node_1. The witness to $A \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$ therefore corresponds to a path in $\mathsf{node}_1 \| \mathsf{node}_2$ (Figure 5). Also recall that the weight associated with a transition in the weighted assumption

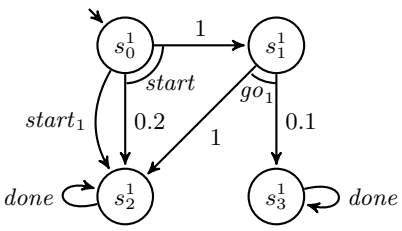


Figure 6: Weighted Assumption A'

A is not less than the probability of the corresponding transition in node_1. The probability of the corresponding path in $\mathsf{node}_1 \| \mathsf{node}_2$ can be much smaller than the weight of the witness to $A \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$. Indeed, the corresponding path in $\mathsf{node}_1 \| \mathsf{node}_2$ has probability $0.64 \times 0.1 \times 0.1 = 0.0064 \leq 0.01$. It does satisfy the intended probabilistic property. The witness to $A \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$ is hence spurious. The mechanical teacher then should help the learning algorithm revising the weighted assumption by sending a counterexample.

2.6 Selecting Counterexamples

In order to remove the spurious witness in Figure 4 from the weighted assumption A, the mechanical teacher selects a transition in the weighted assumption A which contributes most to the spurious witness. In Figure 4, the transitions $s_0^1 \xrightarrow{start} s_1^1$ and $s_1^1 \xrightarrow{go_1} s_3^1$ in the weighted assumption A contribute to the spurious witness. The mechanical teacher can send either of the transitions as a counterexample to the learning algorithm. Here, let us say the mechanical teacher sends the transition $s_1^1 \xrightarrow{go_1} s_3^1$ as the counterexample. The learning algorithm will then update the weight of the selected transition in revised weighted assumptions.

2.7 Learning Assumption

After receiving a counterexample, the learning algorithm will purport another weighted assumption. Suppose the learning algorithm purports the weighted assumption A' (Figure 6). For any transition, its weight in A' is no less than the probability of the corresponding transition in node_1. For example, the weighted assumption A' transits from the state s_1^1 to the states $s_0^1, s_1^1, s_2^1, s_3^1$ with weights $0, 0, 1, 0.1$ respectively on the action go_1. The corresponding transitions in node_1 have probabilities $0, 0, 0.9, 0.1$ respectively. We have $\mathsf{node}_1 \preceq_e A'$. $A' \| \mathsf{node}_2 \models P_{\leq 0.01}[\psi_{failed}]$ moreover holds by model checking. According to our compositional verification proof rule, the mechanical teacher concludes that the system $\mathsf{node}_1 \| \mathsf{node}_2$ satisfies the intended probabilistic property.

Note again that A' is a not a probabilistic assumption. Although A' and node_1 have the same number of states in the explicit representation, their implicit MTBDD representations are different. A' has 26 nodes and 4 terminals; node_1 has 27 nodes with 6 terminals in the implicit representation. Compositional verification replaces the component node_1 with a slightly smaller weighted assumption A'. In fact, node_1 is the only probabilistic assumption that can establish the probabilistic property. If only probabilistic assumptions were considered, assume-guarantee reasoning would not be effective in this example. Adopting weighted assumptions gives our framework more useful assumptions in compositional verification.

3. Preliminaries

3.1 Weighted Automata and Markov Decision Processes

Given a finite set S, a *weighted function* on S is a mapping $\delta : S \to \mathbb{Q}$. A weighted function on S is denoted as a vector of length $|S|$. A *probability distribution* on S is a function $\delta^D : S \to [0,1] \cap \mathbb{Q}$ that $\sum_{s \in S} \delta^D(s) = 1$. A *point distribution* ε_s on $s \in S$ is a probability distribution where $\varepsilon_s(t) = 1$ if $t = s$ and $\varepsilon_s(t) = 0$ otherwise. Denote the set of weighted functions and probability distributions on S by $\Delta(S)$ and $\Delta^D(S)$ respectively. Clearly, $\Delta^D(S) \subseteq \Delta(S)$.

Definition 1 *A* weighted automaton *(WA) is a 4-tuple $M = (S, \bar{s}, Act, T)$ where S is a finite set of* states*, $\bar{s} \in S$ is an* initial state*, Act is a finite alphabet of* actions*, and $T : S \times Act \to \Delta(S)$ is a* weighted transition function*.*

A *finite path* π in M is a non-empty finite sequence $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_{n-1}} s_n$ where $s_0 = \bar{s}$, $\alpha_i \in Act$, and $s_i \xrightarrow{\alpha_i} s_{i+1}$ is a *transition* with $T(s_i, \alpha_i)(s_{i+1}) \neq 0$ for all $0 \leq i < n$. We denote by $\pi[i] = s_i$ the $(i+1)$th state, and $|\pi| = n$ its *length*. The *weight* $Wt(\pi)$ of a finite path π is $T(s_0, \alpha_0)(s_1) \times T(s_1, \alpha_1)(s_2) \times \cdots \times T(s_{n-1}, \alpha_{n-1})(s_n)$. Denote $Path_M$ the set of all finite paths in M. Let $\Pi \subseteq Path_M$ be a set of finite paths. Π is *prefix containment free* if for every $\pi, \pi' \in \Pi$, π is not a proper prefix of π'. When Π is prefix containment free, the *weight* $Wt(\Pi)$ of Π is $\sum_{\pi \in \Pi} Wt(\pi)$.

Definition 2 *A* $0/1$-weighted automaton ($0/1$-WA) *$M = (S, \bar{s}, Act, T)$ is a WA where $0 \leq T(s, \alpha)(t) \leq 1$ for every $s, t \in S$ and $\alpha \in Act$.*

A WA is nondeterministic. There may be multiple transitions between two states on different actions. Adversaries are used to resolve nondeterministic choices in WA's [5]. Let S^+ denote a non-empty sequence of states in S, and $Act(s)$ the set $\{\alpha \in Act : T(s, \alpha)(t) > 0 \text{ for some } t\}$. An (deterministic) *adversary* is a function $\sigma : S^+ \to Act$ such that $\sigma(s_0 s_1 \dots s_n) \in Act(s_n)$. More general notion of adversaries involving randomizations exists, but deterministic ones are sufficient for our problem. A WA M under an adversary σ is therefore deterministic. Let Adv_M denote the set of adversaries of M. We write M^σ for the WA whose transitions are determinized by the adversary $\sigma \in Adv_M$.

Definition 3 *A* Markov decision process *(MDP) $M = (S, \bar{s}, Act, T)$ is a $0/1$-WA where $T(s, \alpha) \in \Delta^D(S)$ or $T(s, \alpha)$ is the constant zero weighted function for every $s \in S$ and $\alpha \in Act$.*

Since the weighted functions returned by weighted transition functions of MDP's are probability distributions, the weight associated with each transition in MDP's is referred to as probability.

Let S_i be a finite set and $\delta_i \in \Delta(S_i)$ for $i = 0, 1$. Define $(\delta_0 \otimes \delta_1)(s_0, s_1) = \delta_0(s_0) \times \delta_1(s_1)$. Observe that $\delta_0 \otimes \delta_1 \in \Delta(S_0 \times S_1)$; and if $0 \leq \delta_0(s_0) \leq 1$ and $0 \leq \delta_1(s_1) \leq 1$, then $0 \leq (\delta_0 \otimes \delta_1)(s_0, s_1) \leq 1$.

Definition 4 *Let $M_i = (S_i, \bar{s}_i, Act_i, T_i)$ be WA for $i = 0, 1$. The* parallel composition *of M_0 and M_1 (written $M_0 \| M_1$) is a WA $M_0 \| M_1 = (S_0 \times S_1, (\bar{s}_0, \bar{s}_1), Act_0 \cup Act_1, T)$ where*

$$T((s_0, s_1), \alpha) = \begin{cases} T_0(s_0, \alpha) \otimes \varepsilon_{s_1} & \text{if } \alpha \notin Act_1 \\ \varepsilon_{s_0} \otimes T_1(s_1, \alpha) & \text{if } \alpha \notin Act_0 \\ T_0(s_0, \alpha) \otimes T_1(s_1, \alpha) & \text{if } \alpha \in Act_0 \cap Act_1 \end{cases}$$

Observe that parallel composition of two $0/1$-WAs yields a $0/1$-WA, and parallel composition of two MDP's yields an MDP.

Example 1 *The process node_1 in Figure 1 is an MDP with $S_{\mathsf{node}_1} = \{s_0^1, s_1^1, s_2^1, s_3^1\}$, $\bar{s}_{\mathsf{node}_1} = s_0^1$, $Act_{\mathsf{node}_1} = \{start, start_1, go_1, done\}$, and $T_{\mathsf{node}_1}(s, \alpha) = \langle 0, 0.8, 0.2, 0 \rangle$ if $s = s_0^1, \alpha = start$ (others are defined similarly). On the other hand, A in Figure 3 is a $0/1$-WA since its weighted transition function does not return probability distributions.*

3.2 Probabilistic Model Checking for MDP's

Fix a finite set AP of *atomic propositions*. We focus on *probabilistic safety properties* specified by *Probabilistic Computation Tree Logic (PCTL)* [8, 24] in the form of $P_{\leq p}[\psi]$ with $p \in [0,1]$ and

$$\phi ::= true \mid a \mid \phi \wedge \phi \mid \neg\phi$$
$$\psi ::= \phi\, U \phi$$

where a is an atomic proposition, ϕ a *state formula*, ψ a *path formula*, and U the "until" temporal operator. For example, "the probability of an error occurrence is at most 0.01" is specified as $P_{\leq 0.01}[true\, U \phi_{err}]$ where ϕ_{err} is a state formula indicating the occurrence of an error.

PCTL and the safety fragment in general allow nested probabilistic operators [27]. In this paper, we consider a fragment, known as *conditional reachability probability*, and leave the extension to general PCTL safety property as our future work.

Given $M = (S, \bar{s}, Act, T)$ and $P_{\leq p}[\psi]$ with $\psi = \phi_1 U \phi_2$. We define $p^{\sigma}_{M,s}(\psi) = Wt(\Pi)$ where $\Pi = \{\pi \in Path_{M^\sigma} \mid \forall i = 0..|\pi|-1.\ (\pi[i] \models \phi_1 \wedge \neg\phi_2) \wedge (\pi[|\pi|] \models \phi_2)\}$. Observe that Π is prefix containment free, and $prob(\Pi)$ is the probability of reaching ϕ_2 states along ϕ_1 states under the adversary σ.

Let $p^{max}_{M,s}(\psi) = \max_{\sigma \in Adv_M} p^{\sigma}_{M,s}(\psi)$ denote the maximal probability that ψ is satisfied at s over all adversaries. We say that s *satisfies* $P_{\leq p}[\psi]$ (written $M, s \models P_{\leq p}[\psi]$) if $p^{max}_{M,s}(\psi) \leq p$; M *satisfies* $P_{\leq p}[\psi]$ (written $M \models P_{\leq p}[\psi]$) if $M, \bar{s} \models P_{\leq p}[\psi]$. We write $p^{\sigma}_{M,s}(\psi)$ and $p^{max}_{M,s}(\psi)$ as $p^{\sigma}_{s}(\psi)$ and $p^{max}_{s}(\psi)$ respectively if M is clear.

Let $\psi = \phi_1 U \phi_2$. The probability $p^{max}_s(\psi)$ can be approximated by an iterative algorithm [5]. The computation starts from the states satisfying ϕ_2 and iterates backward to compute the maximum probability of reaching these states from the states satisfying ϕ_1. More precisely, define

$$p^{max}_{s,i}(\psi) = \begin{cases} 1 & \text{if } s \models \phi_2 \\ 0 & \text{if } s \not\models \phi_2 \wedge s \not\models \phi_1 \\ 0 & \text{if } s \not\models \phi_2 \wedge s \models \phi_1 \wedge i = 0 \\ \max_{\alpha} \left\{ \sum_{t \in S} T(s,\alpha)(t) \times p^{max}_{t,i-1}(\psi) \right\} & \text{otherwise} \end{cases}$$

Then $p^{max}_s(\psi) = \lim_{i \to \infty} p^{max}_{s,i}(\psi)$. The computation iterates until $p^{max}_{s,i}(\psi)$ converges.

A *weighted witness* [23, 41, 42] to $M \not\models P_{\leq p}[\psi]$ is a pair (σ, c) where $\sigma \in Adv_M$ is an adversary with $p^{\sigma}_{\bar{s}}(\psi) > p$, and c is a set of finite paths in M^σ such that (1) for all $\pi \in c$, $\pi \models \psi$; (2) for all proper prefix π' of π, $\pi' \not\models \psi$; and (3) $Wt(c) > p$. Observe that the set c is prefix containment free. Hence $Wt(c)$ is well-defined. We obtain the (σ, c)-*fragment* of M (written $M^{\sigma,c}$) by removing all transitions not appearing in any path of c from M^σ.

Example 2 *Consider the weighted witness (σ, c) shown in Figure 4 where $\sigma(\langle s^1_0 s^2_0\rangle) = start$, $\sigma(\langle s^1_0 s^2_0\rangle\langle s^1_1 s^2_1\rangle) = go_1$, $\sigma(\langle s^1_0 s^2_0\rangle\langle s^1_1 s^2_1\rangle \langle s^1_3 s^2_1 1\rangle) = go_2$, and $c = \{\langle s^1_0 s^2_0\rangle \xrightarrow{start} \langle s^1_1 s^2_1\rangle \xrightarrow{go_1} \langle s^1_3 s^2_1\rangle \xrightarrow{go_2} \langle s^1_3 s^2_3\rangle\}$. The weight of c is $0.8 \times 1 \times 0.1 = 0.08$.*

3.3 Model Checking for $0/1$-WA's

Consider our assume-guarantee proof rule (2), where A is a $0/1$-weighted assumption, the parallel composition $A \| M_1$ usually yields a $0/1$-WA, but not an MDP. The probabilistic model checking algorithm in preceding section needs be adapted to $0/1$-WA's.

Given a state s of a $0/1$-WA M and a probabilistic safety property $P_{\leq p}[\psi]$, we say s *satisfies* $P_{\leq p}[\psi]$ (written $M, s \models P_{\leq p}[\psi]$) if the weight $w^{max}_s(\psi) \leq p$ where $w^{max}_s(\psi)$ is defined similarly as for MDP's. Here $w^{max}_s(\psi)$ is referred to as *weights* rather than *probability*. Note we again omit the subscript M when it is clear. We say that M *satisfies* $P_{\leq p}[\psi]$ (written $M \models P_{\leq p}[\psi]$) if $M, \bar{s} \models P_{\leq p}[\psi]$. An iterative algorithm similar to the one for MDP's is used to compute $w^{max}_s(\psi)$ for $0/1$-WAs. A notable difference is that the iterative computation may not converge in a $0/1$-WA. To avoid divergent computation, define

$$\underline{w}^{max}_{s,i}(\psi) = \min\left\{1, \max_{\alpha}\left\{\sum_{t \in S} T(s,\alpha)(t) \times \underline{w}^{max}_{t,i-1}(\psi)\right\}\right\}$$

if $s \not\models \phi_2 \wedge s \models \phi_1 \wedge i > 0$. Effectively, $\underline{w}^{max}_{s,i}(\psi)$ truncates the value of $w^{max}_{s,i}(\psi)$ when the latter exceeds 1. Let $\underline{w}^{max}_{s}(\psi) = \lim_{i \to \infty} \underline{w}^{max}_{s,i}(\psi)$. We have $\underline{w}^{max}_{s}(\psi) \leq 1$, and $\underline{w}^{max}_{s}(\psi) = w^{max}_{s}(\psi)$ when $w^{max}_s \leq 1$.

Note that the properties we are interested in are of the form $P_{\leq p}[\psi]$ with $p \in [0,1]$. When $p = 1$, such properties are trivially satisfied. For $0 \leq p < 1$, we have $p < w^{max}_s(\psi)$ if $p < \underline{w}^{max}_s(\psi)$.[1]

4. An Assume-Guarantee Reasoning Proof Rule

Assume-guarantee reasoning proof rules for probabilistic systems are proposed in [16, 32]. Those proof rules replace a probabilistic component in a composition with a classical assumption. Since classical assumptions can not characterize all probabilistic behaviors of the replaced component, such rules are not invertible. We propose an assume-guarantee reasoning proof rule that replaces a probabilistic component with a weighted automaton. We begin with the weighted extension of the classical simulation relation.

Definition 5 *Let $M = (S, \bar{s}, Act, T)$ and $M' = (S', \bar{s}', Act', T')$ be WA's, we say M is* embedded *in M' (written $M \preceq_e M'$) if $S = S'$, $\bar{s} = \bar{s}'$, $Act = Act'$, and $T(s,\alpha)(t) \leq T'(s,\alpha)(t)$ for every $s, t \in S$ and $\alpha \in Act$.*

Lemma 1 *Let M, M', N be $0/1$-WA's, and $P_{\leq p}[\psi]$ a probabilistic safety property.*

1. *$M \preceq_e M'$ implies $M \| N \preceq_e M' \| N$;*
2. *$M \preceq_e M'$ and $M' \models P_{\leq p}[\psi]$ imply $M \models P_{\leq p}[\psi]$.*

Proof: *Let $M = (S, \bar{s}, Act, T)$, $M' = (S', \bar{s}', Act', T')$, and $N = (S_N, \bar{s}_N, Act_N, T_N)$ be WA's. By $M \preceq_e M'$, we have $S = S'$, $\bar{s} = \bar{s}'$, and $Act = Act'$. Hence $M \| N$ and $M' \| N$ have the same state space, initial state, and alphabet. Since $T(s,\alpha)(t) \leq T'(s,\alpha)(t)$ and $T_N(p,\alpha)(q) \geq 0$, $T(s,\alpha)(t) \times T_N(p,\alpha)(q) \leq T(s,\alpha)(t) \times T_N(p,\alpha)(q)$ for every $s, t \in S$, $p, q \in S_N$, and $\alpha \in Act \cap Act_N$. Hence $T_{M\|N}((s,p),\beta)(t,q) \leq T_{M'\|N}((s,p),\beta)(t,q)$ for every $\beta \in Act \cup Act_N$ by Definition 4.*

Since $T(s,\alpha)(t) \leq T'(s,\alpha)(t)$, $w^{max}_{M,\bar{s}}(\psi) \leq w^{max}_{M',\bar{s}}(\psi)$. $w^{max}_{M',\bar{s}}(\psi) \leq p$ for $M' \models P_{\leq p}[\psi]$. Thus $w^{max}_{M,\bar{s}}(\psi) \leq w^{max}_{M',\bar{s}}(\psi) \leq p$. $M \models P_{\leq p}[\psi]$ as well. □

Lemma 1 shows that the operator $\preceq_e$ is compositional and preserves probabilistic safety properties. Hence

Theorem 1 *Let $M_i = (S_i, \bar{s}_i, Act_i, T_i)$ be MDP's for $i = 0, 1$ and $P_{\leq p}[\psi]$ a probabilistic safety property, Then the following proof rule is both sound and invertible:*

$$\frac{M_0 \preceq_e A \qquad A \| M_1 \models P_{\leq p}[\psi]}{M_0 \| M_1 \models P_{\leq p}[\psi]}$$

where $A = (S_A, \bar{s}_A, Act_A, T_A)$ is a $0/1$-WA.

Proof: *Soundness of the proof rule follows from Lemma 1. By $M_0 \preceq_e A$, $M_0 \| M_1 \preceq_e A \| M_1$ (Lemma 1(1)). Since $A \| M_1 \models P_{\leq p}[\psi]$, we have $M_0 \| M_1 \models P_{\leq p}[\psi]$ (Lemma 1(2)). The proof rule is also invertible. When the conclusion holds, M_0 itself is a weighted assumption. The two premises are trivially fulfilled.* □

[1] Note here we need $p < 1$ to conclude $p < w^{max}_s(\psi)$.

5. Learning $0/1$-Weighted Automata

We adopt the learning-based framework [13, 16] to generate an assumption A in the assume-guarantee reasoning proof rule (Theorem 1). To apply our new proof rule, a weighted assumption is needed. One could employ learning algorithms that infer explicit quantitative models like multiplicity automata [7]. Those learning algorithms require complex and accurate matrix operations. They hence induce substantial computation and implementation overheads. To avoid such overheads, we adopt a different representation to enable a simple and efficient learning technique. More precisely, we use MTBDD's to represent weighted assumptions implicitly. To infer implicitly represented weighted assumptions, we then develop an MTBDD learning algorithm under Angluin's learning model.

5.1 Multi-Terminal Binary Decision Diagrams

Let $\mathbb{B}$ denote the *Boolean domain* $\{0, 1\}$. Fix a finite ordered set of Boolean variables $\mathbf{x} = \langle x_1, x_2, \ldots, x_n\rangle$. A *valuation* $\nu = \langle v_1, v_2, \ldots, v_n\rangle$ of $\mathbf{x}$ assigns the Boolean value v_i to the Boolean variable x_i. Let μ and ν be valuations of $\mathbf{x}$ and $\mathbf{y}$ respectively with $\mathbf{x} \cap \mathbf{y} = \emptyset$. The *concatenation* of μ and ν is the valuation $\mu\nu$ of $\mathbf{x}\cup\mathbf{y}$ such that $\mu\nu(z) = \mu(z)$ if $z \in \mathbf{x}$ and $\mu\nu(z) = \nu(z)$ if $z \in \mathbf{y}$. For $\mathbf{y} \subseteq \mathbf{x}$, the *restriction* $\nu{\uparrow_{\mathbf{y}}}$ of ν on $\mathbf{y}$ is a valuation of $\mathbf{y}$ that $\nu{\uparrow_{\mathbf{y}}}(y) = \nu(y)$ for $y \in \mathbf{y}$. Let $f(\mathbf{x}) : \mathbb{B}^n \to \mathbb{Q}$ be a function over $\mathbf{x}$. We write $f(\nu)$ for the *function value* of f under the valuation ν. Let $f_1(\mathbf{x})$ and $f_2(\mathbf{x})$ be functions over $\mathbf{x}$, $f_1(\mathbf{x}) \leq f_2(\mathbf{x})$ denotes $f_1(\nu) \leq f_2(\nu)$ for every valuation ν of $\mathbf{x}$.

A *multi-terminal binary decision diagram* (MTBDD) [18] over $\mathbf{x}$ is a rooted, directed, acyclic graph representing a function $f(\mathbf{x}) : \mathbb{B}^n \to \mathbb{Q}$. An MTBDD has two types of nodes. A *non-terminal* node is labeled with a variable x_i; it has two outgoing edges with labels 0 and 1. A *terminal* node is labeled with a rational number. The representation supports binary operations. For instance, the MTBDD of the sum of two functions is computed by traversing the MTBDD's of the two functions.

Given a valuation ν of $\mathbf{x}$, $f(\nu)$ can be obtained by traversing the MTBDD of $f(\mathbf{x})$. Starting from the root, one follows edges by values of the Boolean variables labeling the nodes. When a terminal node is reached, its label is the value $f(\nu)$. Since a function $f(\mathbf{x})$ and its MTBDD are equivalent, $f(\mathbf{x})$ also denotes the MTBDD of the function $f(\mathbf{x})$ by abusing the notation.

It is straightforward to represent a WA by MTBDD's [26]. Let $M = (S, \bar{s}, Act, T)$ be a WA. Without loss of generality, we assume $|S| = 2^n$ and $|Act| = m$. We use $\mathbf{x} = \langle x_1, x_1, \ldots, x_n\rangle$, $\mathbf{x}' = \langle x'_1, x'_2, \ldots, x'_n\rangle$ to encode states and next states in S, and $\mathbf{z} = \langle z_1, z_2, \ldots, z_m\rangle$ to encode actions in Act. Let ν, ν' be valuations of $\mathbf{x}, \mathbf{x}'$ and α a valuation of $\mathbf{z}$. The action valuation α of $\mathbf{x}$ is *valid* if it maps at most one variable to 1 (at most one action can be taken). A valuation ν of $\mathbf{x}$ or $\mathbf{x}'$ encodes a state $\lceil\nu\rceil \in S$. A valid action valuation α encodes an action $\lceil\alpha\rceil \in Act$. Define

$$l_M(\nu) = \begin{cases} 1 & \text{if } \lceil\nu\rceil = \bar{s} \\ 0 & \text{otherwise} \end{cases}$$

$$f_M(\alpha\nu\nu') = \begin{cases} T(\lceil\nu\rceil, \lceil\alpha\rceil)(\lceil\nu'\rceil) & \text{if } \alpha \text{ is valid} \\ 0 & \text{otherwise} \end{cases}$$

Then the MTBDD encoding of M is $(\mathbf{x}, l_M(\mathbf{x}), \mathbf{z}, f_M(\mathbf{z}, \mathbf{x}, \mathbf{x}'))$. We will represent a WA by its MTBDD encoding from now on.

Example 3 *Consider the process* node_1 *in Figure 1 where the states are* s_i^1 *for* $i = 0, \ldots, 3$*, and the alphabet of actions is* $Act = \{start, start_1, go_1, done\}$*. We use* $\mathbf{x} = \langle s^1.0, s^1.1\rangle$ *to encode the set of states, and* $\mathbf{z} = \langle start, start_1, go_1, done\rangle$ *to encode the alphabet of actions. The MTBDD of* $f_{\mathsf{node}_1}(\mathbf{z}, \mathbf{x}, \mathbf{x}')$ *is shown in Figure 7. In the figure, the terminal node labled by the number* 0 *and its incoming edges are not shown. Solid edges are labeled by* 1 *and dotted edges are labeled by* 0*. Hence, for instance, the process* node_1 *transits from* $s_1^1 = \lceil\langle 0, 1\rangle\rceil$ *to* $s_2^1 = \lceil\langle 1, 0\rangle\rceil$ *on the action* go_1 *with probability* 0.9 *by the shaded path. Note that valuations of* $\mathbf{z}$ *need be valid. Thus only the valuations* $\langle 1, 0, 0, 0\rangle, \langle 0, 1, 0, 0\rangle, \langle 0, 0, 1, 0\rangle$*, and* $\langle 0, 0, 0, 1\rangle$ *of* $\mathbf{z}$ *yield non-zero values of* f_{node_1}*.*

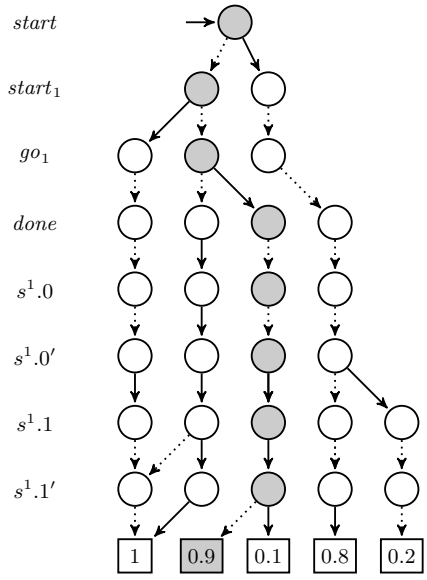


Figure 7: MTBDD of f_{node_1}

Using MTBDD's, Theorem 1 is rephrased as follows.

Corollary 1 *Let* $M_i = (\mathbf{x}_i, l_{M_i}(\mathbf{x}_i), \mathbf{z}, f_{M_i}(\mathbf{z}, \mathbf{x}_i, \mathbf{x}'_i))$ *be MDP's for* $i = 0, 1$ *and* $P_{\leq p}[\psi]$ *a probabilistic safety property. Then*

$$\frac{M_0 \preceq_e A \qquad A \| M_1 \models P_{\leq p}[\psi]}{M_0 \| M_1 \models P_{\leq p}[\psi]}$$

where $A = (\mathbf{x}_0, l_{M_0}(\mathbf{x}_0), \mathbf{z}, f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}'_0))$ *is a* $0/1$*-WA.*

5.2 The L^* Learning Algorithm for Regular Languages

We adapt the L^* algorithm to infer an MTBDD representing a weighted assumption [3]. L^* is a learning algorithm for regular languages. Assume a target regular language only known to a teacher. The L^* algorithm infers a minimal deterministic finite automaton recognizing the target regular language by posing the following queries to the teacher:

- A membership query asks if a string belongs to the target language; and
- An equivalence query asks if a conjectured finite automaton recognizes the target language. If not, the teacher has to provide the learning algorithm a string as a counterexample.

The L^* algorithm uses membership queries to construct the transition function of a deterministic finite automaton. When it constructs a deterministic finite automaton consistent with previous membership queries, L^* poses an equivalence query to check if the automaton does recognize the target regular language. If so, the algorithm has learned the target language correctly. Otherwise, the counterexample is used to improve the conjectured finite automaton.

It can be shown [3] that the L^* algorithm always infers the minimal deterministic finite automaton recognizing any target regular language within a polynomial number of queries.

5.3 An MTBDD Learning Algorithm

Since any $0/1$-WA can be represented by MTBDD's, we develop an MTBDD learning algorithm to infer weighted assumptions. Let $f(\mathbf{x})$ be an unknown target MTBDD. We assume a *teacher* to answer the following types of queries:

- On a *membership query* $MQ(\nu)$ with a valuation ν of $\mathbf{x}$, the teacher answers $f(\nu)$;
- On an *equivalence query* $EQ(g)$ with a *conjecture* MTBDD $g(\mathbf{x})$, the teacher answers YES if $f = g$. Otherwise, she returns a valuation ν of $\mathbf{x}$ with $f(\nu) \neq g(\nu)$ as a *counterexample*.

Observe that a valuation of $\mathbf{x}$ can be represented by a binary string of length $|\mathbf{x}|$. To illustrate how our MTBDD learning algorithm works, consider an unknown MTBDD $f(\mathbf{x})$ with exactly 2 values 0 and r. Since there are finitely many binary strings of length $|\mathbf{x}|$, the language R of binary strings representing valuations of $\mathbf{x}$ that evaluate f to r is regular. The L^* learning algorithm for regular languages hence can be used to infer a finite automaton recognizing the language R [3]. The learning algorithm applies the Myhill-Nerode theorem for regular languages. It constructs the transition function of the minimal deterministic finite automaton for any unknown regular language by posing membership and equivalence queries about the unknown target. Since the minimal deterministic finite automaton for R is structurally similar to the MTBDD f with two terminal nodes [28], the L^* algorithm can be modified to infer MTBDD's with two terminal nodes [19].

Generally, an unknown MTBDD $f(\mathbf{x})$ has k values $r_1, r_2, \ldots, r_k$. It evaluates to a value r_i on a valuation of $\mathbf{x}$. Moreover, the language R_i of binary strings representing valuations of $\mathbf{x}$ that evaluate f to r_i is regular for every $1 \leq i \leq k$. Consider generalized deterministic finite automata with k acceptance types. On any binary string, the computation of a generalized deterministic finite automaton ends in a state of an acceptance type. Formally, define a *k-language* $\mathcal{L}$ over an alphabet Σ to be a partition $\{L_1, L_2, \ldots, L_k\}$ of Σ^*. That is, $\cup_i L_i = \Sigma^*$ and $L_i \cap L_j = \emptyset$ when $i \neq j$. A *k-deterministic finite automaton* (k-*DFA*) $D = (Q, \Sigma, \delta, q_0, \mathcal{F})$ consists of a finite *state set* Q, an *alphabet* Σ, a *transition function* $\delta : Q \times \Sigma \to Q$, an *initial state* $q_0 \in Q$, and *acceptance types* $\mathcal{F} = \{F_1, F_2, \ldots, F_k\}$ where F_i's form a partition of Q. Define $\delta^*(q, \epsilon) = q$ and $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$ where $a \in \Sigma$ and $w \in \Sigma^*$. For a string $w \in \Sigma^*$, we say D *accepts w with type i* if $\delta^*(q_0, w) \in F_i$. Let $L_i(D) = \{w : D \text{ accepts } w \text{ with type } i\}$. A k-DFA hence *accepts* a k-language $\mathcal{L}(D) = \{L_i(D) : 1 \leq i \leq k\}$. It is almost straightforward to show a generalized Myhill-Nerode theorem for k-DFA.

Theorem 2 *The following statements are equivalent:*

1. *A k-language $\mathcal{L} = \{L_1, L_2, \ldots, L_k\}$ is accepted by a k-DFA;*
2. *Define the relation R over Σ^* such that xRy if and only if for every $z \in \Sigma^*$, $xz, yz \in L_i$ for some i. R is of finite index.*

In order to learn general MTBDD's, we modify the L^* algorithm to generate k-DFA. Consider binary strings of length $|\mathbf{x}|$ representing valuations of $\mathbf{x}$. Since an MTBDD evaluates a valuation to a value, the values of an MTBDD partition $\Sigma^{|\mathbf{x}|}$. With $\Sigma^* \setminus \Sigma^{|\mathbf{x}|}$, an MTBDD in fact gives a partition of Σ^*. In other words, an MTBDD defines a k-language. By Theorem 2, the modified L^* algorithm infers a minimal k-DFA that accepts the k-language defined by an unknown MTBDD. It remains to derive an MTBDD learning algorithm from the modified L^* algorithm for k-DFA.

Two minor problems need to be addressed in the design of our MTBDD learning algorithm. First, the modified L^* algorithm makes membership queries on binary strings of arbitrary lengths. The teacher for learning MTBDD's only answers membership queries on valuations over fixed variables. Second, the modified L^* algorithm presents a k-DFA as a conjecture in an equivalence query. The MTBDD teacher however accepts MTBDD's as conjectures. To solve these problems, we apply the techniques in [19].

When the modified L^* algorithm asks a membership query on a binary string, our MTBDD learning algorithm checks if the string has length $|\mathbf{x}|$. If not, the MTBDD learning algorithm returns 0 to denote the weight 0. Otherwise, the MTBDD learning algorithm forwards the corresponding valuation of $\mathbf{x}$ to the teacher and returns the teacher's answer to the modified L^* algorithm. When the modified L^* algorithm gives a k-DFA in an equivalence query, the MTBDD learning algorithm transforms the automaton into an MTBDD. It basically turns the initial state into a root, each state at distance less than n into a non-terminal node labeled with variable x_i, and each state at distance n into a terminal node.

Theorem 3 *Let $f(\mathbf{x})$ be a target MTBDD. The MTBDD learning algorithm outputs f in polynomial time, using $O(|f|^2 + |f| \log |\mathbf{x}|)$ membership queries and at most $|f|$ equivalence queries.*

Proof: *The modified L^* algorithm outputs the minimal k-DFA F, using $O(|F|^2 + |F| \log m)$ membership queries and at most $|F|$ equivalence queries where m is the length of the longest counterexample. Every membership or equivalence query of the modified L^* algorithm induces at most one query in the MTBDD learning algorithm. When the modified L^* algorithm makes an equivalence query with a k-DFA, the MTBDD learning algorithm transforms it into an MTBDD of the same size in polynomial time. Whenever a counterexample is obtained from the MTBDD teacher, the MTBDD learning algorithm forwards the corresponding binary string of length $|\mathbf{x}|$ to the modified L^* algorithm. Hence the learning algorithm infers the MTBDD f with $O(|f|^2 + |f| \log |\mathbf{x}|)$ membership and $|f|$ equivalence queries.* □

6. The Learning-based Verification Framework

With our new assume-guarantee reasoning proof rule (Section 4) and learning algorithm for MTBDD's (Section 5), we can now describe our sound and complete learning framework.

Let $M_i = (\mathbf{x}_i, l_{M_i}(\mathbf{x}_i), \mathbf{z}, f_{M_i}(\mathbf{z}, \mathbf{x}_i, \mathbf{x}_i'))$ be MDP's ($i = 0, 1$), and $P_{\leq p}[\psi]$ a probabilistic safety property. To verify if $M_0 || M_1 \models P_{\leq p}[\psi]$ holds, we aim to generate a $0/1$-WA $A = (\mathbf{x}_0, l_{M_0}(\mathbf{x}_0), \mathbf{z}, f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}_0'))$ to fulfill the premises $M_0 \preceq_e A$ and $A || M_1 \models P_{\leq p}[\psi]$. To find such a weighted assumption A, we use the MTBDD learning algorithm to infer an MTBDD $f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}_0')$ as the weighted transition function. Recall that the MTBDD learning algorithm relies on a teacher to answer queries about the target MTBDD. We therefore will design a mechanical teacher to answer queries from the learning algorithm (Figure 8).

Let α be a valuation encoding an action, ν and ν' valuations encoding states. The mechanical teacher consists of the membership query resolution algorithm $ResolveMQ(\alpha\nu\nu')$ and the equivalence query resolution algorithm $ResolveEQ(f_A)$. The membership query resolution algorithm answers a membership query $MQ(\alpha\nu\nu')$ by the weight associated with the transition from $\lceil\nu\rceil$ to $\lceil\nu'\rceil$ on action $\lceil\alpha\rceil$ in a weighted assumption fulfilling the premises of the assume-guarantee reasoning proof rule. Similarly, the equivalence query resolution algorithm answers an equivalence query $EQ(f_A)$ by checking whether the MTBDD f_A represents the weighted transition function of a weighted assumption. The equivalence query resolution algorithm should return a counterexample when f_A does not represent a suitable weighted transition function. Recall that M_0 itself is trivially a weighted assumption. Our teacher simply uses the weighted transition function f_{M_0} of M_0 as the target. In the worst case, our framework will find the weighted assumption M_0 and hence attain completeness. In practice, it often finds useful weighted assumptions before M_0 is inferred.

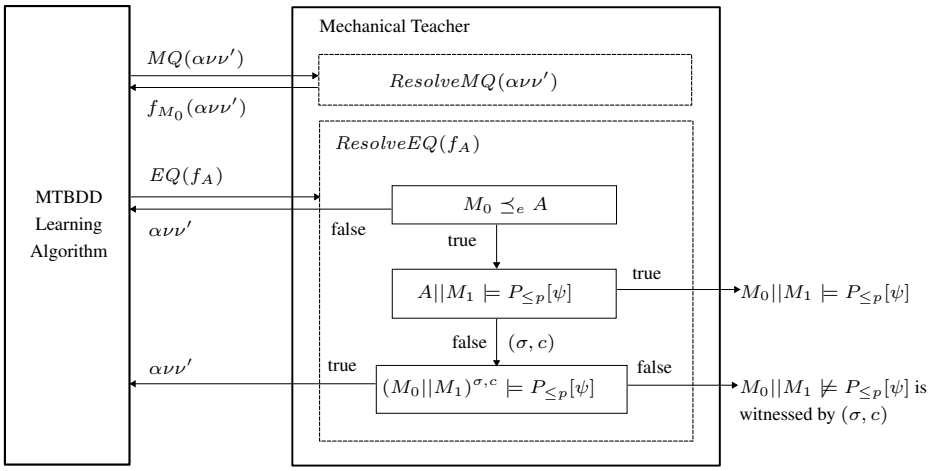


Figure 8: Learning Framework for Compositional Verification

6.1 Resolving Membership Queries

Our membership query resolution algorithm targets the weighted transition function of M_0. Clearly, M_0 embeds itself and hence can be used as a weighted assumption. On the membership query $MQ(\alpha\nu\nu')$, the mechanical teacher simply returns $f_{M_0}(\alpha\nu\nu')$.

Input : $MQ(\alpha\nu\nu')$
Output: a rational number
answer $MQ(\alpha\nu\nu')$ with $f_{M_0}(\alpha\nu\nu')$;

Algorithm 1: $ResolveMQ(\alpha\nu\nu')$

6.2 Resolving Equivalence Queries

On an equivalence query $EQ(f_A)$, the mechanical teacher is given an MTBDD $f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}_0')$. Consider the WA $A = (\mathbf{x}_0, l_{M_0}(\mathbf{x}_0), \mathbf{z},\ f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}_0'))$. We need to verify if both premises of the assume-guarantee reasoning proof rule in Corollary 1 hold. The equivalence query resolution algorithm first checks if $M_0 \preceq_e A$. If not, there are valuations α, ν, and ν' with $f_{M_0}(\alpha\nu\nu') > f_A(\alpha\nu\nu')$. The equivalence query resolution algorithm returns $\alpha\nu\nu'$ as a counterexample to $EQ(f_A)$.

If $M_0 \preceq_e A$, the equivalence query resolution algorithm continues to check whether $A\|M_1 \models P_{\leq p}[\psi]$ holds by model checking. If $A\|M_1 \models P_{\leq p}[\psi]$ holds, the MTBDD learning algorithm has inferred a weighted assumption that establishes $M_0\|M_1 \models P_{\leq p}[\psi]$ by the assume-guarantee reasoning proof rule in Corollary 1. Otherwise, the equivalence query resolution algorithm obtains a weighted witness (σ, c) to $A\|M_1 \not\models P_{\leq p}[\psi]$ from model checking. It then checks if the weighted witness is spurious. Recall that $M_0\|M_1$ and $A\|M_1$ have the same state set and action alphabet due to $M_0\|M_1 \preceq_e A\|M_1$. The (σ, c)-fragment $(M_0\|M_1)^{\sigma,c}$ is well-defined. If $(M_0\|M_1)^{\sigma,c} \models P_{\leq p}[\psi]$, the weighted witness (σ, c) is spurious. The algorithm then analyzes the spurious weighted witness (σ, c) and returns a valuation as the counterexample. Otherwise, the algorithm concludes $(M_0\|M_1)^{\sigma,c} \not\models P_{\leq p}[\psi]$ with the weighted witness (σ, c) (Algorithm 2).

Example 4 *Consider the weighted witness in Figure 4. The (σ, c)-fragment $(\mathsf{node}_1\|\mathsf{node}_2)^{\sigma,c}$ is shown in Figure 5. There is but one path in $(\mathsf{node}_1\|\mathsf{node}_2)^{\sigma,c}$. This path ends in $\langle s_3^1 s_3^2\rangle$ and hence satisfies ψ_{failed}. Its weight however is $0.64 \times 0.1 \times 0.1 = 0.0064 \leq 0.01$. Thus $(\mathsf{node}_1\|\mathsf{node}_2)^{\sigma,c} \models P_{\leq 0.01}[\psi_{failed}]$. The weighted witness in Figure 4 is spurious.*

Input : $EQ(f_A)$
Output: YES, a counterexample to $EQ(f_A)$
$A \leftarrow (\mathbf{x}_0, l_{M_0}(\mathbf{x}_0), \mathbf{z}, f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}_0'))$;
if $\exists \alpha\nu\nu'.f_A(\alpha\nu\nu') < f_{M_0}(\alpha\nu\nu')$ **then**
  answer $EQ(f_A)$ with the counterexample $\alpha\nu\nu'$;
  receive a new equivalence query $EQ(f_{A'})$;
  call $ResolveEQ(f_{A'})$;
if $A||M_1 \models P_{\leq p}[\psi]$ **then**
  answer $EQ(f_A)$ with YES;
  return "$M_0||M_1 \models P_{\leq p}[\psi]$";
else
  let (σ, c) be a weighted witness to $A||M_1 \not\models P_{\leq p}[\psi]$;
  if $(M_0\|M_1)^{\sigma,c} \models P_{\leq p}[\psi]$ **then**
    select a transition $\lceil\mu\rceil \xrightarrow{\lceil\alpha\rceil} \lceil\mu'\rceil$ with the maximal contribution from $(A\|M_1)^{\sigma,c}$;
    answer $EQ(f_A)$ with $\alpha{\uparrow_{\mathbf{x}_0}}\mu'{\uparrow_{\mathbf{x}_0'}}$;
    receive a new equivalence query $EQ(f_{A'})$;
    call $ResolveEQ(f_{A'})$;
  else
    return "$M_0||M_1 \not\models P_{\leq p}[\psi]$" with (σ, c);
  end
end

Algorithm 2: $ResolveEQ(f_A)$

Selecting Counterexamples. Given a spurious weighted witness (σ, c), the mechanical teacher selects a transition from c as a counterexample to the MTBDD learning algorithm. The counterexample is intended to remove the spurious weighted witness (σ, c) from weighted assumptions.

Let (σ, c) be a spurious weighted witness with $(A\|M_1)^{\sigma,c} \not\models P_{\leq p}[\psi]$ and $(M_0\|M_1)^{\sigma,c} \models P_{\leq p}[\psi]$. Recall that $(M_0\|M_1)^{\sigma,c}$ and $(A\|M_1)^{\sigma,c}$ have the same state set, initial state, and alphabet. The only differences between $(M_0\|M_1)^{\sigma,c}$ and $(A\|M_1)^{\sigma,c}$ are the weights associated with transitions. In order to remove the spurious weighted witness (σ, c), we would like to select transitions which differentiate $(M_0\|M_1)^{\sigma,c}$ from $(A\|M_1)^{\sigma,c}$ most significantly.

More precisely, for any transition t in c, let $\Pi_A(t)$ and $\Pi_0(t)$ be the sets of paths in respectively $(A\|M_1)^{\sigma,c}$ and $(M_0\|M_1)^{\sigma,c}$ which contain transition t. Define $\omega(t) = Wt(\Pi_A(t)) - Wt(\Pi_0(t))$ to be the *contribution* of transition t in the spurious weighted witness (σ, c). The mechanical teacher simply selects a transition t

with the maximal contribution. The weight of the selected transition in A will be revised to the probability of the corresponding transition in M_0. Its contribution will be 0 in following revisions. Note that contributions of transitions are computed using MTBDD's for efficiency. Details are omitted due to space limit.

Observe moreover that selecting one transition may not eliminate the spurious weighted witness. Since a spurious weighted witness contains several transitions, the weight of the witness may not be reduced sufficiently after revising a few transitions. Subsequently, the same spurious weighted witness may be recomputed by model checking the premises with a revised weighted assumption. In order to reduce the number of model checking invocations, we reuse the same spurious weighted witness to compute counterexamples [25]. More precisely, our implementation checks if the current spurious weighted witness is eliminated from revised weighted assumptions. If not, the mechanical teacher selects another transition from the spurious weighted witness to further refine the revised weighted assumptions. Since a spurious weighted witness is used to revise several weighted assumptions, the number of model checking invocations is reduced.

6.3 Correctness

The correctness of our assume-guarantee reasoning framework for probabilistic systems follows from Theorem 1. We establish the soundness, completeness, and termination of the new learning-based framework in the remainder of this section.

Theorem 4 (Soundness) *Let* $M_i = (\mathbf{x}_i, l_{M_i}(\mathbf{x}_i), \mathbf{z}, f_{M_i}(\mathbf{z}, \mathbf{x}_i, \mathbf{x}'_i))$ *be MDP's for* $i = 0, 1$, $P_{\leq p}[\psi]$ *a probabilistic safety property, and* $f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}'_0)$ *an MTBDD.*

- *If ResolveEQ*(f_A) *returns "*$M_0||M_1 \models P_{\leq p}[\psi]$*," then* $M_0 \| M_1 \models P_{\leq p}[\psi]$ *holds.*
- *If ResolveEQ*(f_A) *returns "*$M_0||M_1 \not\models P_{\leq p}[\psi]$*" with* (σ, c)*, then* (σ, c) *is a weighted witness to* $M_0||M_1 \not\models P_{\leq p}[\psi]$.

Proof: *When our learning-based framework reports "*$M_0 \| M_1 \models P_{\leq p}[\psi]$*" in Algorithm 2, a weighted assumption* $A = (\mathbf{x}_0, l_{M_0}(\mathbf{x}_0), \mathbf{z}, f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}'_0))$ *such that* $M_0 \preceq_e A$ *and* $A \| M_1 \models P_{\leq p}[\psi]$ *has been inferred. By the soundness of the assume-guarantee reasoning proof rule (Theorem 1),* $M_0 \| M_1 \models P_{\leq p}[\psi]$. *On the other hand, suppose our learning-based framework reports "*$M_0 \| M_1 \not\models P_{\leq p}[\psi]$*." The weighted witness* (σ, c) *to* $A \| M_1 \not\models P_{\leq p}[\psi]$ *has been verified to be a witness to* $M_0 \| M_1 \not\models P_{\leq p}[\psi]$. □

Theorem 5 (Completeness) *Let* $M_i = (\mathbf{x}_i, l_{M_i}(\mathbf{x}_i), \mathbf{z}, f_{M_i}(\mathbf{z}, \mathbf{x}_i, \mathbf{x}'_i))$ *be MDP's for* $i = 0, 1$*, and* $P_{\leq p}[\psi]$ *a probabilistic safety property.*

- *If* $M_0||M_1 \models P_{\leq p}[\psi]$*, then ResolveEQ*$(f_A)$ *returns "*$M_0||M_1 \models P_{\leq p}[\psi]$*" for some MTBDD* $f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}'_0)$.
- *If* $M_0||M_1 \not\models P_{\leq p}[\psi]$*, then ResolveEQ*$(f_A)$ *returns "*$M_0||M_1 \not\models P_{\leq p}[\psi]$*" with a weighted witness* (σ, c).

Proof: *In our framework, the MTBDD learning algorithm targets the weighted transition function of* M_0*. It will infer* $f_{M_0}(\mathbf{z}, \mathbf{x}_0, \mathbf{x}'_0)$ *eventually (Theorem 3). If* $M_0 \| M_1 \models P_{\leq p}[\psi]$*, the learning algorithm always infers a weighted assumption* A *(in the worst case,* A *is* M_0*) such that* $M_0 \preceq_e A$ *and* $A \| M_1 \models P_{\leq p}[\psi]$*. Hence ResolveEQ*$(f_A)$ *returns "*$M_0 \| M_1 \models P_{\leq p}[\psi]$*." Otherwise, the learning algorithm always infers a weighted assumption* A *(in the worst case,* A *is* M_0*) such that* $M_0 \| M_1 \not\models P_{\leq p}[\psi]$ *is witnessed by* (σ, c)*. ResolveEQ*(f_A) *returns "*$M_0 \| M_1 \not\models P_{\leq p}[\psi]$*."* □

Theorem 6 (Termination) *Let* $M_i = (\mathbf{x}_i, l_{M_i}(\mathbf{x}_i), \mathbf{z}, f_{M_i}(\mathbf{z}, \mathbf{x}_i, \mathbf{x}'_i))$ *be MDP's for* $i = 0, 1$*, and* $P_{\leq p}[\psi]$ *a probabilistic safety property. Our learning-based framework reports "*$M_0 \| M_1 \models P_{\leq p}[\psi]$*" or "*$M_0 \| M_1 \not\models P_{\leq p}[\psi]$*" within a polynomial number of queries in* $|f_{M_0}(\mathbf{z}, \mathbf{x}_0, \mathbf{x}'_0)|$ *and* $|\mathbf{z} \cup \mathbf{x}_0 \cup \mathbf{x}'_0|$.

Proof: *In our learning-based framework, the MTBDD learning algorithm targets the weighted transition function of* M_0*. It will infer the target MTBDD* $f_{M_0}(\mathbf{z}, \mathbf{x}_0, \mathbf{x}'_0)$ *using* $O(n^2 + n \log m)$ *membership queries and at most* n *equivalence queries where* $n = |f_{M_0}(\mathbf{z}, \mathbf{x}_0, \mathbf{x}'_0)|$ *and* $m = |\mathbf{z} \cup \mathbf{x}_0 \cup \mathbf{x}'_0|$ *(Theorem 3). At this point, the weighted assumption* A *is* M_0*. The mechanical teacher reports either "*$M_0 \| M_1 \models P_{\leq p}[\psi]$*" or "*$M_0 \| M_1 \not\models P_{\leq p}[\psi]$*."* □

7. Experiments

We have implemented a prototype of our compositional verification technique on top of PRISM 4.0.1 [38]. It accepts an MDP specified in the PRISM modeling language and a probabilistic safety property. The MTBDD learning algorithm is implemented by modifying the L^* algorithm in libalf 0.3 library [9] with CUDD 2.5.0 package.[2] The membership query resolution algorithm (Algorithm 1) and the embedded checking algorithm (Algorithm 2) are implemented using CUDD. Probabilistic model checking in the equivalence query resolution algorithm (Algorithm 2) is performed by PRISM (using the MTBDD engine). We generate counterexamples by the techniques in [23]. All experiments were run on a virtual machine with 2.6GHz CPU and 4GB RAM.

Our compositional approach is evaluated on several parameterized examples. All examples are derived from the PRISM website.[3] For each model, we check a probabilistic safety property. All models and properties are briefly described below:

- *Consensus* models a randomized coin algorithm [4] which allows N processes in a distributed network to reach a consensus by accessing a global shared counter parameterized by K. The specification describes "the probability that eventually all processes make a decision but some two processes do not agree on the same value is at most p".
- *WLAN* models the wireless local area networks specified in the IEEE 802.11 standard [2]. Stations of a wireless network cannot listen to their own transmission. Each station has a backoff counter (with the maximal value of B) to minimize the likelihood of transmission collision. The time bounded version of this model is considered. The specification describes "the probability that either station's backoff counter hits the number K within some time is at most p".
- *FireWire* models the tree identify protocol of the IEEE 1394 high performance serial bus [1]. Among nodes connected in a network, a root node needs to be elected to act as the manager of the bus in the network. The time bound for message transmission is parameterized by $deadline$. The implementation version of this model is considered. The specification describes "the probability that a root node is elected eventually before some time deadline passes is at most p".
- *Philo* models a randomized solution to the dining philosophers problem [33]. N philosophers sit around a circular table. Neighboring philosophers share a resource. A philosopher can eat if he obtains the resources from both sides. The specification describes "the probability that neighboring philosophers do not obtain their shared resource simultaneously is at most 0.00001."

Our tool selects one process of the model as M_0 and the composition of other processes as M_1. Selecting the composition of multiple processes as M_0 can be done by solving the two-way decomposition problem [14, 43]. Here we employ a simple heuristic:

[2] `http://vlsi.colorado.edu/~fabio/CUDD/`

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

choose the process with the minimal *interface alphabet*. The interface alphabet of a process is the set of shared actions. For example, the *WLAN* model consists of four processes: *medium*, *station1*, *station2* and *timer*, with interface alphabets $\{send1, send2, finish1, finish2\}$, $\{time, finish1, send1\}$, $\{time, finish2, send2\}$ and $\{time\}$, respectively. We choose *timer* as M_0 by our heuristic.

The first experiment compares the performance of our compositional approach (*compositional*) with the monolithic probabilistic model checking in PRISM (*monolithic*). Experimental results are listed in Table 1. For each model and a corresponding probabilistic safety property $P_{\leq p}[\psi]$, we compute the property of $P_{max=?}[\psi]$ by PRISM and report it in the P_{max} column. Note that the property $P_{\leq p}[\psi]$ holds on the model if and only if $p \geq P_{max}$. The satisfiability of the property $P_{\leq p}[\psi]$ on the model is reported in the *Result* column. For each test case, we show the model size (*size*) and run time (*time*). The model size counts the number of MTBDD nodes for the weighted transition function of the composed model.[4] The run time includes the time spent on all stages, including model construction, model checking, witness analysis, and assumption learning. For monolithic approach, both PRISM with the MTBDD engine (PRISM-M) and PRISM with the hybrid engine (PRISM-H) are tested. Their run times are listed in T_M and T_H columns respectively. For compositional approach, the number of PRISM calls (*#Call*) is also reported. The last column (*reduction*) shows the reduction of model size and time of our compositional approach to PRISM-M. All time is in seconds, and the symbol "–" indicates either time-out (4 hours) or memory-out (4GB).

The results are very encouraging. In 20 of 23 cases, the compositional verifier outperforms PRISM-M significantly. Moreover, a reduction of 90% in time is achieved in 9 cases; and a reduction of 80% in model sizes is attained in 8 cases. Our compositional approach benefits the verification by avoiding the construction of the whole model. In the *size reduction* column of Table 1, our compositional approach succeeds in learning an assumption A such that the size of $A \| M_1$ is much smaller than that of $M_0 \| M_1$ in most cases. Only for the two smallest unsatisfied cases in the *Consensus* example, our compositional approach performs worse. One possible reason is that the sizes of the models are so small that compositional verification is redundant. Also, observe that PRISM-H performs much better then PRISM-M in the *Consensus* example. MTBDD-based techniques (monolithic or not) may not be the best choice for this example.

Hybrid (PRISM-H) and MTBDD-based (PRISM-M and ours) techniques can also be compared in Table 1. The results heavily depend on the examples. Similar phenomenons were also reported in [31]. For all cases in the *Consensus* example, PRISM-H performs much better than both MTBDD-based techniques. For all cases in the *Philos* example, the performances of PRISM-H and PRISM-M are similar. However, in the more realistic *WLAN* and *FireWire* examples, PRISM-H runs out of memory quickly when the model size becomes large.

The second experiment evaluates the impact of the probability bound p on the effectiveness of compositional verification. This experiment is performed on examples with different probability bounds. The results on the *WLAN* example with $(B, K) = (4, 2)$ are plotted in Figure 9(a). When p is above or nearly above P_{max} (≈ 0.1836), the performance of our approach goes down quickly. The reason is that the property becomes satisfied when $p \geq P_{max}$. More equivalence queries are then required to infer a proper assumption to prove both premises of the reasoning rule. On the other hand, if the probability bound p is less than P_{max}, a coarse weighted assumption suffices to verify the property. Similar phenomenons can be observed on the *Consensus* example with $(N, K) = (4, 2)$ (Figure 9(b)). The result from the *FireWire* example with *deadline* = 400 (Figure 9(c)) is quite different. Observe that the actual probability P_{max} of the *FireWire* examples is 1. Its properties are trivially unsatisfied for any p. Thus there is no rising edge in Figure 9(c) as in other figures. In the *Philos* example, the probability P_{max} is 0 and hence the properties are always satisfied for any p. Similar to *FireWire*, we do not observe any rising edge and hence skip the figure of the *Philos* example. Compositional verification however always outperforms the monolithic algorithm regardless of satisfiability of properties in both examples.

[4] By composed model, we mean $M_0 \| M_1$ for monolithic checker and $A \| M_1$ for our checker.

8. Related Works

The most relevant works to ours are [16, 17, 32]. In their proof rules, assumptions are classical deterministic finite automata. The L^* algorithm has been applied to infer classical assumptions in [16]. As discussed above, classical assumptions cannot express general probabilistic behaviors. Such techniques are sound but incomplete. We adopt weighted automata as assumptions to have a sound and invertible proof rule. Our technique is both sound and complete. A sound and invertible assume-guarantee reasoning proof rule for probabilistic I/O systems is given in [17]. The framework however only works for fully probabilistic discrete time Markov chains and may not terminate. Our technique in contrast applies to Markov decision processes and always terminates.

Undecidability of inferring labeled probabilistic transition systems under Angluin's active learning model is shown in [30]. A (necessarily) restricted learning algorithm for such probabilistic systems is also proposed in the same paper. In addition to learning different concepts, the restricted algorithm does not utilize membership queries whereas ours does. An alternative direction for generating probabilistic assumptions is to use abstraction refinement (AGAR) techniques [21]. In [29], probabilistic assumptions are conservative abstractions of system components. They are iteratively refined by counterexamples [12]. However, AGAR relies on partitioning the explicit state space to construct assumptions [21, 29]. We are not aware of any symbolic implementation of AGAR techniques for classical or probabilistic systems.

Various learning algorithms have been proposed for probabilistic systems [10, 34, 35, 40]. These learning algorithms adopt passive learning model. They are not applicable to the learning-based assume-guarantee reasoning framework in [13].

Learning algorithms for binary decision diagrams were proposed in [19, 37]. In [19], an L^*-based algorithm was developed. The work in [37] used a classification tree-based learning algorithm for regular languages. Both algorithms inferred deterministic finite automata and transformed them into decision diagrams.

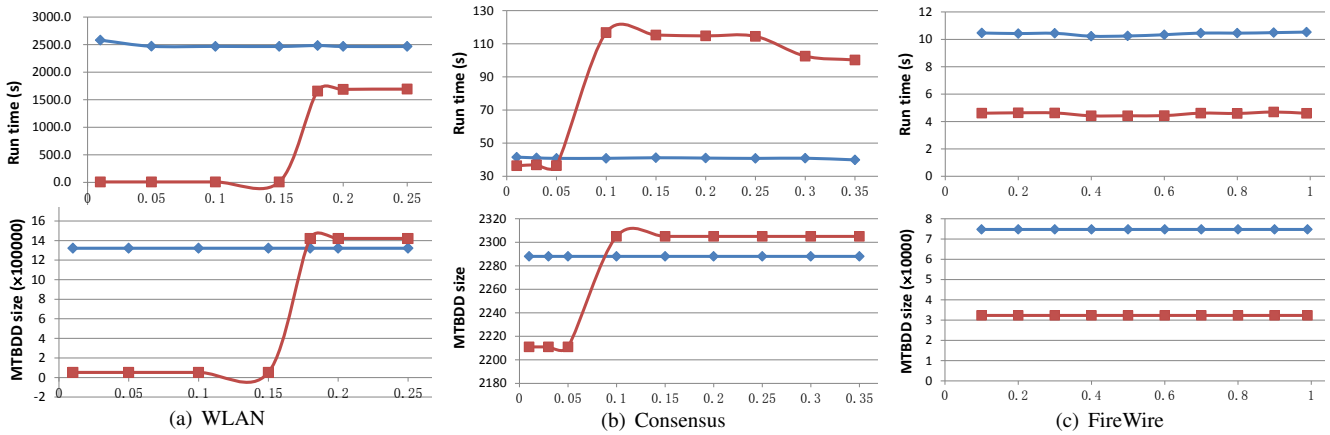
9. Conclusion

We proposed a sound and complete learning-based assume-guarantee reasoning technique for probabilistic safety properties on MDP's. Instead of probabilistic assumptions, we infer weighted assumptions for compositional verification. Using an MTBDD learning algorithm, our technique generates implicit representations of weighted assumptions. Experimental results show that the assume-guarantee reasoning technique outperforms the monolithic probabilistic model checking in most of the test cases.

Our technique can be applied to sequential probabilistic systems. Let M be an MDP and $P_{\leq p}[\psi]$ a probabilistic safety property. One generates a $0/1$-WA A such that $M \preceq_e A$ and $A \models P_{\leq p}[\psi]$ by our learning-based technique. It is however not recommended when M is composed of concurrent MDP's. Since the construction of the composition can be very expensive, the computation should

Table 1: Experimental Results: Monolithic vs. Compositional

Example	Param	P_{max}	Result	Monolithic			Compositional			Reduction(%)	
				T_H	T_M	Size	#Call	T	Size	Time	Size
Consensus (p=0.01)	(2, 6)	0.04	false	0.16	3.86	393	6	6.08	423	-57.5	-7.6
	(2, 10)	0.02	false	0.50	17.36	417	6	24.87	460	-43.3	-10.3
	(4, 2)	0.29	false	2.86	40.22	2288	6	35.79	2211	11.0	3.4
	(4, 6)	0.10	false	42.15	1109.71	2340	6	703.53	2269	36.6	3.0
	(4, 10)	0.06	false	169.21	4747.15	2384	6	3051.72	2327	35.7	2.4
	(6, 2)	0.36	false	411.57	1611.16	7075	6	1068.06	6753	33.7	4.6
	(6, 4)	0.19	false	2841.36	10270.29	7055	6	5751.38	6728	44.0	4.6
WLAN (p=0.1)	(2, 2)	0.18	false	430.21	381.38	304314	4	5.91	33237	98.5	89.1
	(3, 2)	0.18	false	–	675.15	626744	4	5.94	40945	99.1	93.5
	(3, 3)	0.02	true	–	687.04	626744	11	977.89	664801	-42.3	-6.1
	(4, 2)	0.18	false	–	2620.32	1321221	4	7.64	52129	99.7	96.1
	(4, 3)	0.02	true	–	2530.00	1321221	11	1776.14	1421456	29.8	-7.6
	(4, 4)	0.00	true	–	2683.98	1321221	11	2061.69	1421456	23.2	-7.6
FireWire (p=0.1)	200	1.00	false	182.58	86.04	836852	4	37.06	129750	56.9	84.5
	400	1.00	false	1897.39	507.58	1220933	4	38.92	129761	92.3	89.4
	600	1.00	false	–	954.33	1243547	4	39.17	129778	95.9	89.6
	800	1.00	false	–	1380.22	1243552	4	39.21	129772	97.2	89.6
	1000	1.00	false	–	1803.40	1243551	4	39.52	129780	97.8	89.6
Philos (p=0.01)	20	0.00	true	29.80	29.74	58565	1	13.43	19328	54.9	67.0
	25	0.00	true	55.98	56.36	91325	1	31.48	29908	44.1	67.3
	30	0.00	true	109.01	108.84	131260	1	54.71	42763	49.7	67.4
	35	0.00	true	1006.60	1022.58	178370	1	96.45	57893	90.6	67.5
	40	0.00	true	2370.48	2348.22	232655	1	168.14	75298	92.8	67.6



(a) WLAN (b) Consensus (c) FireWire

Figure 9: Impacts of p on the Performance of MTBDD-Based Approaches (red for compositional, blue for monolithic).

be deferred after concurrent components are simplified. Assume-guarantee reasoning presented in this paper is certainly preferred.

Currently, our PRISM-based implementation receives a finite set of paths as weighted witnesses to $M \not\models P_{\leq p}[\psi]$. Generally, weighted witnesses to $M \not\models P_{\lhd p}[\psi]$ where $\lhd \in \{<, \leq\}$ are represented as graphs with strongly connected components [41, 42]. We plan to generalize transition contributions to select counterexamples from spurious weighted witnesses with strongly connected components. We moreover would like to extend our learning framework to verifying richer properties such as general probabilistic safety or liveness properties.

Acknowledgments

Fei He (corresponding author) and Xiaowei Gao are supported by the Chinese National 973 Plan (2010CB328003); the NSF of China (61272001, 60903030, 91218302); the Chinese National Key Technology R&D Program (SQ2012BAJY4052), the Importation and Development of High-Caliber Talents Project of Beijing Municipal Institutions (YETP0167), and the Tsinghua University Initiative Scientific Research Program; Bow-Yaw Wang (corresponding author) is supported by the Ministry of Science and Technology of Taiwan (103-2221-E-001 -020 -MY3); Lijun Zhang (corresponding author) is supported by the Natural Science Foundation of China (NSFC) under grant No. 61472473, 61428208, 61361136002, the CAS/SAFEA International Partnership Program for Creative Research Teams.

References

[1] IEEE standard for a high-performance serial bus. *IEEE Std 1394-2008*, pages 1–954, Oct 2008.

[2] IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, March 2012.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[4] J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *Journal of Algorithms*, 11(3):441–460, 1990.

[5] C. Baier and J.-P. Katoen. *Principles of model checking*. MIT Press, 2008.

[6] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *ICALP*, volume 1256 of *LNCS*, pages 430–440. Springer, 1997.

[7] A. Beimel, F. Bergadano, N. H. Bshouty, E. Kushilevitz, and S. Varricchio. Learning functions represented as multiplicity automata. *Journal of ACM*, 47(3):506–530, May 2000. .

[8] A. Bianco and L. de Alfaro. Model checking of probabalistic and nondeterministic systems. In *FSTTCS*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[9] B. Bollig, J.-P. Katoen, C. Kern, M. Leucker, D. Neider, and D. R. Piegdon. libalf: The automata learning framework. In *CAV*, volume 6174 of *LNCS*, pages 360–364. Springer, 2010.

[10] Y. Chen, H. Mao, M. Jaeger, T. Nielsen, K. Guldstrand Larsen, and B. Nielsen. Learning Markov models for stationary system behaviors. In *NASA Formal Methods*, volume 7226 of *LNCS*, pages 216–230. Springer, 2012.

[11] Y.-F. Chen, E. M. Clarke, A. Farzan, M.-H. Tsai, Y.-K. Tsay, and B.-Y. Wang. Automated assume-guarantee reasoning through implicit learning. In *CAV*, volume 6174 of *LNCS*, pages 511–526. Springer, 2010.

[12] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, volume 1855 of *LNCS*, pages 154–169. Springer, 2000.

[13] J. M. Cobleigh, D. Giannakopoulou, and C. S. Păsăreanu. Learning assumptions for compositional verification. In *TACAS*, volume 2619 of *LNCS*, pages 331–346. Springer, 2003.

[14] J. M. Cobleigh, G. S. Avrunin, and L. A. Clarke. Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 17 (2):7, 2008.

[15] L. De Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In *TACAS*, volume 1758 of *LNCS*, pages 395–410. Springer, 2000.

[16] L. Feng, M. Kwiatkowska, and D. Parker. Compositional verification of probabilistic systems using learning. In *QEST*, pages 133–142. IEEE, 2010.

[17] L. Feng, T. Han, M. Kwiatkowska, and D. Parker. Learning-based compositional verification for synchronous probabilistic systems. In *ATVA*, volume 6996 of *LNCS*, pages 511–521. Springer-Verlag, 2011.

[18] M. Fujita, P. C. McGeer, and J.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design*, 10(2/3):149–169, 1997.

[19] R. Gavaldà and D. Guijarro. Learning ordered binary decision diagrams. In *ALT*, volume 997 of *LNCS*, pages 228–238. Springer, 1995.

[20] M. Gheorghiu, D. Giannakopoulou, and C. S. Păsăreanu. Refining interface alphabets for compositional verification. In *TACAS*, volume 4424 of *LNCS*, pages 292–307. Springer, 2007.

[21] M. Gheorghiu Bobaru, C. S. Păsăreanu, and D. Giannakopoulou. Automated assume-guarantee reasoning by abstraction refinement. In *CAV*, volume 5123 of *LNCS*, pages 135–148. Springer, 2008.

[22] A. Gupta, K. L. McMillan, and Z. Fu. Automated assumption generation for compositional verification. In *CAV*, volume 4590 of *LNCS*, pages 420–432. Springer, 2007.

[23] T. Han, J.-P. Katoen, and D. Berteun. Counterexample generation in probabilistic model checking. *IEEE Transactions on Software Engineering*, 35(2):241–257, 2009.

[24] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[25] F. He, B.-Y. Wang, L. Yin, and L. Zhu. Symbolic assume-guarantee reasoning through BDD learning. In *ICSE*, pages 1071–1082. ACM, 2014.

[26] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *TACAS*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[27] J.-P. Katoen, L. Song, and L. Zhang. Probably safe or live. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 55:1–55:10. ACM, 2014.

[28] S. Kimura and E. M. Clarke. A parallel algorithm for constructing binary decision diagrams. In *ICCD*, pages 220–223. IEEE, 1990.

[29] A. Komuravelli, C. S. Păsăreanu, and E. M. Clarke. Assume-guarantee abstraction refinement for probabilistic systems. In *CAV*, volume 7358 of *LNCS*, pages 310–326. Springer, 2012.

[30] A. Komuravelli, C. S. Păsăreanu, and E. M. Clarke. Learning probabilistic systems from tree samples. In *LICS*, pages 441–450. IEEE, 2012.

[31] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer*, 6(2):128–142, 2004.

[32] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In *TACAS*, volume 6015 of *LNCS*, pages 23–37. Springer, 2010.

[33] D. Lehmann and M. O. Rabin. On the advantage of free choice: A symmetric and fully distributed solution to the dining philosophers problem (extended abstract). In *POPL*, pages 133–138. ACM, 1981.

[34] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen. Learning probabilistic automata for model checking. In *QEST*, pages 111–120. IEEE, 2011.

[35] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen. Learning Markov decision processes for model checking. *arXiv preprint arXiv:1212.3873*, 2012.

[36] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[37] A. Nakamura. An efficient query learning algorithm for ordered binary decision diagrams. *Information and Computation*, 201(2):178–198, 2005.

[38] D. A. Parker. *Implementation of symbolic model checking for probabilistic systems*. PhD thesis, University of Birmingham, 2002.

[39] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In *CONCUR*, volume 836 of *LNCS*, pages 481–496. Springer, 1994.

[40] W.-G. Tzeng. Learning probabilistic automata and Markov chains via queries. *Machine Learning*, 8(2):151–166, 1992.

[41] R. Wimmer, N. Jansen, E. Ábrahám, B. Becker, and J.-P. Katoen. Minimal critical subsystems for discrete-time Markov models. In *TACAS*, volume 7214 of *LNCS*, pages 299–314. Springer, 2012.

[42] R. Wimmer, N. Jansen, A. Vorpahl, E. Ábrahám, J.-P. Katoen, and B. Becker. High-level counterexamples for probabilistic automata. In *QEST*, pages 39–54. IEEE, 2013.

[43] H. Zhu, F. He, W. N. Hung, X. Song, and M. Gu. Data mining based decomposition for assume-guarantee reasoning. In *FMCAD*, pages 116–119. IEEE, 2009.