Learning Weighted Assumptions for Compositional Verification of Markov Decision Processes

FEI HE, XIAOWEI GAO, and MIAOFEI WANG, KLiss, MoE; TNList; School of Software, Tsinghua University
BOW-YAW WANG, Academia Sinica
LIJUN ZHANG, Chinese Academy of Sciences

Probabilistic models are widely deployed in various systems. To ensure their correctness, verification techniques have been developed to analyze probabilistic systems. 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 multiterminal binary decision diagrams (MTBDDs), we give an L^*-based learning algorithm for MTBDDs to infer weighted assumptions. Experimental results suggest promising outlooks for our compositional technique.

CCS Concepts: • **Software and its engineering → Formal software verification;**

Additional Key Words and Phrases: Compositional verification, probabilistic model checking, algorithmic learning

ACM Reference Format:
Fei He, Xiaowei Gao, Miaofei Wang, Bow-Yaw Wang, and Lijun Zhang. 2016. Learning weighted assumptions for compositional verification of Markov decision processes. ACM Trans. Softw. Eng. Methodol. 25, 3, Article 21 (June 2016), 39 pages.
DOI: http://dx.doi.org/10.1145/2907943

1. INTRODUCTION

Probabilistic models are widely used to analyze and verify systems that exhibit "quantified uncertainty," such as security protocols, biological systems, and resilient systems [Baier and Katoen 2008; Rutten et al. 2004]. Typical examples that can be naturally modeled in a stochastic fashion include the message loss in an unreliable communication channel, reactions occurring between large number of molecules, and failure occurring in a long run of the system. Probabilistic models are also used to model and analyze randomized algorithms, which are expected to solve many hard problems more efficiently than any classical algorithms [Motwani and Raghavan

This work was supported in part by the Chinese National 973 Plan (2010CB328003), the NSF of China (61272001, 91218302, 61472473, 61532019, GZ1023), the Tsinghua University Initiative Scientific Research Program, the Ministry of Science and Technology of Taiwan (103-2221-E-001 -020 -MY3), and the CAS/SAFEA International Partnership Program for Creative Research Teams.
Authors' addresses: F. He, X. Gao, and M. Wang, School of Software, Tsinghua University, Beijing, China; emails: hefei@tsinghua.edu.cn, syczgxw@163.com, wmf12@mails.tsinghua.edu.cn; B.-Y. Wang, Academia Sinica, Taiwan; email: bywang@iis.sinica.edu.tw; L. Zhang, State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China; email: zhanglj@ios.ac.cn.

DOI: http://dx.doi.org/10.1145/2907943

1995]. Indeed, the IEEE 802.11 standard has employed randomized methods to avoid the transmission collision in wireless networks [IEEE 2012].

Similar to classical systems, probabilistic systems are not always free of errors. 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. Consider a classical system $M_0 \| M_1$ composed of two concurrent components M_0, M_1, and P, which is an intended property about the system. Consider the assume-guarantee reasoning proof rule for classical systems [Cobleigh et al. 2003]:

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

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 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 [Cobleigh et al. 2003]. In active machine learning [Angluin 1987], 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 [Angluin 1987] suffices to infer classical finite automata as classical assumptions [Cobleigh et al. 2003]. Other techniques have also been developed to find useful assumptions for compositional verification of classical systems [Gupta et al. 2007; Gheorghiu et al. 2007; Chen et al. 2010; He et al. 2014].

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.

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 [Segala and Lynch 1994]. However, learning probabilistic assumptions 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 [Komuravelli et al. 2012b]. 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 [Kwiatkowska et al. 2010; Feng et al. 2010]. 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 (MDPs). Our most ingenious idea is to consider *weighted* assumptions in our new assume-guarantee reasoning proof rule. Compared to the proof rules in Kwiatkowska et al. [2010] and Feng et al. [2010], 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 MDPs. Our assume-guarantee reasoning proof rule is trivially invertible.

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 (MTBDDs) [Fujita et al. 1997; Baier et al. 1997; De Alfaro et al. 2000]. We hence develop an L^*-based learning algorithm for MTBDDs 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 efficient. Note that assumptions oftentimes are not unique. 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. Compositional verification can alleviate the state explosion problem even for probabilistic programs.

This is an extended and revised version of a preliminary conference paper [He et al. 2015]. Compared to He et al. [2015], this article makes following new contributions. First, the logic for verification is extended from $P_{\leq p}[\psi]$ to $P_{\trianglelefteq p}[\psi]$, where $\trianglelefteq \in \{\leq, \geq\}$. The lower-bound probabilistic properties $P_{\geq p}[\psi]$ and the boundary cases with $p = 0$ or 1 are newly supported with the proposed technique in this work. Second, the model checking of 0/1-weighted automatons (WAs) is intensively studied here. We show that the model checking of 0/1-WAs can be achieved with little changes to the probabilistic model checking algorithm for MDPs. Guarantees of this algorithm are discussed. The assume-guarantee reasoning rule is further proved in the presence of this algorithm. Finally, as a further contribution, this article reports a new set of experimental results to evaluate the impact of the learning process (which lies at the core of the learning-based verification framework) to the proposed approach.

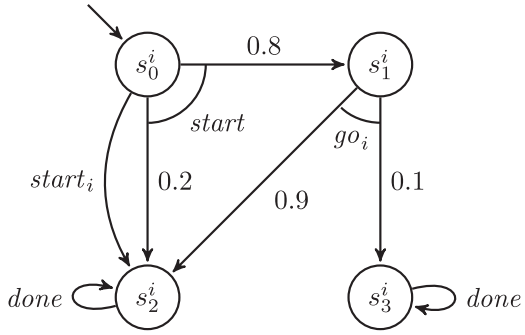


Fig. 1. Node_i.

1.1. Our Contributions

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 MDPs.
—We give an MTBDD learning algorithm under Angluin's active learning model. It uses a polynomial number of queries in the sizes of target MTBDDs 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 MDPs.
—We compare our new technique to the monolithic probabilistic model checker PRISM [Parker 2002]. Experimental results suggest promising outlooks for our compositional technique.

1.2. Outline of the Article

This article is organized as follows. In Section 2, we illustrate our learning-based compositional verification technique with a small example. In Section 3, backgrounds of probabilistic systems and probabilistic model checking are provided. In Section 4, a model checking algorithm for 0/1-WAs is proposed. Section 5 presents our sound and invertible assume-guarantee reasoning proof rule. The MTBDD learning algorithm is described in Section 6. Our learning-based assume-guarantee reasoning framework is given in Section 7. Section 8 reports the experimental results on parameterized test cases. Related works are discussed in Section 9. Finally, Section 10 concludes this article.

2. A MOTIVATING EXAMPLE

Consider the probabilistic system $\mathsf{node}_1 \| \mathsf{node}_2$ composed of two MDP 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 transit 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 therefore gives a

probabilistic distribution over states. For nonshared 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. Thus, 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 specified by the probabilistic computation tree logic (PCTL) 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

It is possible to derive a variant of the proof rule (1) so that a probabilistic assumption is used to establish probabilistic properties on MDPs. To apply the variant in a learning-based assume-guarantee reasoning framework, a learning algorithm is needed to infer probabilistic assumptions. However, an active learning algorithm for MDPs is still missing. The naive probabilistic variant of the proof rule (1) does not help. Consider another variant of the proof rule (1):

$$\frac{\mathsf{node}_1 \preceq_e A \qquad A \| \mathsf{node}_2 \models P_{\leq 0.01}[\psi_{failed}]}{\mathsf{node}_1 \| \mathsf{node}_2 \models P_{\leq 0.01}[\psi_{failed}]}, \tag{2}$$

where A is a weighted assumption. In contrast to probabilistic assumptions, weights associated with transitions in weighted assumptions need not form probabilistic distributions. In out setting, $\mathsf{node}_1 \preceq_e A$ means that every transition of node_1 is also a transition of A. Moreover, each transition of A has a weight not less than the probability of the corresponding transition in node_1. The probabilistic proof rule says the following: to show that $\mathsf{node}_1 \| \mathsf{node}_2$ satisfies a probabilistic property, it suffices to find a weighted assumption A that

—A performs every transition of node_1 with no less probability; and
—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 Section 2.1 [Cobleigh et al. 2003; Chen et al. 2010; Kwiatkowska et al. 2010; He et al. 2014]. 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.

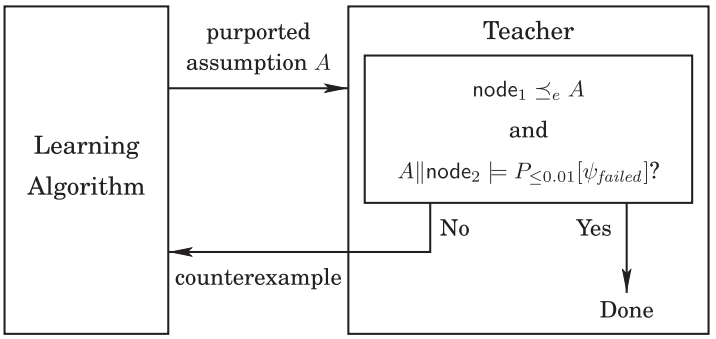


Fig. 2. Overview.

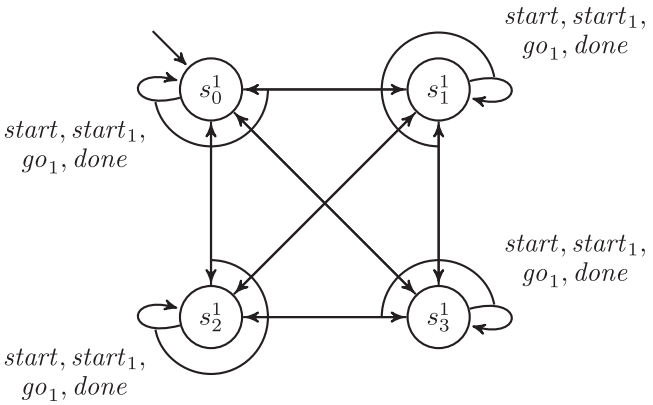


Fig. 3. Weighted assumption A.

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 MDPs, 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, as the sum of weights from the state s_1^1 on the action go_1 is 4.

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

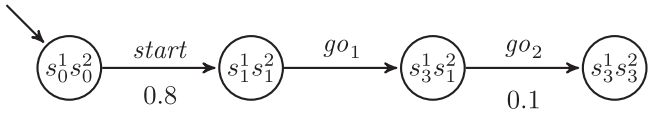


Fig. 4. Witness to $A \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$.

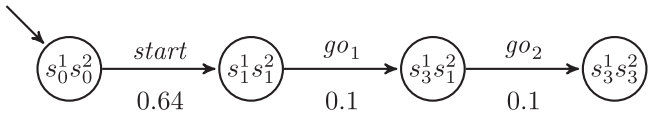


Fig. 5. Corresponding path in $\mathsf{node}_1 \| \mathsf{node}_2$.

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 cannot 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). In addition, recall that the weight associated with a transition in the weighted assumption 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. Hence, the witness to $A \| \mathsf{node}_2 \not\models P_{\leq 0.01}[\psi_{failed}]$ is spurious. The mechanical teacher then should help the learning algorithm revising the weighted assumption by sending a counterexample.

2.6. Selecting Counterexamples

To remove the spurious witness in Figure 4 from the weighted assumption A, the mechanical teacher selects a transition in the weighted assumption A that 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.

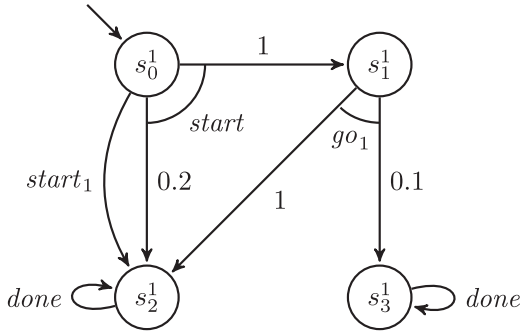


Fig. 6. Weighted assumption A'.

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 that 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 MDPs

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 $|S|$. Denote $dom(\delta)$ the domain of δ. 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 3.1. A *weighted automaton* 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*.

An *infinite path* π^ω in M is an infinite sequence $s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \ldots$ such that $s_0 = \bar{s}$, $\alpha_i \in Act$, and $s_i \xrightarrow{\alpha_i} s_{i+1}$ is a *transition* in M with $T(s_i, \alpha_i)(s_{i+1}) \neq 0$ for all $i \geq 0$. A *finite*

path π is a finite prefix of an infinite path. Denote the set of all infinite and finite paths in M by $Path_M^\omega$ and $Path_M$, respectively. For a path τ that is either finite or infinite, we denote $|\tau|$ the *length* of τ and $\tau[i] = s_i$ the $(i+1)$-th state in τ.

Let $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots \xrightarrow{\alpha_{n-1}} s_n$ be a finite path in M. The *cylinder set* $C(\pi)$ for π is the set of infinite paths with the common finite prefix π—that is,

$$Cyl(\pi) = \{\pi^\omega \mid \pi \text{ is prefix of } \pi^\omega\}.$$

The σ algebra associated with WA M is the smallest σ algebra that contains the cylinder sets for all finite paths of M. The measure on this σ algebra is defined as

$$\mathcal{W}(Cyl(\pi)) = T(s_0, \alpha_0)(s_1) \times T(s_1, \alpha_1)(s_2) \times \cdots \times T(s_{n-1}, \alpha_{n-1})(s_n).$$

The *weight* of the finite path π is defined as

$$Wt(\pi) = T(s_0, \alpha_0)(s_1) \times T(s_1, \alpha_1)(s_2) \times \cdots \times T(s_{n-1}, \alpha_{n-1})(s_n).$$

Note that although $\mathcal{W}(Cyl(\pi)) = Wt(\pi)$, they have different meanings: $\mathcal{W}$ is a measure on infinite paths, whereas Wt refers to finite ones. 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 3.2. A 0/1-*weighted automaton* $M = (S, \bar{s}, Act, T)$ is a WA where $0 \leq T(s, \alpha)(t) \leq 1$ for every $s, t \in S$ and $\alpha \in Act$.

In other words, a 0/1-WA is a WA on which all weights associated are within the domain [0,1].

A WA is nondeterministic. There may be multiple transitions between two states on different actions. Adversaries are used to resolve nondeterministic choices in WAs [Baier and Katoen 2008]. Let S^+ denote a nonempty sequence of states in S, and let $Act(s)$ denote 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 exist, 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 3.3. A *Markov decision process* $M = (S, \bar{s}, Act, T)$ 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 MDPs are probability distributions, the weight associated with each transition in MDPs is referred to as probability.

Example 3.4. 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

$$T_{\mathsf{node}_1}(s, \alpha) = \begin{cases} \langle 0,\ 0.8,\ 0.2,\ 0\rangle & \text{if } s = s_0^1, \alpha = start \\ \langle 0,\ 0,\ 1,\ 0\rangle & \text{if } s = s_0^1, \alpha = start_1 \\ \langle 0,\ 0,\ 0.9,\ 0.1\rangle & \text{if } s = s_1^1, \alpha = go_1 \\ \langle 0,\ 0,\ 1,\ 0\rangle & \text{if } s = s_2^1, \alpha = done \\ \langle 0,\ 0,\ 0,\ 1\rangle & \text{if } s = s_3^1, \alpha = done \\ \langle 0,\ 0,\ 0,\ 0\rangle & \text{otherwise.} \end{cases}$$

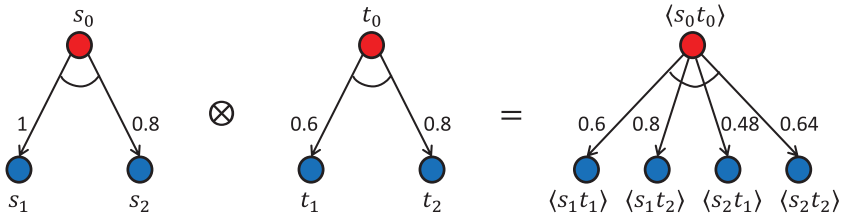


Fig. 7. Composition of two weighted functions.

In contrast, A in Figure 3 is a 0/1-WA where $S_A = S_{\text{node}_1}$, $\bar{s}_A = \bar{s}_{\text{node}_1}$, $Act_A = Act_{\text{node}_1}$, and $\forall s \in S_A, \forall \alpha \in Act_A : T_A(s, \alpha) = \langle 1, 1, 1, 1 \rangle$.

3.2. Parallel Composition

Let S_i be a finite set and $\delta_i \in \Delta(S_i)$ for $i = 0, 1$. Define $(\delta_0 \otimes \delta_1)(s_0, s_1) = \delta_0(s_0) \times \delta_1(s_1)$ for any $s_0 \in S_0$, $s_1 \in S_1$. Figure 7 gives an example for composing two weighted functions. Observe that $\delta_0 \otimes \delta_1 \in \Delta(S_0 \times S_1)$, and the following facts hold:

—If $dom(\delta_i) \subseteq [0, 1]$ for $i = 0, 1$, then $dom(\delta_0 \otimes \delta_1) \subseteq [0, 1]$.
—If $\delta_i \in \Delta^D(S_i)$ for $i = 0, 1$, then $\delta_0 \otimes \delta_1 \in \Delta^D(S_0 \times S_1)$.
—If $\delta_0 = \mathbf{0}$ or $\delta_1 = \mathbf{0}$, then $\delta_0 \otimes \delta_1 = \mathbf{0}$.

Definition 3.5. Let $M_i = (S_i, \bar{s}_i, Act_i, T_i)$ be a 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 MDPs yields an MDP.

3.3. Logic

Fix a finite set AP of atomic propositions. We focus on probabilistic safety properties specified by PCTL [Hansson and Jonsson 1994; Bianco and de Alfaro 1995]. PCTL in general allows nested probabilistic operators [Katoen et al. 2014]. In this article, we consider a fragment, known as conditional reachability probability, which is in the form of $P_{\trianglelefteq p}[\psi]$ with $p \in [0, 1]$ being a probability, $\trianglelefteq \in \{\leq, \geq\}$, and

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

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

Qualitative properties. For the properties with $p = 0$ or 1, consider following facts:

$$P_{\geq 0}[\psi] \equiv true, \qquad P_{\leq 1}[\psi] \equiv true,$$
$$P_{\leq 0}[\psi] \equiv \neg P_{>0}[\psi], \qquad P_{\geq 1}[\psi] \equiv P_{=1}[\psi].$$

Let $\psi = \phi_1 \,\mathtt{U}\, \phi_2$. The probabilistic model checking of qualitative properties of the form $P_{>0}[\psi]$ or $P_{=1}[\psi]$ can be reduced to the classical model checking of corresponding CTL properties [Baier and Katoen 2008].

In the remainder of the article, we assume that $p \neq 0, 1$. We call properties in the form of $P_{\leq p}[\psi]$ the upper-bound probabilistic properties and properties in the form of $P_{\geq p}[\psi]$ the lower-bound probabilistic properties, respectively. We abuse notations and consider all of them as probabilistic safety properties.[1]

3.4. Probabilistic Model Checking of MDPs

Given an MDP $M = (S, \bar{s}, Act, T)$ and a probabilistic safety property $P_{\trianglelefteq p}[\psi]$ with $\psi = \phi_1 \,\mathtt{U}\, \phi_2$, define $p^{\sigma}_{M,s}(\psi) = Wt(\Pi)$, where

$$\Pi = \{\pi \in Path_{M^{\sigma}} \mid \forall i = 0..|\pi| - 2.\ (\pi[i] \models \phi_1 \wedge \neg\phi_2) \wedge (\pi[|\pi - 1|] \models \phi_2)\}.$$

Observe that Π is prefix containment free and $Wt(\Pi)$ is the probability of reaching ϕ_2 states along ϕ_1 states under the adversary σ. Let

$$p^{max}_{M,s}(\psi) = \max_{\sigma \in Adv_M} p^{\sigma}_{M,s}(\psi),$$
$$p^{min}_{M,s}(\psi) = \min_{\sigma \in Adv_M} p^{\sigma}_{M,s}(\psi)$$

denote the *maximal* and *minimal* probability that ψ is satisfied at s over all adversaries, respectively. In the remainder of the article, we use $p^{ext}_{M,s}(\psi)$ with $ext \in \{max, min\}$ to represent either $p^{max}_{M,s}(\psi)$ or $p^{min}_{M,s}(\psi)$. We omit the subscript M and the formula ψ in $p^{\sigma}_{M,s}(\psi)$, $p^{ext}_{M,s}(\psi)$ when they are clear from the context.

Definition 3.6. Given a state s of an MDP M and a probabilistic safety property $P_{\trianglelefteq p}[\psi]$ with $\trianglelefteq \in \{\leq, \geq\}$, we say that s *satisfies* $P_{\trianglelefteq p}[\psi]$ (written $M, s \models P_{\trianglelefteq p}[\psi]$) if $p^{ext}_{M,s}(\psi) \trianglelefteq p$, where ext is max if $\trianglelefteq$ is $\leq$ and min otherwise. We say that M *satisfies* $P_{\trianglelefteq p}[\psi]$ (written $M \models P_{\trianglelefteq p}[\psi]$) if $M, \bar{s} \models P_{\trianglelefteq p}[\psi]$.

The probabilistic model checking of MDPs has been proposed in Bianco and de Alfaro [1995] and Parker [2002]. Let $\psi = \phi_1 \mathtt{U} \phi_2$. The probability $p^{ext}_{s}(\psi)$ can be approximated by an iterative algorithm [Baier and Katoen 2008]. The computation starts from the states satisfying ϕ_2 and iterates backward to compute the extreme (maximum or minimum) probability of reaching these states from the states satisfying ϕ_1. More precisely, define

$$p^{ext}_{s,i}(\psi) = \begin{cases} 1, & \text{if } s \models \phi_2 \\ 0, & \text{if } s \not\models \phi_2 \wedge s \not\models \phi_1 \\ 0, & \text{if } s \not\models \phi_2 \wedge s \models \phi_1 \wedge i = 0 \\ ext_{\alpha}\left\{\sum_{t \in S} T(s, \alpha)(t) \times p^{ext}_{t,i-1}(\psi)\right\}, & \text{otherwise,} \end{cases} \tag{3}$$

where ext is max if $\trianglelefteq$ is $\leq$ and min otherwise. The computation iterates until $p^{ext}_{s,i}(\psi)$ converges. Then $p^{ext}_{s}(\psi) = \lim_{i \to \infty} p^{ext}_{s,i}(\psi)$.

[1]Strictly speaking, a low-bound probabilistic property is not a safety property.

Table I. Values of $p^{max}_{s,i}$ Computed in Example 3.7

	$p^{max}_{s_0^1,i}$	$p^{max}_{s_1^1,i}$	$p^{max}_{s_2^1,i}$	$p^{max}_{s_3^1,i}$
$i=0$	0	0	0	1
$i=1$	0	0.1	0	1
$i=2$	0.08	0.1	0	1
$i=3$	0.08	0.1	0	1

Example 3.7. Assume that we want to verify $P_{\leq 0.1}[true\,\mathtt{U}\,\langle s_3^1\rangle]$ on the process node_1.[2] The values of $p^{max}_{s,i}$ for $s = s_0^1, s_1^1, s_2^1, s_3^1$ and $i = 0, 1, 2, 3$ are computed according to Equation (3). The results are listed in Table I. For example, $p^{max}_{s_3^1,0} = 1$ since $s_3^1 \models \langle s_3^1\rangle$,

$$\begin{aligned} p^{max}_{s_1^1,1} &= T(s_1^1, go_1)(s_2^1) \times p^{max}_{s_2^1,0} + T(s_1^1, go_1)(s_3^1) \times p^{max}_{s_3^1,0} \\ &= 0.9 \times 0 + 0.1 \times 1 \\ &= 0.1, \\ p^{max}_{s_0^1,2} &= \max\left\{T(s_0^1, start)(s_1^1) \times p^{max}_{s_1^1,1} + T(s_0^1, start)(s_2^1) \times p^{max}_{s_2^1,1}, T(s_0^1, start_1)(s_2^1) \times p^{max}_{s_2^1,1}\right\} \\ &= \max\{0.8 \times 0.1 + 0.2 \times 0, 1 \times 0\} \\ &= \max\{0.08, 0\} \\ &= 0.08. \end{aligned}$$

In the 3-th iteration, the computation converges, and $p^{max}_{s_0^1} = 0.08 \leq 0.1$. Thus, we conclude that $\mathsf{node}_1 \models P_{\leq 0.1}[true\,\mathtt{U}\,\langle s_3^1\rangle]$.

3.5. Counterexamples

A *weighted witness* [Han et al. 2009; Wimmer et al. 2012, 2013] 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 (by the first two conditions). Hence, $Wt(c)$ is well defined.

As to the weighted witness to $M \not\models P_{\geq p}[\psi]$, note that [Baier and Katoen 2008]

$$P_{\geq p}[\phi_1\,\mathtt{U}\,\phi_2] \equiv P_{\leq 1-p}[(\phi_1 \wedge \neg\phi_2)\,\mathtt{W}\,(\neg\phi_1 \wedge \neg\phi_2)],$$

where $\mathtt{W}$ is the "weak until" operation. Using the technique in Han et al. [2009], the witness generation for this kind of property can be reduced to the case with upper-bound probability properties.

Without loss of generality, we assume that the weighted witness to a probabilistic safety property $P_{\trianglelefteq p}[\psi]$ is a pair (σ, c). 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 3.8. Consider the weighted witness (σ, c) shown in Figure 4, where $\sigma(\langle s_0^1 s_0^2\rangle) = start$, $\sigma(\langle s_0^1 s_0^2\rangle\langle s_1^1 s_1^2\rangle) = go_1$, $\sigma(\langle s_0^1 s_0^2\rangle\langle s_1^1 s_1^2\rangle\langle s_3^1 s_1^2\rangle) = go_2$, and $c = \{\langle s_0^1 s_0^2\rangle \xrightarrow{start} \langle s_1^1 s_1^2\rangle \xrightarrow{go_1} \langle s_3^1 s_1^2\rangle \xrightarrow{go_2} \langle s_3^1 s_3^2\rangle\}$. The weight of c is $0.8 \times 1 \times 0.1 = 0.08$.

[2] In the motivating example in Section 2, we verify a property on the composition of $\mathsf{node}_1 \| \mathsf{node}_2$.

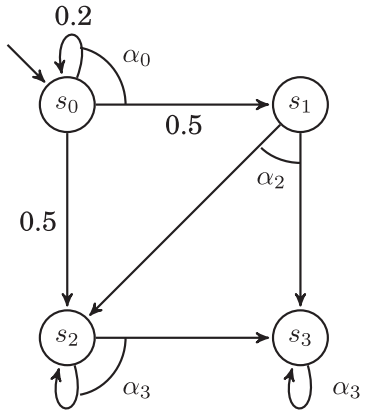


Fig. 8. A 0/1-WA example.

Table II. Values of $w^{min}_{s,i}$ and $w^{max}_{s,i}$ Computed in Example 4.2: One Value Shown if $w^{min}_{s,i} = w^{max}_{s,i}$

	$w^{ext}_{s_0,i}$	$w^{ext}_{s_1,i}$	$w^{ext}_{s_2,i}$	$w^{ext}_{s_3,i}$
$i=0$	0	0	0	1
$i=1$	0	1	1	1
$i=2$	0.5	2	2	1
$i=3$	1.1 / 1	3	3	1
$i=4$	1.72 / 1.5	4	4	1

4. MODEL CHECKING OF A 0/1-WA

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

Let M be a 0/1-WA and $P_{\trianglelefteq p}[\psi]$ a probabilistic safety property. For any state s, denote $w^{ext}_{M,s}(\psi)$ (with $ext \in \{max, min\}$) the *extreme* (i.e., *maximal* or *minimal*) weight that ψ is satisfied at s. The $w^{ext}_{M,s}(\psi)$ is defined similarly as for MDPs, except that $w^{ext}_{M,s}(\psi)$ are now referred to as weights rather than probabilities. Note here that we again omit the subscript M and the formula ψ in $w^{ext}_{M,s}(\psi)$ when they are clear from the context.

Definition 4.1. Given a state s of a 0/1-WA M and a probabilistic safety property $P_{\trianglelefteq p}[\psi]$ with $\trianglelefteq \in \{\leq, \geq\}$, we say that s *satisfies* $P_{\trianglelefteq p}[\psi]$ (written $M, s \models P_{\trianglelefteq p}[\psi]$) if $w^{ext}_s(\psi) \trianglelefteq p$, where ext is max if $\trianglelefteq$ is $\leq$ and min otherwise. We say that M *satisfies* $P_{\trianglelefteq p}[\psi]$ (written $M \models P_{\trianglelefteq p}[\psi]$) if $M, \bar{s} \models P_{\trianglelefteq p}[\psi]$.

4.1. Iterative Computation for 0/1-WAs

The iterative computation procedure for MDPs cannot be applied to 0/1-WAs. A notable difference is that the iterative computation may not converge here.

Example 4.2. Consider verifying $P_{\leq 0.5}[true\ \mathtt{U}\ \langle s_3 \rangle]$ on the 0/1-WA in Figure 8. Note that the weight is not shown in the figure when it is 1. Using the iterative computation described in Section 3.4, the computed results of $w^{ext}_{s,i}$ for $s = s_0, s_1, s_2, s_3$ and $i = 0, 1, 2, 3, 4$ are listed in Table II. Note that only one value is shown if $w^{min}_{s,i} = w^{max}_{s,i}$. Observe that $s_3 \models \langle s_3 \rangle$, and thus the values of $w^{min}_{s_3,i}$ and $w^{max}_{s_3,i}$ for any i are 1's (the last column in Table II). Other values are iteratively updated according to (3). For example,

Table III. Values of $\underline{w}^{min}_{s,i}$ and $\underline{w}^{min}_{s,i}$ Computed in Example 4.3: One Value Shown if $\underline{w}^{min}_{s,i} = \underline{w}^{max}_{s,i}$

	$\underline{w}^{ext}_{s_0,i}$	$\underline{w}^{ext}_{s_1,i}$	$\underline{w}^{ext}_{s_2,i}$	$\underline{w}^{ext}_{s_3,i}$
$i=0$	0	0	0	1
$i=1$	0	1	1	1
$i=2$	0.5	1	1	1
$i=3$	0.6 / 0.5	1	1	1
$i=4$	0.62 / 0.5	1	1	1

$$w^{max}_{s_0,4} = \max\left\{0.2 \times w^{max}_{s_0,3} + 0.5 \times w^{max}_{s_1,3}, 0.5 \times w^{max}_{s_2,3}\right\} = 1.72,$$
$$w^{min}_{s_0,4} = \min\left\{0.2 \times w^{min}_{s_0,3} + 0.5 \times w^{min}_{s_1,3}, 0.5 \times w^{min}_{s_2,3}\right\} = 1.5.$$

Apparently, the values of $w^{ext}_{s_0,i}$, $w^{ext}_{s_1,i}$, and $w^{ext}_{s_2,i}$ are all unbounded when i increases. Thus, the computations of $w^{ext}_{s_0}$, $w^{ext}_{s_1}$, and $w^{ext}_{s_2}$ do not converge.

We therefore need to adapt the iterative computation for MDPs to 0/1-WAs. The purpose is to avoid divergent computation. Let s be a state of a 0/1-WA M and $P_{\trianglelefteq p}[\psi]$ a probabilistic safety property with $\trianglelefteq \in \{\leq, \geq\}$. Note the following:

(1) if $w^{max}_{s,i}(\psi) \geq 1$, then $w^{max}_{s,i}(\psi) > p$, and hence $M, s \not\models P_{\leq p}[\psi]$; and
(2) if $w^{min}_{s,i}(\psi) \geq 1$, then $w^{min}_{s,i}(\psi) > p$, and hence $M, s \models P_{\geq p}[\psi]$.

In conclusion, the satisfiability of $P_{\trianglelefteq p}[\psi]$ on the state s of M is known if only $w^{ext}_{s,i} \geq 1$. Here *ext* is *max* if $\trianglelefteq$ is $\leq$ and *min* otherwise.

We thus need not to consider the exact values of $w^{ext}_{s,i}$ whenever it exceeds 1. Instead of computing $w^{max}_{s}(\psi)$, which is oftentimes unbounded, we propose to compute the truncated value of $w^{max}_{s}(\psi)$. More specifically, we define

$$\underline{w}^{ext}_{s,i}(\psi) = \begin{cases} 1, & \text{if } s \models \phi_2 \\ 0, & \text{if } s \not\models \phi_2 \wedge s \not\models \phi_1 \\ 0, & \text{if } s \not\models \phi_2 \wedge s \models \phi_1 \wedge i = 0 \\ \min\{1, ext_\alpha\{\sum_{t \in S} T(s,\alpha)(t) \times \underline{w}^{ext}_{t,i-1}(\psi)\}\}, & \text{otherwise .} \end{cases} \tag{4}$$

Observe that the only difference between (3) and (4) lies in the last row, where $\underline{w}^{ext}_{s,i}(\psi)$ truncates the value of $w^{ext}_{s,i}(\psi)$ when the latter exceeds 1.

Apparently, the computation of $\underline{w}^{ext}_{s,i}$ always converges. In the worst case, it converges to 1. Let $\underline{w}^{ext}_{s}(\psi) = \lim_{i \to \infty} \underline{w}^{ext}_{s,i}(\psi)$; we have $\underline{w}^{ext}_{s}(\psi) \leq 1$ and $\underline{w}^{ext}_{s}(\psi) \leq w^{ext}_{s}(\psi)$.

Example 4.3. Consider the model and property in Example 4.2 again. Now we are using Equation (4) to compute $\underline{w}^{ext}_{s,i}$. The computed results are listed in Table III. Observe that the truncation operation does take effect in computing $\underline{w}^{ext}_{s_1,i}$ and $\underline{w}^{ext}_{s_2,i}$ when $i > 1$, and leads their values to 1's. And with these truncated weights, $\underline{w}^{ext}_{s_0,i}$ is computed. For example,

$$\underline{w}^{max}_{s_0,4} = \min\left\{1, \max\left\{0.2 \times \underline{w}^{max}_{s_0,3} + 0.5 \times \underline{w}^{max}_{s_1,3}, 0.5 \times \underline{w}^{max}_{s_2,3}\right\}\right\} = 0.62,$$
$$\underline{w}^{min}_{s_0,4} = \min\left\{1, \min\left\{0.2 \times \underline{w}^{min}_{s_0,3} + 0.5 \times \underline{w}^{min}_{s_1,3}, 0.5 \times \underline{w}^{min}_{s_2,3}\right\}\right\} = 0.5.$$

If the computation continues, we can find that $\underline{w}^{max}_{s_0} = \lim_{i \to \infty} \underline{w}^{max}_{s,i} = 0.625$, $\underline{w}^{min}_{s_0} = \lim_{i \to \infty} \underline{w}^{min}_{s,i} = 0.5$. Thus, $M \not\models_a P_{\leq 0.5}[\mathit{true}\, \mathtt{U}\, \langle s_3 \rangle]$.

Algorithm 1 gives the iterative algorithm for computing $\underline{w}_s^{ext}(\psi)$ in a 0/1-WA. The algorithm takes as inputs a 0/1-WA and a probabilistic safety property, and outputs a vector W, which records the computed weights $\underline{w}_s^{ext}$ for each state s, where ext is max if the property is of the form $P_{\leq p}[\psi]$ and min otherwise. For simplicity, we use $W[s]$ to denote the stored weight in W with respect to the state s. Given a state formula ϕ, let $SAT(\phi)$ denote the set of states satisfying ϕ. The weights in W are initialized to 1 for all states in $SAT(\phi_2)$ and 0 for all other states. Then it iteratively updates weights in W using Equation (4) for states $s \not\models \phi_2 \wedge s \models \phi_1$, until the terminating condition is achieved. The terminating condition (where ϵ is a predefined small value) represents that the vector distance between W and W' is sufficiently small, and thus the computation is close enough to converge.[3] We remark that this algorithm is simplified to demonstrate the iterative computation procedure. In the real implementation, the algorithm is realized on top of MTBDD.

ALGORITHM 1: Iterative Computation Procedure for 0/1-WA

Input: $M = (S, \bar{s}, Act, T)$: a 0/1-WA; and $P_{\trianglelefteq p}[\phi_1 \,\mathtt{U}\, \phi_2]$: a probabilistic safety property.
Output: W: a vector that records w_s^{ext} for each state $s \in S$.

```
S1 ← SAT(φ1);
S2 ← SAT(φ2);
for s ∈ S2 do W[s] ← 1 for s ∉ S2 do W[s] ← 0 repeat
    W' ← W;
    for s ∈ S1\S2 do
        update W[s] using Equation (4);
    end
until |(W − W')/W| ≤ ε;
return W;
```

4.2. Guarantees from the Algorithm

Definition 4.4. Given a state s of a 0/1-WA M and a probabilistic safety property $P_{\trianglelefteq p}[\psi]$ with $\trianglelefteq \in \{\leq, \geq\}$, we say that $M, s \models_a P_{\trianglelefteq p}[\psi]$ if $\underline{w}_s^{ext}(\psi) \trianglelefteq p$, where ext is max if $\trianglelefteq$ is $\leq$ and min otherwise. We say that $M \models_a P_{\trianglelefteq p}[\psi]$ if $M, \bar{s} \models_a P_{\trianglelefteq p}[\psi]$.

Note that $M \models_a P_{\trianglelefteq p}[\psi]$ and $M \models P_{\trianglelefteq p}[\psi]$ are not equivalent. The correspondences between $\models$ and $\models_a$ are summarized in the following lemma.

LEMMA 4.5. *Given a state s of a 0/1-WA M and a probabilistic safety property $P_{\trianglelefteq p}[\psi]$ with $\trianglelefteq \in \{\leq, \geq\}$ and $p \in (0, 1)$,*

(1) $M, s \models P_{\leq p}[\psi]$ implies that $M, s \models_a P_{\leq p}[\psi]$, but the reverse may not hold.
(2) $M, s \models_a P_{\geq p}[\psi]$ implies that $M, s \models P_{\geq p}[\psi]$, but the reverse may not hold.

PROOF. (1) By $M, s \models P_{\leq p}[\psi]$, we know that $w_s^{max} \leq p$ (Definition 4.1). Note that $\underline{w}_s^{max} \leq w_s^{max}$, and thus $\underline{w}_s^{max} \leq p$. Therefore, $M, s \models_a P_{\leq p}[\psi]$ (Definition 4.4). To show that the reverse does not hold, consider Examples 4.2 and 4.3, where $M, s_0 \models_a P_{\leq 0.7}[true \,\mathtt{U}\, \langle s_3 \rangle]$ (since $\underline{w}_{s_0}^{max} = 0.625 < 0.7$), but $M, s_0 \not\models P_{\leq 0.7}[true \,\mathtt{U}\, \langle s_3 \rangle]$ (since $w_{s_0,3}^{max} = 1.1 > 0.7$).

(2) By $M, s \models_a P_{\geq p}[\psi]$, we know that $\underline{w}_s^{min} \geq p$ (Definition 4.4). Note that $\underline{w}_s^{min} \leq w_s^{min}$, and thus $w_s^{min} \geq p$. Therefore, $M, s \models P_{\geq p}[\psi]$ (Definition 4.1). To show the reverse does

[3]This procedure may never stop if we set $W = W'$ as the terminating condition.

not hold, consider again Examples 4.2 and 4.3, where $M, s_0 \models P_{\geq 0.7}[true\ \mathtt{U}\ \langle s_3 \rangle]$ (since $w^{min}_{s_0}$ is infinitely large), but $M, s_0 \not\models_a P_{\geq 0.7}[true\ \mathtt{U}\ \langle s_3 \rangle]$ (since $\underline{w}^{min}_{s_0} = 0.5 < 0.7$). □

Considering the special case that M is an MDP, the following lemma holds.

LEMMA 4.6. *Given a state s of an MDP M and a probabilistic safety property $P_{\trianglelefteq p}[\psi]$ with $\trianglelefteq \in \{\leq, \geq\}$, $M, s \models P_{\trianglelefteq p}[\psi]$ if and only if $M, s \models_a P_{\trianglelefteq p}[\psi]$.*

PROOF. Since M is an MDP, $p^{ext}_{s,i}(\psi) \in [0, 1]$ for any state s and iteration i [Parker 2002]. No truncation is needed in the computation of $\underline{w}^{ext}_s$. Thus, $\underline{w}^{ext}_s = w^{ext}_s$, and the conclusion follows. □

5. ASSUME-GUARANTEE REASONING WITH WA

Assume-guarantee reasoning proof rules for probabilistic systems are proposed in Kwiatkowsa et al. [2010] and Feng et al. [2010]. Those proof rules replace a probabilistic component in a composition with a classical assumption. Since classical assumptions cannot 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 WA. We begin with the weighted extension of the classical simulation relation.

Definition 5.1. Let $M = (S, \bar{s}, Act, T)$ and $M' = (S', \bar{s}', Act', T')$ be WAs; we say that M is *embedded* in M' (written $M \preceq_e M'$, or equivalently $M' \succeq_e M$) 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 $\alpha \in Act$.

For example, consider node_1 (in Figure 1) and A (in Figure 3). Apparently, $\mathsf{node}_1 \preceq_e A$ since $S_{\mathsf{node}_1} = S_A$, $\bar{s}_{\mathsf{node}_1} = \bar{s}_A$, $Act_{\mathsf{node}_1} = Act_A$, and for any $s, t \in S_A$, $\alpha \in Act_A$, $T_{\mathsf{node}_1}(s, \alpha)(t) \leq T_A(s, \alpha)(t) = 1$.

For simplicity, we use $\trianglelefteq_e$ to denote either $\preceq_e$ or $\succeq_e$. The following lemma says that the operator $\trianglelefteq_e$ is compositional and preserves the satisfiability of probabilistic safety properties.

LEMMA 5.2. *Let M, M', N be 0/1-WAs, $\trianglelefteq_e$ an operator in $\{\preceq_e, \succeq_e\}$, and $P_{\trianglelefteq p}[\psi]$ a probabilistic safety property with $\trianglelefteq \in \{\leq, \geq\}$. Thus,*

(1) $M \trianglelefteq_e M'$ implies that $M \| N \trianglelefteq_e M' \| N$, and
(2) $M \trianglelefteq_e M'$ and $M' \models P_{\trianglelefteq p}[\psi]$ imply that $M \models P_{\trianglelefteq p}[\psi]$, where $\trianglelefteq_e$ is $\preceq_e$ if $\trianglelefteq$ is $\leq$ and $\succeq_e$ otherwise.

PROOF. (1) 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 0/1-WAs. We consider the case of $\trianglelefteq_e = \preceq_e$; the other case can be proved similarly. 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 $T(s, \alpha)(t) \leq T'(s, \alpha)(t)$ (by $M \preceq_e M'$) and $T_N(p, \alpha)(q) \geq 0$ (by Definition 3.2), $T(s, \alpha)(t) \times T_N(p, \alpha)(q) \leq T'(s, \alpha)(t) \times T_N(p, \alpha)(q)$ 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 3.5).

(2) We consider the case of $\trianglelefteq_e = \preceq_e$ and $\trianglelefteq = \leq$; the other case can be proved similarly. By $M \preceq_e M'$, we know that $T(s, \alpha)(t) \leq T'(s, \alpha)(t)$. Hence, $w^{max}_{M,\bar{s}}(\psi) \leq w^{max}_{M',\bar{s}}(\psi)$. By $M' \models P_{\leq p}[\psi]$, we have $w^{max}_{M',\bar{s}}(\psi) \leq p$. Hence, $w^{max}_{M,\bar{s}}(\psi) \leq w^{max}_{M',\bar{s}}(\psi) \leq p$. Thus, $M \models P_{\leq p}[\psi]$. □

We are now capable to define the assume-guarantee reasoning rule for MDPs. A rule is *sound* if its conclusion holds when its premises are fulfilled. A rule is *invertible* if its premises can be fulfilled when its conclusion holds. Note that a weighted assumption is introduced into the rule.

THEOREM 5.3. *Let $M_i = (S_i, \bar{s}_i, Act_i, T_i)$ be MDPs for $i = 0, 1$ and $P_{\trianglelefteq p}[\psi]$ a probabilistic safety property with $\trianglelefteq \in \{\leq, \geq\}$ and $p \in (0, 1)$. The following proof rules are sound and invertible:*

$$\frac{M_0 \trianglelefteq_e A \qquad A \| M_1 \models P_{\trianglelefteq p}[\psi]}{M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]}, \tag{5}$$

where $A = (S_A, \bar{s}_A, Act_A, T_A)$ is a 0/1-WA, $\trianglelefteq_e \in \{\preceq_e, \succeq_e\}$, and $\trianglelefteq_e$ is $\preceq_e$ if $\trianglelefteq$ is $\leq$ and $\succeq_e$ otherwise.

PROOF. Soundness of the proof rule follows from Lemma 5.2. By $M_0 \trianglelefteq_e A$, $M_0 \| M_1 \trianglelefteq_e A \| M_1$ (Lemma 5.2 (1)). Since $A \| M_1 \models P_{\trianglelefteq p}[\psi]$, we have $M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]$ (Lemma 5.2 (2)). Invertibility is trivial since M_0 is also a 0/1-WA. When the conclusion holds, M_0 itself is a weighted assumption. The two premises are trivially fulfilled. □

5.1. The Reasoning Rule with $\models_a$ Relation

In this section, we show that the assume-guarantee reasoning rule (Theorem 5.3) also holds in case of the $\models_a$ relation.

Lemma 5.2 (2) says that the $\trianglelefteq_e$ operator preserves the probabilistic safety property along the $\models$ relation. A similar lemma can be established in case of the $\models_a$ relation, by additionally requiring one of the components to be an MDP.

LEMMA 5.4. *Let M be an MDP, M' a 0/1-WA, and $P_{\trianglelefteq p}[\psi]$ a probabilistic safety property with $\trianglelefteq \in \{\leq, \geq\}$ and $p \in (0, 1)$. Thus,*

(1) $M \preceq_e M'$ and $M' \models_a P_{\leq p}[\psi]$ imply that $M \models P_{\leq p}[\psi]$, and
(2) $M \succeq_e M'$ and $M' \models_a P_{\geq p}[\psi]$ imply that $M \models P_{\geq p}[\psi]$.

PROOF. (1) We prove by induction that $p^{max}_{M,s,i}[\psi] \leq \underline{w}^{max}_{M',s,i}[\psi]$ for any i. When $i = 0$, $p^{max}_{M,s,0}[\psi] = \underline{w}^{max}_{M',s,0}[\psi]$ apparently holds. Assume that $p^{max}_{M,s,k}[\psi] \leq \underline{w}^{max}_{M',s,k}[\psi]$; we now prove that $p^{max}_{M,s,k+1}[\psi] \leq \underline{w}^{max}_{M',s,k+1}[\psi]$ also holds. By $M \preceq_e M'$, we know that $T(s,\alpha)(t) \leq T'(s,\alpha)(t)$ for any state t and any action α. Thus, $p^{max}_{M,s,k+1}[\psi] = \max_\alpha\{\sum_{t\in S} T(s,\alpha)(t) \times p^{max}_{M,t,k}(\psi)\} \leq \max_\alpha\{\sum_{t\in S} T'(s,\alpha)(t) \times \underline{w}^{max}_{M',t,k}(\psi)\}$. For M being an MDP, $p^{max}_{M,s,k+1}[\psi] \leq 1$. Thus, $p^{max}_{M,s,k+1}[\psi] \leq \min\{1, \max_\alpha\{\sum_{t\in S} T'(s,\alpha)(t) \times \underline{w}^{max}_{M',t,k}(\psi)\}\} = \underline{w}^{max}_{M',s,k+1}[\psi]$. Therefore, by induction on the value of i, $p^{max}_{M,s,i}[\psi] \leq \underline{w}^{max}_{M',s,i}[\psi]$ holds for any i. Thus, $p^{max}_{M,\bar{s}}[\psi] \leq \underline{w}^{max}_{M',\bar{s}}[\psi]$. By $M' \models_a P_{\leq p}[\psi]$, $\underline{w}^{max}_{M',\bar{s}}[\psi] \leq p < 1$. Hence, $p^{max}_{M,\bar{s}} \leq p$, and thus $M \models P_{\leq p}[\psi]$.

(2) Since $M' \models_a P_{\geq p}[\psi]$, $M' \models P_{\geq p}[\psi]$ (Lemma 4.5). $M \models P_{\geq p}[\psi]$ follows from $M' \models P_{\geq p}[\psi]$ and $M \succeq_e M'$ (Lemma 5.2 (2)). □

THEOREM 5.5. *Let $M_i = (S_i, \bar{s}_i, Act_i, T_i)$ be MDPs for $i = 0, 1$ and $P_{\trianglelefteq p}[\psi]$ a probabilistic safety property with $\trianglelefteq \in \{\leq, \geq\}$ and $p \in (0, 1)$. The following proof rule is sound and invertible:*

$$\frac{M_0 \trianglelefteq_e A \qquad A \| M_1 \models_a P_{\trianglelefteq p}[\psi]}{M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]},$$

where $A = (S_A, \bar{s}_A, Act_A, T_A)$ is a 0/1-WA, $\trianglelefteq_e \in \{\preceq_e, \succeq_e\}$, and $\trianglelefteq_e$ is $\preceq_e$ if $\trianglelefteq$ is $\leq$ and $\succeq_e$ otherwise.

PROOF. Soundness follows from Lemma 5.2 (1) and Lemma 5.4. Invertibility is trivial since M_0 is also a 0/1-WA. □

6. LEARNING 0/1-WA

We adopt the learning-based framework [Cobleigh et al. 2003; Feng et al. 2010] to generate an assumption A in the assume-guarantee reasoning proof rule (Theorem 5.3). To

apply our new proof rule, a weighted assumption is needed. One could employ learning algorithms that infer explicit quantitative models like multiplicity automata [Beimel et al. 2000]. Those learning algorithms require complex and accurate matrix operations. Hence, they 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 MTBDDs to represent weighted assumptions implicitly. To infer implicitly represented weighted assumptions, we then develop an MTBDD learning algorithm under Angluin's learning model.

6.1. Multiterminal 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 $\mathbf{x}$.

An MTBDD [Fujita et al. 1997] 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 *nonterminal* 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 MTBDDs 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 MTBDDs [Kwiatkowska et al. 2011]. Let $M = (S, \bar{s}, Act, T)$ be a WA. Without loss of generality, we assume that $|S| = 2^n$ 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.[4] Let ν, ν' be valuations of $\mathbf{x}, \mathbf{x}'$ and α a valuation of $\mathbf{z}$. The action valuation α of $\mathbf{z}$ 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.

Given a WA $M = (\mathbf{x}, l_M(\mathbf{x}), \mathbf{z}, f_M(\mathbf{z}, \mathbf{x}, \mathbf{x}'))$, let ν, ν' be two state valuations and α a valid action valuation. If M is a 0/1-WA, $0 \leq f_M(\alpha, \nu, \nu') \leq 1$ (Definition 3.2); if M is an MDP, $\sum_{\nu'} f_M(\alpha, \nu, \nu') \in \{0, 1\}$ (Definition 3.3).

[4]This encoding scheme for actions is beneficial to the variable ordering. Refer to Parker [2002] for details.

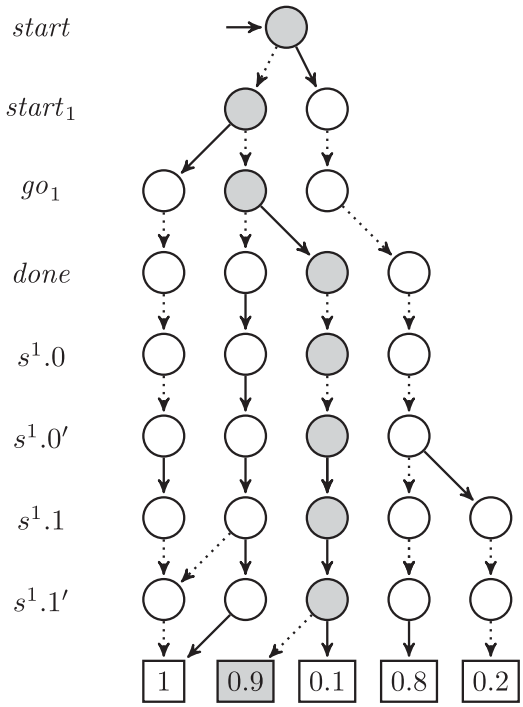


Fig. 9. MTBDD of f_{node_1}.

Example 6.1. Consider the process node_1 in Figure 1 where the states are s_j^1 for $j = 0, \ldots, 3$, 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. In particular, the valuations $\langle 0, 0\rangle$, $\langle 0, 1\rangle$, $\langle 1, 0\rangle$, and $\langle 1, 1\rangle$ of $\mathbf{x}$ represent the states s_0^1, s_1^1, s_2^1, and s_3^1, respectively. Note that valuations of $\mathbf{z}$ need to be valid. 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 $\mathbf{z}$ represent the action $start, start_1, go_1$, and $done$, respectively. The MTBDD of $f_{\mathsf{node}_1}(\mathbf{z}, \mathbf{x}, \mathbf{x}')$ is shown in Figure 9. In the figure, the terminal node labeled by the number 0 and its incoming edges are not shown. Solid edges are labeled by 1, and dotted edges are labeled by 0. For example, the shaded path in Figure 1 corresponds to the valuations of $\mathbf{x} = \langle 0, 1\rangle$, $\mathbf{x}' = \langle 1, 0\rangle$, and $\mathbf{z} = \langle 0, 0, 1, 0\rangle$. It thus represents the transition from s_1^1 to s_2^1 on the action go_1. The rational number labeling the terminal node of this path (i.e., 0.9) indicates the probability of this transition. □

Using MTBDDs, Theorem 5.3 is rephrased as follows.

COROLLARY 6.2. *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 MDPs for* $i = 0, 1$ *and* $P_{\trianglelefteq p}[\psi]$ *a probabilistic safety property with* $\trianglelefteq \{\leq, \geq\}$ *and* $p \in (0, 1)$. *Then*

$$\frac{M_0 \trianglelefteq_e A \qquad A \| M_1 \models P_{\trianglelefteq p}[\psi]}{M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]},$$

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, and* $\trianglelefteq_e$ *is* $\preceq_e$ *if* $\trianglelefteq$ *is* $\leq$ *and* $\succeq_e$ *otherwise.*

6.2. The L^* Learning Algorithm for Regular Languages

We adapt the L^* algorithm to infer an MTBDD representing a weighted assumption [Angluin 1987]. 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 two queries to the teacher:

—A membership query asks if a string belongs to the target language.
—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 [Angluin 1987] that the L^* algorithm always infers the minimal deterministic finite automaton recognizing any target regular language within a polynomial number of queries.

6.3. An MTBDD Learning Algorithm

Since any 0/1-WA can be represented by MTBDDs, 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 two types of queries:

—On a *membership query* $MQ(v)$ with a valuation v of $\mathbf{x}$, the teacher answers $f(v)$.
—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 v of $\mathbf{x}$ with $f(v) \neq g(v)$ 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 two 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 therefore can be used to infer a finite automaton recognizing the language R [Angluin 1987]. 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 [Kimura and Clarke 1990], the L^* algorithm can be modified to infer MTBDDs with two terminal nodes [Gavaldà and Guijarro 1995].

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^*. In other words, $\cup_i L_i = \Sigma^*$ 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 that 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 therefore *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 6.3. *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.*

To learn general MTBDDs, 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 $\Sigma^* \backslash \Sigma^{|\mathbf{x}|}$, an MTBDD in fact gives a partition of Σ^*. In other words, an MTBDD defines a k-language. By Theorem 6.3, 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.

6.3.1. Algorithm. Our L^* learning-based algorithm for MTBDD's is shown in Algorithm 2. Let S be a finite prefix-closed set of strings over $\mathbb{B}$, and let E be a finite suffix-closed set of strings over $\mathbb{B}$. Denote λ as the empty string. We maintain an observation table T from $(S \cup S\mathbb{B}) \times E$ to $\mathcal{F}$. Similar to the L^* algorithm, the observation table T can be seen as a matrix with $|S \cup S\mathbb{B}|$ rows and $|E|$ columns. For $s \in S$ and $e \in E$, the entry $T(s, e)$ denotes the acceptance type $F \in \mathcal{F}$ of an unknown finite automaton. When $\mathcal{F} = \mathbb{B}$, we obtain the observation table in the L^* algorithm for regular languages.

The rows with indices from S correspond to states in the unknown finite automaton; the rows with indices from $S\mathbb{B}$ give the transition function of the unknown finite automaton. Let $s \in S \cup S\mathbb{B}$. We write $T(s, \bullet)$ for the row with index s. Closeness and consistency of an observation table in the L^* algorithm generalize straightforwardly. An observation table T is *closed* if there is an $s \in S$ with $T(s, \bullet) = T(t, \bullet)$ for each $t \in S\mathbb{B}$. An observation table T is *consistent* if $T(sb, \bullet) = T(s'b, \bullet)$ for each $b \in \mathbb{B}$ whenever $T(s, \bullet) = T(s', \bullet)$. A closed and consistent observation table induces a finite automaton with outcomes U. Our modified L^* algorithm in fact infers a minimal finite automaton with outcomes U by the generalized Myhill-Nerode theorem [Chen et al. 2009].

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 MTBDDs 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. However, the MTBDD teacher accepts MTBDDs as conjectures. To solve these problems, we apply the techniques in Gavaldà and Guijarro [1995].

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, with each state at distance less than $|\mathbf{x}|$ into a nonterminal node labeled with variable x_i and each state at distance $|\mathbf{x}|$ into a terminal node.

ALGORITHM 2: L^*-Based Learning Algorithm for MTBDD's

$S, E \leftarrow \{\lambda\}$;
ask membership queries for λ and each $b \in \mathbb{B}$;
initialize observation table (S, E, T);
repeat
  while (S, E, T) *is not closed* **do**
    find $s \in S$ and $b \in \mathbb{B}$ such that $T(sb, \bullet) \neq T(s', \bullet)$ for all $s' \in S$;
    $S \leftarrow S \cup \{sb\}$;
    extend T using membership queries;
  end
  construct candidate DFA C from (S, E, T);
  transform C into an MTBDD g;
  ask equivalence query $EQ(g)$;
  if *the teacher answers with a counterexample* v **then**
    let t be the corresponding binary string of v;
    find a suffix e of t that witnesses the counterexample;
    $E \leftarrow E \cup \{e\}$;
    extend T using membership queries;
  end
until *the teacher answers YES*;

6.3.2. Time Analysis.

THEOREM 6.4. *Let $f(\mathbf{x})$ be a target MTBDD. The MTBDD learning algorithm outputs f in polynomial time, using $O(|f_A|^2 + |f_A| \log |\mathbf{x}|)$ membership queries and at most $|f_A|$ equivalence queries, where $|f_A|$ is the size of the automaton before reduced to the target MTBDD.*

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 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 $O(|f_A|^2 + |f_A| \log |\mathbf{x}|)$ membership and $|f_A|$ equivalence queries. □

7. THE LEARNING-BASED VERIFICATION FRAMEWORK

With our new assume-guarantee reasoning proof rule (Section 5) and learning algorithm for MTBDDs (Section 6), 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 MDPs $(i = 0, 1)$, and let $P_{\trianglelefteq p}[\psi]$ be a probabilistic safety property. To verify if $M_0 || M_1 \models P_{\trianglelefteq p}[\psi]$ 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 $M_0 \trianglelefteq_e A$ and $A || M_1 \models P_{\trianglelefteq p}[\psi]$. 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 design a mechanical teacher to answer queries from the learning algorithm (Figure 10).

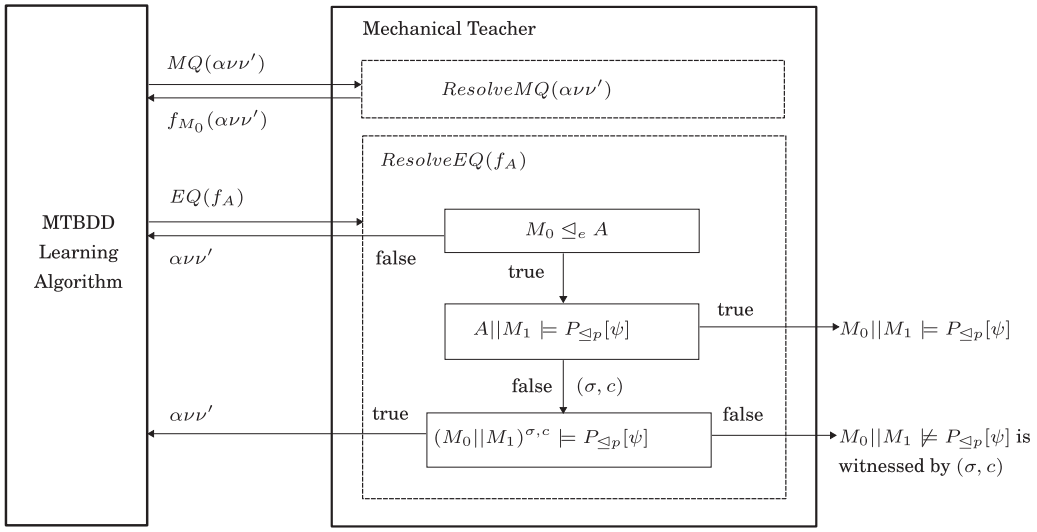


Fig. 10. Learning framework for compositional verification.

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 therefore attain completeness. In practice, it often finds useful weighted assumptions before M_0 is inferred.

7.1. Resolving Membership Queries

Our membership query resolution algorithm (Algorithm 3) targets the weighted transition function of M_0. Clearly, M_0 embeds itself and therefore can be used as a weighted assumption. On the membership query $MQ(\alpha\nu\nu')$, the mechanical teacher simply returns $f_{M_0}(\alpha\nu\nu')$.

ALGORITHM 3: *ResolveMQ*$(\alpha\nu\nu')$

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

7.2. 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 6.2 hold.

The equivalence query resolution algorithm first checks if $M_0 \trianglelefteq_e A$. If not, there are valuations α, ν, and ν' with $f_{M_0}(\alpha\nu\nu') \rhd f_A(\alpha\nu\nu')$.[5] The equivalence query resolution algorithm returns $\alpha\nu\nu'$ as a counterexample to $EQ(f_A)$.

If $M_0 \trianglelefteq_e A$, the equivalence query resolution algorithm continues to check whether $A \| M_1 \models P_{\trianglelefteq p}[\psi]$ holds by model checking. If $A \| M_1 \models P_{\trianglelefteq p}[\psi]$ holds, the MTBDD learning algorithm has inferred a weighted assumption that establishes $M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]$ by the assume-guarantee reasoning proof rule in Corollary 6.2. Otherwise, the equivalence query resolution algorithm obtains a weighted witness (σ, c) to $A \| M_1 \not\models P_{\trianglelefteq p}[\psi]$ 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 $M_0 \| M_1 \trianglelefteq_e A \| M_1$. The (σ, c)-fragment $(M_0 \| M_1)^{\sigma,c}$ is well defined. If $(M_0 \| M_1)^{\sigma,c} \models P_{\trianglelefteq p}[\psi]$, 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 $(M_0 \| M_1) \not\models P_{\trianglelefteq p}[\psi]$ with the weighted witness (σ, c) (Algorithm 4).

Example 7.1. 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 thus satisfies ψ_{failed}. However, its weight is $0.64 \times 0.1 \times 0.1 = 0.0064 \leq 0.01$. Therefore, $(\mathsf{node}_1 \| \mathsf{node}_2)^{\sigma,c} \models P_{\leq 0.01}[\psi_{failed}]$. The weighted witness in Figure 4 is spurious.

ALGORITHM 4: *ResolveEQ*(f_A)

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 $\exists \alpha\nu\nu'. f_A(\alpha\nu\nu') \rhd f_{M_0}(\alpha\nu\nu')$ **then**
  answer $EQ(f_A)$ with the counterexample $\alpha\nu\nu'$;
  receive a new equivalence query $EQ(f_{A'})$;
  call *ResolveEQ*$(f_{A'})$;
if $A||M_1 \models P_{\trianglelefteq p}[\psi]$ **then**
  answer $EQ(f_A)$ with *YES*;
  return "$M_0||M_1 \models P_{\trianglelefteq p}[\psi]$";
else
  let (σ, c) be a weighted witnessto $A||M_1 \not\models P_{\trianglelefteq p}[\psi]$;
  if $(M_0 \| M_1)^{\sigma,c} \models P_{\trianglelefteq p}[\psi]$ **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 "$M_0||M_1 \not\models P_{\trianglelefteq p}[\psi]$" with (σ, c);
  end
end

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 $(A \| M_1)^{\sigma,c} \not\models P_{\trianglelefteq p}[\psi]$ and $(M_0 \| M_1)^{\sigma,c} \models P_{\trianglelefteq p}[\psi]$. Recall that $(M_0 \| M_1)^{\sigma,c}$ and $(A \| M_1)^{\sigma,c}$ have the same state set, initial state,

[5] $\rhd$ is $>$ if $\trianglelefteq_e$ is $\preceq_e$ and $<$ otherwise.

and alphabet. The only differences between $(M_0 \| M_1)^{\sigma,c}$ and $(A \| M_1)^{\sigma,c}$ are the weights associated with transitions. To remove the spurious weighted witness (σ, c), we would like to select transitions that 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 $\Pi_0(t)$ respectively be the sets of paths in $(A \| M_1)^{\sigma,c}$ and $(M_0 \| M_1)^{\sigma,c}$ that 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 MTBDDs for efficiency.

Additionally, observe 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. To reduce the number of model checking invocations, we reuse the same spurious weighted witness to compute counterexamples [He et al. 2014]. 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.

7.3. Correctness

The correctness of our assume-guarantee reasoning framework for probabilistic systems follows from Theorem 5.3. We establish the soundness, completeness, and termination of the new learning-based framework in the remainder of this section.

THEOREM 7.2 (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 MDPs for* $i = 0, 1$, $P_{\trianglelefteq p}[\psi]$ *a probabilistic safety property, and* $f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}_0')$ *an MTBDD.*

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

PROOF. When our learning-based framework reports $M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]$ in Algorithm 4, 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 $M_0 \trianglelefteq_e A$ and $A \| M_1 \models P_{\trianglelefteq p}[\psi]$ has been inferred. By the soundness of the assume-guarantee reasoning proof rule (Theorem 5.3), $M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]$. On the other hand, suppose that our learning-based framework reports $M_0 \| M_1 \not\models P_{\trianglelefteq p}[\psi]$. The weighted witness (σ, c) to $A \| M_1 \not\models P_{\trianglelefteq p}[\psi]$ has been verified to be a witness to $M_0 \| M_1 \not\models P_{\trianglelefteq p}[\psi]$. □

THEOREM 7.3 (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 MDPs for* $i = 0, 1$, *and let* $P_{\trianglelefteq p}[\psi]$ *be a probabilistic safety property:*

—*If* $M_0 || M_1 \models P_{\trianglelefteq p}[\psi]$, *then ResolveEQ*$(f_A)$ *returns* $M_0 || M_1 \models P_{\trianglelefteq p}[\psi]$ *for some MTBDD* $f_A(\mathbf{z}, \mathbf{x}_0, \mathbf{x}_0')$.
—*If* $M_0 || M_1 \not\models P_{\trianglelefteq p}[\psi]$, *then ResolveEQ*$(f_A)$ *returns* $M_0 || M_1 \not\models P_{\trianglelefteq p}[\psi]$ *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 6.4). If $M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]$, the learning algorithm always infers a weighted assumption A (in the worst case, A is M_0) such that $M_0 \trianglelefteq_e A$ and $A \| M_1 \models P_{\trianglelefteq p}[\psi]$. Hence, *ResolveEQ*$(f_A)$

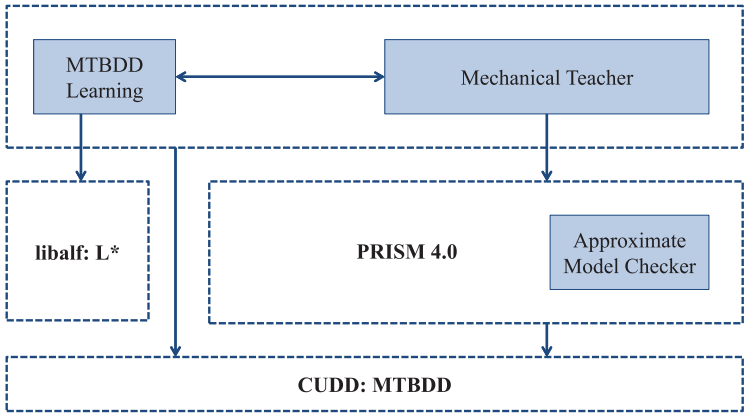


Fig. 11. Implementation.

returns $M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]$. Otherwise, the learning algorithm always infers a weighted assumption A (in the worst case, A is M_0) such that $M_0 \| M_1 \not\models P_{\trianglelefteq p}[\psi]$ is witnessed by (σ, c). *ResolveEQ*(f_A) returns $M_0 \| M_1 \not\models P_{\trianglelefteq p}[\psi]$. The completeness of our assume-guarantee reasoning framework follows from the preceding observations. □

THEOREM 7.4 (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 MDPs for* $i = 0, 1$, *and let* $P_{\trianglelefteq p}[\psi]$ *be a probabilistic safety property. Our learning-based framework reports* $M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]$ *or* $M_0 \| M_1 \not\models P_{\trianglelefteq p}[\psi]$ *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 6.4). At this point, the weighted assumption A is M_0. The mechanical teacher reports either $M_0 \| M_1 \models P_{\trianglelefteq p}[\psi]$ or $M_0 \| M_1 \not\models P_{\trianglelefteq p}[\psi]$. □

8. EVALUATIONS

We have implemented a prototype of our compositional verification technique on top of PRISM 4.0.1 [Parker 2002]. It accepts an MDP specified in the PRISM modeling language and a probabilistic safety property. The architecture of our prototype is shown in Figure 11. Approximate model checking of 0/1-WAs (Algorithm 1) is implemented by revising the probabilistic model checking algorithm for MDPs in PRISM. The MTBDD learning algorithm is implemented by modifying the L^* algorithm in the libalf 0.3 library [Bollig et al. 2010] with the CUDD 2.5.0 package.[6] The membership query resolution algorithm (Algorithm 3) and the embedded checking algorithm are implemented using CUDD. The equivalence query resolution algorithm (Algorithm 4) is implemented by making calls to the embedded checker and the approximate model checker. We generate counterexamples by the techniques in Han et al. [2009]. All experiments were run on a virtual machine with a 2.6GHz CPU and 4GB RAM.

We conducted four experiments to evaluate our compositional approach. The first and second experiments compare the performance of our compositional approach (*compositional*) with the monolithic model checking algorithms in PRISM. PRISM with the MTBDD engine (PRISM-M) and PRISM with the hybrid engine (PRISM-H) are tested

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

in sequence. The third experiment tests the impact of the probability bounds to the effectiveness of our approach. The fourth experiment tests the impact of the learning strategy to our compositional approach. A new version of our compositional approach (*compositional*$^-$) that disables the learning process is used for comparison.

All experiments are conducted on several parameterized examples: the randomized consensus coin protocol, wireless LAN, FireWire root contention protocol, and randomized dining philosophers. All examples are derived from the PRISM Web site.[7] Each example model contains several interacting processes. 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 [Zhu et al. 2009; Cobleigh et al. 2008]. Here we employ a simple heuristic: choose the process with the minimal interface alphabet. The *interface alphabet* of a process is the set of shared actions. For example, the wireless LAN 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.

8.1. Compositional Versus Monolithic (MTBDD Based)

The first experiment compares the performance of our compositional approach (*compositional*) with the MTBDD-based monolithic model checking in PRISM (PRISM-M). Note that our approach is also MTBDD based.

Experimental results are listed in Tables IV through VII. For each model and a corresponding probabilistic safety property $P_{\trianglelefteq p}[\psi]$, we compute the property of $P_{ext=?}[\psi]$ by PRISM and report it in the P_{ext} column, where *ext* is *max* if $\trianglelefteq$ is $\leq$ and *min* otherwise. Note that the property $P_{\trianglelefteq p}[\psi]$ holds on the model if and only if $p \rhd P_{ext}$. The satisfiability of the property $P_{\trianglelefteq p}[\psi]$ on the model is reported in the *Result* column. For each test case, we show the model size (*Size*) and runtime (*Time*). The model size counts the number of MTBDD nodes for the weighted transition function of the composed model.[8] The runtime includes the time spent on all stages, including model construction, model checking, witness analysis, and assumption learning. For the compositional approach, the number of model checker calls (#*MC*) is also reported. The last column (*Reduction*) shows the reduction of model size and time of our compositional approach to PRISM-M. All times are in seconds, sizes are in 10^3 nodes, and the symbol "–" indicates either time-out (4 hours) or memory-out (4GB).

Randomized consensus coin protocol. In the randomized consensus coin protocol (for short, *Consensus*), a set of N distributed processes access a global shared counter [Aspnes and Herlihy 1990]. Each process periodically increases or decreases the value of the shared counter depending on the outcome of a random coin flip. The boundary value that the counter can take is parameterized by K. To ensure that all processes agree on the outcome of some decision, we verify the property "the probability that eventually all processes make a decision but some two processes do not agree on the same value is at most 0.01."

Experimental results are given in Table IV, where the first column lists the values of the parameters (N, K). The results show that the compositional verification outperforms the monolithic probabilistic model checking significantly with more than two processes. Our compositional approach improves the verification time by 39.3% on average and reduces the model size by 3.2% on average. If there are only two processes, the size of the original model is so small that compositional verification is redundant.

[7] http://www.prismmodelchecker.org

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

Table IV. Results on *Consensus*: Time in Seconds, Size in 10^3 Nodes

Param	P_{max}	Result	PRISM-M		Compositional			Reduction (%)	
			Time	*Size*	*#MC*	*Time*	*Size*	*Time*	*Size*
(2, 6)	0.04	False	3.9	0.4	6	6.1	0.4	–57.5	–7.6
(2, 8)	0.03	False	9.0	0.4	6	13.7	0.5	–51.8	–10.6
(2, 10)	0.02	False	17.4	0.4	6	24.9	0.5	–43.3	–10.3
(4, 2)	0.29	False	40.2	2.3	6	35.8	2.2	11.0	3.4
(4, 4)	0.16	False	315.1	2.3	6	241.1	2.3	23.5	2.8
(4, 6)	0.10	False	1,109.7	2.3	6	703.5	2.3	36.6	3.0
(4, 8)	0.08	False	2,522.1	2.4	6	1,627.4	2.3	35.5	2.3
(4, 10)	0.06	False	4,747.2	2.4	6	3,051.7	2.3	35.7	2.4
(6, 2)	0.36	False	1,611.2	7.1	6	1,068.1	6.8	33.7	4.6
(6, 4)	0.19	False	10,270.3	7.1	6	5,751.4	6.7	44.0	4.6
All			20,645.9	27.1		12,523.6	26.2	39.3	3.2

Wireless LAN protocol. The wireless local area networks protocol (for short, *WLAN*) is specified in IEEE 802.11 standard [IEEE 2012], which enable the use of heterogeneous communication devices within the same network. 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 (with four components) is considered. We verify the property "the probability that any station's backoff counter hits the number K within the time limit T is at most 0.1."

Experimental results with $T = 2{,}000$ and $T = 1{,}000$ are given in Table V(a) and (b), respectively. The parameters (B, K) are listed in the first column. Experimental results show that the compositional verification outperforms the monolithic probabilistic model checking consistently on failed properties. When the property is verified to be false, the improvement of model size and verification time are consistent: the improvement of verification time varies from 98.5% to 99.8% in the first set of experiments and varies from 97.2% to 99.5% in the second set of experiments, and the improvement of model size varies from 89.1% to 96.1% in the first set of experiments and varies from 88.7% to 95.9% in the second set of experiments. If a property holds, the compositional verification does not reduce the model size but still improves the verification time in four out of six cases.

FireWire root contention protocol. The IEEE 1394 FireWire root contention protocol (for short, *FireWire*) is a root election protocol for connected networks [Kwiatkowska et al. 2003]. 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 (with five components) is considered. We verify the property "the probability that a root node is elected eventually before some time deadline passes is at most 0.1."

Experimental results in Table VI show that the compositional verification outperforms the monolithic probabilistic model checking in all experiments. The improvement of model size and verification time are stable. The verification time is improved by 95.9%, and the model size is reduced by 89.1% on average.

Randomized dining philosophers. This model (for short, *Philos*) gives a randomized solution to the dining philosophers problem [Lehmann and Rabin 1981]. N philosophers sit around a circular table. Neighboring philosophers share a resource. A philosopher can eat if he obtains the resources from both sides. We verify the property "the

Table V. Results on *WLAN*: Time in Seconds, Size in 10^3 Nodes

(a) Time Limit $T = 2,000$

Param	P_{max}	Result	PRISM-M		Compositional			Reduction (%)	
			Time	*Size*	*#MC*	*Time*	*Size*	*Time*	*Size*
(2,1)	1.00	False	360.6	304	4	3.8	33	99.0	89.1
(2,2)	0.18	False	381.4	304	4	5.9	33	98.5	89.1
(3,1)	1.00	False	652.2	627	4	3.8	41	99.4	93.5
(3,2)	0.18	False	675.1	627	4	5.9	41	99.1	93.5
(3,3)	0.02	True	687.0	627	11	977.9	665	–42.3	–6.1
(4,1)	1.00	False	2,481.3	1,321	4	5.1	52	99.8	96.1
(4,2)	0.18	False	2,620.3	1,321	4	7.6	52	99.7	96.1
(4,3)	0.02	True	2,530.0	1,321	11	1,776.1	1,421	29.8	–7.6
(4,4)	0.00	True	2,684.0	1,321	11	2,061.7	1,421	23.2	–7.6
Cases		False	7,170.9	4,505	24	32.2	253	99.6	94.4
All			13,072.0	7,774	57	4,847.9	3,760	62.9	51.6

(b) Time Limit $T = 1,000$

Param	P_{max}	Result	PRISM-M		Compositional			Reduction (%)	
			Time	*Size*	*#MC*	*Time*	*Size*	*Time*	*Size*
(2,1)	1.00	False	177.1	295	4	3.1	33	98.3	88.7
(2,2)	0.18	False	178.4	295	4	4.9	33	97.2	88.7
(3,1)	1.00	False	307.8	604	4	3.4	41	98.9	93.2
(3,2)	0.18	False	313.4	604	4	5.4	41	98.3	93.2
(3,3)	0.02	True	324.5	604	9	358.5	604	–10.5	–0.1
(4,1)	1.00	False	876.6	1,271	4	4.8	52	99.5	95.9
(4,2)	0.18	False	886.5	1,271	4	7.0	52	99.2	95.9
(4,3)	0.02	True	901.4	1,271	9	635.8	1,272	29.5	0.0
(4,4)	0.00	True	961.7	1,271	9	757.2	1,272	21.3	0.0
Cases		False	2,739.7	4,341	24	28.5	252	99.0	94.2
All			4,927.3	7,487	51	1,780.0	3,400	63.9	54.6

Table VI. Results on *FireWire*: Time in Seconds, Size in 10^3 Nodes

Param	P_{max}	Result	PRISM-M		Compositional			Reduction (%)	
			Time	*Size*	*#MC*	*Time*	*Size*	*Time*	*Size*
200	1.00	False	86.0	837	4	37.1	130	56.9	84.5
300	1.00	False	293.2	1,196	4	38.5	130	86.9	89.2
400	1.00	False	507.6	1,221	4	38.9	130	92.3	89.4
500	1.00	False	718.5	1,222	4	39.3	130	94.5	89.4
600	1.00	False	954.3	1,244	4	39.2	130	95.9	89.6
700	1.00	False	1,160.9	1,244	4	39.2	130	96.6	89.6
800	1.00	False	1,380.2	1,244	4	39.2	130	97.2	89.6
900	1.00	False	1,590.2	1,244	4	39.2	130	97.5	89.6
1,000	1.00	False	1,803.4	1,244	4	39.5	130	97.8	89.6
All			8,494.4	10,693	36	350.1	1,168	95.9	89.1

probability that neighboring philosophers do not obtain their shared resource simultaneously is at most 0.01."

Experimental results in Table VII show that the compositional verification outperforms the monolithic probabilistic model checking consistently. The verification time is improved by 89.7% on average and the model size is reduced by 67.4% on average. The monolithic probabilistic model checking fails to verify the property if the number

Table VII. Results on *Philos*: Time in Seconds, Size in 10^3 Nodes

Param	P_{max}	Result	PRISM-M		Compositional			Reduction (%)	
			Time	*Size*	*#MC*	*Time*	*Size*	*Time*	*Size*
10	0.00	True	2.4	15	1	1.2	5	52.1	65.7
15	0.00	True	11.5	33	1	5.0	11	56.3	66.6
20	0.00	True	29.7	59	1	13.4	19	54.9	67.0
25	0.00	True	56.4	91	1	31.5	30	44.1	67.3
30	0.00	True	108.8	131	1	54.7	43	49.7	67.4
35	0.00	True	1,022.6	178	1	96.5	58	90.6	67.5
40	0.00	True	2,348.2	233	1	168.1	75	92.8	67.6
45	0.00	True	—	—	1	926.2	95	—	—
All ($N \leq 40$)			3,579.7	740	7	370.4	241	89.7	67.4

of philosophers reaches 45 due to the out-of-memory error, whereas the compositional approach verifies the instance successfully.

Summary. The results are quite encouraging. In 40 of 45 cases, the compositional verifier outperforms PRISM-M significantly. Moreover, a reduction of 90% in time is achieved in 21 cases, and a reduction of 80% in model sizes is attained in 21 cases. Our compositional approach benefits the verification by avoiding the construction of the whole model. As shown in the size reduction column of the preceding tables, 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.

Note that for the satisfied cases in the *WLAN* example, the compositional approach does not reduce the model size and the ratio of saved time is not as significant as for the unsatisfiable cases. One possible reason is that it usually takes more equivalence queries for the learning algorithm to prove both premises of the reasoning rule than to disprove those. Observe, however, that for all satisfied cases in the *Philos* example, the compositional approach outperforms PRISM-M significantly. We consider its reason as the large gap between the actual probability P_{max} and the probability bound p. In later experiments, we would test the impact of this gap on the effectiveness of our learning-based framework.

8.2. Compositional Versus Monolithic (Hybrid Based)

The second experiment compares our compositional approach against PRISM-H. The same set of examples are used to test both approaches. Note that PRISM-H consumes an equal size of MTBDDs as PRISM-M, whereas the latter has already been reported in the first experiment. We compare only runtimes in this experiment.

The results are plotted in Figure 12, where each point represents one benchmark case, the x-axis represents the time in seconds taken by our compositional approach, and the y-axis stands for the time in seconds taken by PRISM-H. For simplicity, we do not distinguish the benchmark cases of *WLAN* models with different time limits, and we represent them using the same type of points. A time value of 3,600 indicates a time-out after 3,600 seconds. Note that the x-axis is logarithmic and the y-axis is linear. Points above the parabola (representing $y = x$) indicate an advantage for our compositional approach.

The comparison results heavily depend on the examples. Similar phenomenons were also reported in Kwiatkowska et al. [2004b]. We observe in Figure 12 that PRISM-H wins in the *Consensus* example, whereas our compositional approach wins in all other examples. Especially, in the *WLAN* and *FireWire* examples, PRISM-H runs out of memory quickly when the model size becomes large. We also observe 16 time-outs ($\geq$3600 seconds) for PRISM-H and only 1 for our compositional approach.

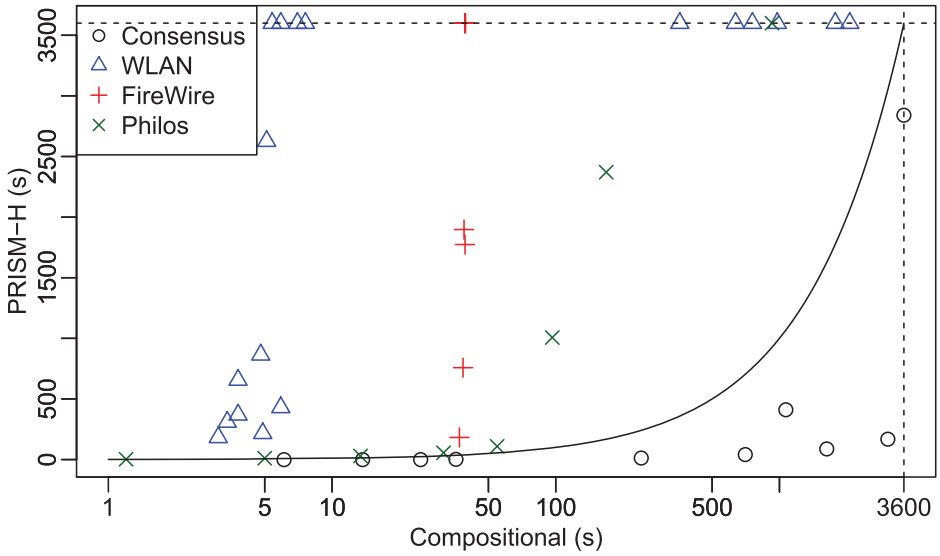


Fig. 12. Runtimes of the compositional approach and PRISM-H on all examples.

8.3. Impact of the Probability Bound

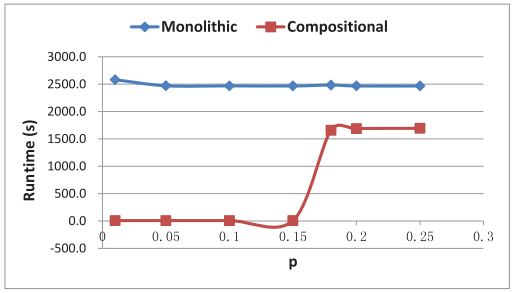
The third experiment evaluates the impact of the probability bound p on the effectiveness of compositional verification. This experiment is performed on examples with different probability bounds.

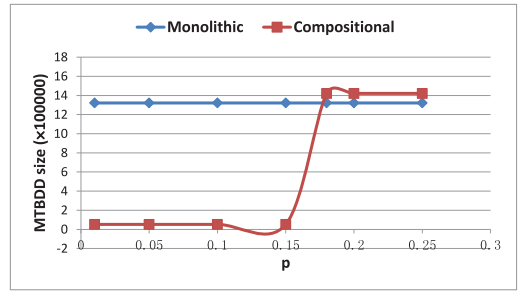
The results on the *WLAN* example with $(B, K) = (4, 2)$ are plotted in Figure 13(a). When p is above or nearly above P_{max} (≈ 0.1836), the performance of our approach goes down quickly. The reason is that the property becomes satisfied when $p \geq P_{max}$. More equivalence queries are then required 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 13(b)). The result from the *FireWire* example with *deadline* $= 400$ (Figure 13(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 13(c) as in other figures. In the *Philos* example, the probability P_{max} is 0, and therefore the properties are always satisfied for any p. Similar to *FireWire*, we do not observe any rising edge in Figure 13(d). However, compositional verification always outperforms the monolithic algorithm regardless of satisfiability of properties in both examples.

8.4. Impact of the Learning Process

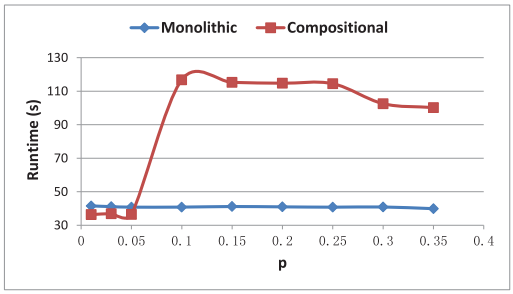
The fourth experiment tests the impact of the learning process to that of our compositional approach. In the learning-based verification framework, each time the mechanical teacher finds a counterexample, a learning algorithm is employed to refine the previous assumption. In this experiment, we disable the learning process and use a more direct method to refine the assumptions.

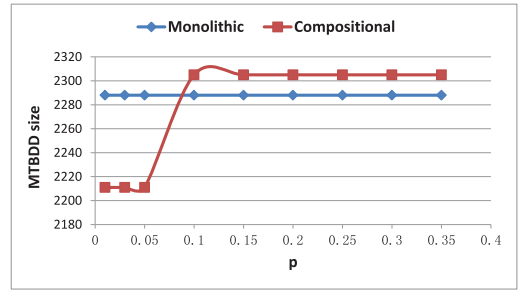
Let A_i be the current weighted assumption, and let t be the counterexample returned by the mechanical teacher. Recall that t is a single transition in A_i selected by the mechanical teacher (Section 7.2). Instead of passing t to the learning algorithm, we may directly revise the weight of t to that of the corresponding transition in M_0. We call



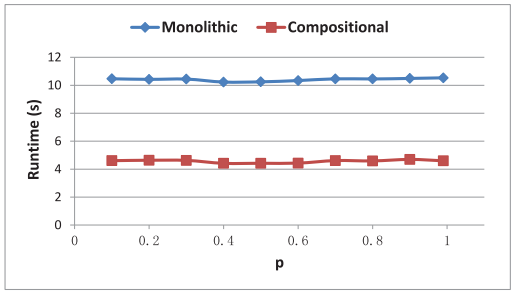


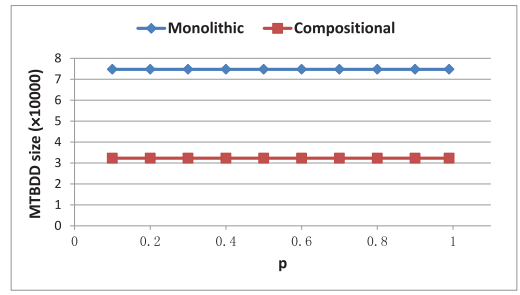
(a) WLAN ($B = 4, K = 2$)



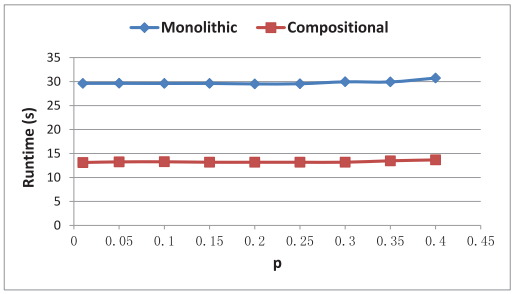


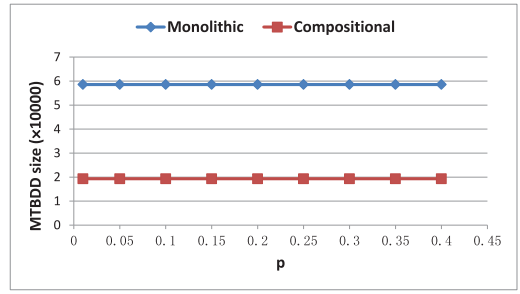
(b) Consensus ($N = 4, K = 2$)





(c) FireWire ($deadline = 400$)





(d) Philos ($N = 20$)

Fig. 13. Impacts of p on the performance of MTBDD-based approaches (red for compositional, blue for PRISM).

the revision A_{i+1}. Then A_{i+1} is presented to the mechanical teacher, and the teacher will check if A_{i+1} is a proper assumption (Algorithm 4). If not, the teacher returns one transition of A_{i+1} as the counterexample and the preceding refinement applies again. Apparently, A_{i+1} has *one* more identical transition to M_0 than A_i. Recall that M_0 contains a finite number of transitions; this procedure always terminates. In the worst case, this approach will find M_0 as the weighted assumption. We call this approach the *compositional approach without learning*, denoted as *compositional*$^-$.

Two types of initializations are realized for *compositional*$^-$:

(1) *true*-initialization[9]: The initial assumption has a transition on any action between any two states, with the weight of 1. For example, the weighted assumption in Figure 3 is the *true*-initialization of node_1.
(2) M_0-initialization: The initial assumption has a transition if M_0 does, with the weight of 1. For example, revising all weights of automaton in Figure 1 to 1's gives the M_0-initialization of node_1.

Experimental results are listed in Table VIII. For each test case, we show the model size (*Size*), runtime (*Time*), and required number of model checker calls (#*MC*) by *compositional*$^-$ along with each of the preceding initializations. For comparison, the results of *compositional* are also listed here.

We first compare the performance of *compositional*$^-$ using different types of initializations. The experimental results show that M_0-initialization is superior to *true*-initialization. *Compositional*$^-$ with M_0-initialization succeeds in giving results for all cases, whereas *compositional*$^-$ with *true*-initialization runs out of time for three cases of the *Consensus* example and all cases of following three examples: *WLAN* with $T = 2{,}000$, *WLAN* with $T = 1{,}000$, and *FireWire*. Looking at the #*MC* columns, we further find that the required number of model checking calls by *compositional*$^-$ with M_0-initialization is much less than that with *true*-initialization in most cases. This fact well explains the reason of M_0-initialization outperforming *true*-initialization. In fact, M_0-initialization is much more aggressive than *true*-initialization. In most cases, M_0-initialization itself is already a proper assumption for the compositional verification framework.

We then compare the performance of *compositional*$^-$ to *compositional*. In the last row of Table VIII, we sum up the results of all test cases. Note that *compositional* uses *true*-initialization as well, but it requires fewer iterations of refinement than *compositional*$^-$ with *true*-initialization. This witnesses the acceleration of the learning procedure in finding proper assumptions in the compositional verification framework. The learning process leads our approach to revise several transitions in each refinement iteration. In contrast, *compositional*$^-$ revises exactly one transition in each refinement iteration. Comparing *compositional* to *compositional*$^-$ with M_0-initialization, we observe that the former performs much better than the latter. In total, *compositional*$^-$ with M_0-initialization requires about double runtimes and triple MTBDDs than *compositional*. This observation witnesses the power of the learning algorithm. Let A_{M_0} be the M_0-initialization. The learning algorithm often infers a much smaller assumption than A_{M_0}.

9. RELATED WORKS

Probabilistic model checking. There is much literature that deals with verification and analysis of probabilistic systems [Baier and Katoen 2008]. Two categories of correctness properties are identified for probabilistic systems: the qualitative properties that require the satisfaction of the specification with probability 1 and the quantitative

[9]Our approach *compositional* uses the *true*-initialization.

Table VIII. Experimental Results on the Learning Algorithm

Example	Param	Compositional			Compositional$^-$ (M_0-init)			Compositional$^-$ (*true*-init)		
		#MC	*Time*	*Size*	*#MC*	*Time*	*Size*	*#MC*	*Time*	*Size*
Consensus	(2, 6)	6	6.1	0.4	2	3.2	0.4	30	30.7	0.4
	(2, 8)	6	13.7	0.5	2	7.0	0.4	38	97.1	0.5
	(2, 10)	6	24.9	0.5	2	12.9	0.4	46	186.9	0.5
	(4, 2)	6	35.8	2.2	2	24.6	2.2	6	50.2	2.2
	(4, 4)	6	241.1	2.3	2	174.5	2.3	6	319.7	2.2
	(4, 6)	6	703.5	2.3	2	513.1	2.3	8	1,339.9	2.2
	(4, 8)	6	1,627.4	2.3	2	1,223.0	2.3	—	—	—
	(4, 10)	6	3,051.7	2.3	4	7,173.6	2.4	—	—	—
	(6, 2)	6	1,068.1	6.8	2	802.7	7.0	6	1220.3	6.7
	(6, 4)	6	5,751.4	6.7	2	7,858.1	6.9	—	—	—
WLAN ($T = 2000$)	(2,1)	4	3.8	33.2	2	359.0	304.3	—	—	—
	(2,2)	4	5.9	33.2	2	337.0	304.3	—	—	—
	(3,1)	4	3.8	40.9	2	646.4	626.7	—	—	—
	(3,2)	4	5.9	40.9	2	632.0	626.7	—	—	—
	(3,3)	11	977.9	664.8	1	636.0	626.7	—	—	—
	(4,1)	4	5.1	52.1	2	1,067.6	1,321.2	—	—	—
	(4,2)	4	7.6	52.1	2	1,066.0	1,321.2	—	—	—
	(4,3)	11	1,776.1	1,421.5	1	1,088.7	1,321.2	—	—	—
	(4,4)	11	2,061.7	1,421.5	1	1,129.8	1,321.2	—	—	—
WLAN ($T = 1000$)	(2,1)	4	3.1	33.2	2	180.7	295.2	—	—	—
	(2,2)	4	4.9	33.2	2	185.4	295.2	—	—	—
	(3,1)	4	3.4	40.9	2	304.0	603.9	—	—	—
	(3,2)	4	5.4	40.9	2	308.6	603.9	—	—	—
	(3,3)	9	358.5	604.3	1	316.1	603.9	—	—	—
	(4,1)	4	4.8	52.1	2	573.6	1,271.2	—	—	—
	(4,2)	4	7.0	52.1	2	577.3	1,271.2	—	—	—
	(4,3)	9	635.8	1,271.7	1	590.5	1,271.2	—	—	—
	(4,4)	9	757.2	1,271.7	1	621.5	1,271.2	—	—	—
FireWire	200	4	37.1	129.8	2	84.8	836.9	—	—	—
	300	4	38.5	129.8	2	276.4	1,196.4	—	—	—
	400	4	38.9	129.8	2	481.1	1,220.9	—	—	—
	500	4	39.3	129.8	2	649.1	1,221.5	—	—	—
	600	4	39.2	129.8	2	878.6	1,243.5	—	—	—
	700	4	39.2	129.8	2	1,015.6	1,243.6	—	—	—
	800	4	39.2	129.8	2	1,300.8	1,243.6	—	—	—
	900	4	39.2	129.8	2	1,405.6	1,243.6	—	—	—
	1,000	4	39.5	129.8	2	1,664.1	1,243.6	—	—	—
Philos	10	1	1.2	5.0	1	2.2	14.5	1	0.9	5.0
	15	1	5.0	11.0	1	11.2	32.9	1	4.7	11.0
	20	1	13.4	19.3	1	29.3	58.5	1	13.2	19.3
	25	1	31.5	29.9	1	59.3	91.3	1	30.7	29.9
	30	1	54.7	42.8	1	117.4	131.2	1	53.2	42.8
	35	1	96.5	57.9	1	1,082.7	178.3	1	93.2	57.9
	40	1	168.1	75.3	1	2,252.5	232.6	1	140.4	75.3
Total		211	19,872	8,596	77	39,723	26,720	—	—	—

properties that require the satisfaction of the specification measurable with a numerical value. Verification techniques for finite Markov chains and finite MDPs against qualitative properties were first addressed by Vardi [1985, 1999] and Vardi and Wolper [1986, 1994], Pnueli and Zuck [1986], and Baier and Kwiatkowska [1998]. Techniques for the quantitative analysis of probabilistic models were proposed by Courcoubetis and Yannakakis [1995], Couvreur et al. [2003], and Bustan et al. [2004].

Techniques for extending the temporal logic to probabilistic settings exist in many works. A branching-time temporal logic for reasoning about probabilistic systems was originally proposed in Hart and Sharir [1986]. Hansson and Jonsson [1994] introduced PCTL, which is capable of specifying properties with both time and probability (e.g., "after a request there is at least a 95% probability that this request will be responded within 2 seconds"). They also developed algorithms for checking PCTL properties on Markov chains. Variants of PCTL have been proposed for MDPs and other probabilistic models [Hansson 1994; Segala and Lynch 1994; Bianco and de Alfaro 1995]. A good survey on PCTL model checking can be found in Kwiatkowska et al. [2004a]. Two popular probabilistic model checkers are PRISM [Kwiatkowska et al. 2011] and MRMC [Katoen et al. 2005].

The notion of counterexamples is also important to the approach of model checking. Criteria for defining counterexamples for nonprobabilistic systems are identified in Clarke et al. [2002]. Extending the notion of counterexamples to probabilistic settings has been investigated by many researchers. Various formalisms of counterexamples for probabilistic systems have been proposed, including sets of paths [Aljazzar et al. 2005; Han et al. 2009], Markov chains [Chatterjee et al. 2005; Hermanns et al. 2008], MDPs [Chadha and Viswanathan 2010], and graphs with strongly connected components [Wimmer et al. 2012, 2013].

The preceding techniques form the basis of probabilistic model checking. Our technique differs from these techniques by focusing on compositional verification of probabilistic systems. Given a system consisting of several concurrent components, the preceding techniques attempt to construct a system model by composing the behaviors of all components and then perform model checking on this system model. Note that the composition of concurrent components may lead to severe state space explosion. Our compositional approaches use the divide-and-conquer strategy to avoid the construction of the system model. In the following, we discuss related works in compositional verification of probabilistic systems.

Compositional verification of probabilistic systems. The most relevant works to ours are Kwiatkowska et al. [2010] and Feng et al. [2010, 2011]. In their proof rules, assumptions are classical deterministic finite automata. The L^* algorithm has been applied to infer classical assumptions in Feng et al. [2010]. As discussed earlier, 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 Feng et al. [2011]. However, the framework only works for fully probabilistic discrete time Markov chains and may not terminate. In contrast, our technique applies to MDPs and always terminates.

The undecidability of inferring labeled probabilistic transition systems under Angluin's active learning model is shown in Komuravelli et al. [2012b]. A (necessarily) restricted learning algorithm for such probabilistic systems is also proposed in the same work. 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 techniques. This technique, called *assume-guarantee*

abstraction-refinement (AGAR), was first proposed for compositional verification of classical systems [Gheorghiu Bobaru et al. 2008] and then extended in Komuravelli et al. [2012a, 2012b] for probabilistic systems. In Komuravelli et al. [2012a], probabilistic assumptions are conservative abstractions of system components. They are iteratively refined by counterexamples [Clarke et al. 2000]. However, AGAR relies on partitioning the explicit state space to construct assumptions [Gheorghiu Bobaru et al. 2008; Komuravelli et al. 2012a]. We are not aware of any symbolic implementation of AGAR techniques for classical or probabilistic systems.

Learning algorithms for probabilistic systems. Various learning algorithms have been proposed for probabilistic systems [Sen et al. 2004; Mao et al. 2011, 2012; Chen et al. 2012]. These learning algorithms adopt a passive learning model. They are not applicable to the learning-based assume-guarantee reasoning framework in Cobleigh et al. [2003]. To the best of our knowledge, an exact learning algorithm for probabilistic systems under Angluin's active learning model is yet to be found.

Multiplicity automata is another formalism for representing stochastic languages. Efficient algorithms exist for learning multiplicity automata under Angluin's learning model [Bergadano and Varricchio 1996; Beimel et al. 1996, 2000]. However, a multiplicity automaton may generate a stochastic language that cannot be generated by any probabilistic automaton [Denis and Esposito 2004]. This formalism is thus not applicable to our verification framework.

Learning algorithms for binary decision diagrams were proposed in Gavaldà and Guijarro [1995] and Nakamura [2005]. In Gavaldà and Guijarro [1995], an L^*-based algorithm was developed. The work in Nakamura [2005] used a classification tree-based learning algorithm for regular languages. Both algorithms inferred deterministic finite automata and transformed them into decision diagrams.

10. CONCLUSION

We proposed a sound and complete learning-based assume-guarantee reasoning technique for probabilistic safety properties on MDPs. 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 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 let $P_{\leq p}[\psi]$ be 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. However, it is not recommended when M is composed of concurrent MDPs. Since the construction of the composition can be expensive, the computation should be deferred after concurrent components are simplified. Assume-guarantee reasoning presented in this article 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 $\lhd \in \{<, \leq, >, \geq\}$ are represented as graphs with strongly connected components [Wimmer et al. 2012, 2013]. We plan to generalize transition contributions to select counterexamples from spurious weighted witnesses with strongly connected components. Additionally, we would like to extend our learning framework to verifying richer properties, such as general probabilistic safety or liveness properties.

REFERENCES

Husain Aljazzar, Holger Hermanns, and Stefan Leue. 2005. Counterexamples for timed probabilistic reachability. In *Formal Modeling and Analysis of Timed Systems.* Lecture Notes in Computer Science, Vol. 3829. Springer, 177–195.

Dana Angluin. 1987. Learning regular sets from queries and counterexamples. *Information and Computation* 75, 2, 87–106.

James Aspnes and Maurice Herlihy. 1990. Fast randomized consensus using shared memory. *Journal of Algorithms* 11, 3, 441–460.

Christel Baier, Edmund M. Clarke, Vasiliki Hartonas-Garmhausen, Marta Kwiatkowska, and Mark Ryan. 1997. Symbolic model checking for probabilistic processes. In *Automata, Languages and Programming*. Lecture Notes in Computer Science, Vol. 1256. Springer, 430–440.

Christel Baier and Joost-Pieter Katoen. 2008. *Principles of Model Checking*. MIT Press, Cambridge, MA.

Christel Baier and Marta Kwiatkowska. 1998. On the verification of qualitative properties of probabilistic processes under fairness constraints. *Information Processing Letters* 66, 2, 71–79.

Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. 1996. On the applications of multiplicity automata in learning. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science.* IEEE, Los Alamitos, CA, 349–358.

Amos Beimel, Francesco Bergadano, Nader H. Bshouty, Eyal Kushilevitz, and Stefano Varricchio. 2000. Learning functions represented as multiplicity automata. *Journal of the ACM* 47, 3, 506–530.

Francesco Bergadano and Stefano Varricchio. 1996. Learning behaviors of automata from multiplicity and equivalence queries. *SIAM Journal on Computing* 25, 6, 1268–1280.

Andrea Bianco and Luca de Alfaro. 1995. Model checking of probabilistic and nondeterministic systems. In *Foundations of Software Technology and Theoretical Computer Science*. Lecture Notes in Computer Science, Vol. 1026. Springer, 499–513.

Benedikt Bollig, Joost-Pieter Katoen, Carsten Kern, Martin Leucker, Daniel Neider, and David R. Piegdon. 2010. libalf: The automata learning framework. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 6174. Springer, 360–364.

Doron Bustan, Sasha Rubin, and Moshe Y. Vardi. 2004. Verifying ω-regular properties of Markov chains. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 3114. Springer, 189–201.

Rohit Chadha and Mahesh Viswanathan. 2010. A counterexample-guided abstraction-refinement framework for Markov decision processes. *ACM Transactions on Computational Logic* 12, 1, 1:1–1:49.

Krishnendu Chatterjee, Thomas A. Henzinger, Ranjit Jhala, and Rupak Majumdar. 2005. Counterexample-guided planning. In *Proceedings of the 21st International Conference on Uncertainty in Artificial Intelligence*. 104–111.

Yingke Chen, Hua Mao, Manfred Jaeger, ThomasDyhre Nielsen, Kim Guldstrand Larsen, and Brian Nielsen. 2012. Learning Markov models for stationary system behaviors. In *NASA Formal Methods.* Lecture Notes in Computer Science, Vol. 7226. Springer, 216–230.

Yu-Fang Chen, Edmund M. Clarke, Azadeh Farzan, Ming-Hsien Tsai, Yih-Kuen Tsay, and Bow-Yaw Wang. 2010. Automated assume-guarantee reasoning through implicit learning. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 6174. Springer, 511–526.

Yu-Fang Chen, Azadeh Farzan, Edmund M. Clarke, Yih-Kuen Tsay, and Bow-Yaw Wang. 2009. Learning minimal separating DFA's for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science, Vol. 5505. Springer, 31–45.

Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. 2000. Counterexample-guided abstraction refinement. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 1855. Springer, 154–169.

Edmund Clarke, Somesh Jha, Yuan Lu, and Helmut Veith. 2002. Tree-like counterexamples in model checking. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*. IEEE, Los Alamitos, CA, 19–29.

Jamieson M. Cobleigh, George S. Avrunin, and Lori A. Clarke. 2008. Breaking up is hard to do: An evaluation of automated assume-guarantee reasoning. *ACM Transactions on Software Engineering and Methodology* 17, 2, 7.

Jamieson M. Cobleigh, Dimitra Giannakopoulou, and Corina S. Păsăreanu. 2003. Learning assumptions for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science, Vol. 2619. Springer, 331–346.

Costas Courcoubetis and Mihalis Yannakakis. 1995. The complexity of probabilistic verification. *Journal of the ACM* 42, 4, 857–907.

Jean-Michel Couvreur, Nasser Saheb, and Grégoire Sutre. 2003. An optimal automata approach to LTL model checking of probabilistic systems. In *Logic for Programming, Artificial Intelligence, and Reasoning.*, Lecture Notes in Computer Science, Vol. 2850. Springer, 361–375.

Luca De Alfaro, Marta Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. 2000. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In *Tools and*

Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science, Vol. 1758. Springer, 395–410.

François Denis and Yann Esposito. 2004. Learning classes of probabilistic automata. In *Learning Theory*. Springer, 124–139.

Lu Feng, Tingting Han, Marta Kwiatkowska, and David Parker. 2011. Learning-based compositional verification for synchronous probabilistic systems. In *Automated Technology for Verification and Analysis*. Lecture Notes in Computer Science, Vol. 6996. Springer-Verlag, 511–521.

Lu Feng, Marta Kwiatkowska, and David Parker. 2010. Compositional verification of probabilistic systems using learning. In *Proceedings of the 2010 7th International Conference on the Quantitative Evaluation of Systems (QEST'10)*. IEEE, Los Alamitos, CA, 133–142.

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

Ricard Gavaldà and David Guijarro. 1995. Learning ordered binary decision diagrams. In *Algorithmic Learning Theory*. Lecture Notes in Computer Science, Vol. 997. Springer, 228–238.

Mihaela Gheorghiu, Dimitra Giannakopoulou, and Corina S. Păsăreanu. 2007. Refining interface alphabets for compositional verification. In *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science, Vol. 4424. Springer, 292–307.

Mihaela Gheorghiu Bobaru, Corina S. Păsăreanu, and Dimitra Giannakopoulou. 2008. Automated assume-guarantee reasoning by abstraction refinement. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 5123. Springer, 135–148.

Anubhav Gupta, Kenneth L. McMillan, and Zhaohui Fu. 2007. Automated assumption generation for compositional verification. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 4590. Springer, 420–432.

Tingting Han, Joost-Pieter Katoen, and Damman Berteun. 2009. Counterexample generation in probabilistic model checking. *IEEE Transactions on Software Engineering* 35, 2, 241–257.

Hans Hansson and Bengt Jonsson. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6, 5, 512–535.

Hans A. Hansson. 1994. *Time and Probability in Formal Design of Distributed Systems*. Elsevier Science, New York, NY.

Sergiu Hart and Micha Sharir. 1986. Probabilistic propositional temporal logics. *Information and Control* 70, 2, 97–155.

Fei He, Xiaowei Gao, Bow-Yaw Wang, and Lijun Zhang. 2015. Leveraging weighted automata in compositional reasoning about concurrent probabilistic systems. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, New York, NY, 503–514.

Fei He, Bow-Yaw Wang, Liangze Yin, and Lei Zhu. 2014. Symbolic assume-guarantee reasoning through BDD learning. In *Proceedings of the 36th International Conference on Software Engineering (ICSE'14)*. ACM, New York, NY, 1071–1082.

Holger Hermanns, Björn Wachter, and Lijun Zhang. 2008. CEGAR. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 5123. Springer, 162–175.

IEEE. 2012. *IEEE Standard for Information Technology–Telecommunications and Information Exchange Between sSystems Local and Metropolitan Area Networks–Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE, Los Alamitos, CA.

Joost-Pieter Katoen, Maneesh Khattri, and Ivan S. Zapreev. 2005. A Markov reward model checker. In *Proceedings of the 2nd International Conference on the Quantitative Evaluation of Systems*. IEEE, Los Alamitos, CA, 243–244.

Joost-Pieter Katoen, Lei Song, and Lijun Zhang. 2014. Probably safe or live. In *Proceedings of the Joint Meeting of the 23rd EACSL Annual Conference on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. ACM, New York, NY, 55:1–55:10.

Shinji Kimura and Edmund M. Clarke. 1990. A parallel algorithm for constructing binary decision diagrams. In *Proceedings of the IEEE International Conference on Computer Design (ICCD'90)*. IEEE, Los Alamitos, CA, 220–223.

Anvesh Komuravelli, Corina S. Păsăreanu, and Edmund M. Clarke. 2012a. Assume-guarantee abstraction refinement for probabilistic systems. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 7358. Springer, 310–326.

Anvesh Komuravelli, Corina S. Păsăreanu, and Edmund M. Clarke. 2012b. Learning probabilistic systems from tree samples. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science (LICS'12)*. IEEE, Los Alamitos, CA, 441–450.

Marta Kwiatkowska, Gethin Norman, and David Parker. 2004a. Modelling and verification of probabilistic systems. In *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*. CRM Monograph Series, Vol. 23, American Mathematical Society, 93–215.

Marta Kwiatkowska, Gethin Norman, and David Parker. 2004b. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer* 6, 2, 128–142.

Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification*. Lecture Notes in Computer Science, Vol. 6806. Springer, 585–591.

Marta Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. 2010. Assume-guarantee verification for probabilistic systems. In *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science, Vol. 6015. Springer, 23–37.

Marta Kwiatkowska, Gethin Norman, and Jeremy Sproston. 2003. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Formal Aspects of Computing* 14, 3, 295–318.

Daniel Lehmann and Michael O. Rabin. 1981. On the advantage of free choice: A symmetric and fully distributed solution to the dining philosophers problem (extended abstract). In *Proceedings of the 8th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'81)*. ACM, New York, NY, 133–138.

Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. 2011. Learning probabilistic automata for model checking. In *Proceedings of the 8th International Conference on Quantitative Evaluation of Systems (QEST'11)*. IEEE, Los Alamitos, CA, 111–120.

Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. 2012. Learning Markov decision processes for model checking. arXiv:1212.3873.

Rajeev Motwani and Prabhakar Raghavan. 1995. *Randomized Algorithms*. Cambridge University Press.

Atsuyoshi Nakamura. 2005. An efficient query learning algorithm for ordered binary decision diagrams. *Information and Computation* 201, 2, 178–198.

David Anthony Parker. 2002. *Implementation of Symbolic Model Checking for Probabilistic Systems*. Ph.D. Dissertation. University of Birmingham.

Amir Pnueli and Lenore Zuck. 1986. Probabilistic verification by tableaux. In *Proceedings of the 1st Symposium in Logic in Computer Science (LICS'86)*. IEEE, Los Alamitos, CA, 322–331.

Jan Rutten, Marta Kwiatkowska, Gethin Norman, and David Parker. 2004. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems,* P. Panangaden and F. van Breugel (Eds.). CRM Monograph Series, Vol. 23. American Mathematical Society.

Roberto Segala and Nancy Lynch. 1994. Probabilistic simulations for probabilistic processes. In *CONCUR'94: Concurrency Theory*. Lecture Notes in Computer Science, Vol. 836. Springer, 481–496.

Koushik Sen, Mahesh Viswanathan, and Gul Agha. 2004. Learning continuous time Markov chains from sample executions. In *Proceedings of the 1st International Conference on the Quantitative Evaluation of Systems (QEST'04)*. IEEE, Los Alamitos, CA, 146–155.

Moshe Y. Vardi. 1985. Automatic verification of probabilistic concurrent finite state programs. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science*. IEEE, Los Alamitos, CA, 327–338.

Moshe Y. Vardi. 1999. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *Formal Methods for Real-Time and Probabilistic Systems*. Springer, 265–276.

Moshe Y. Vardi and Pierre Wolper. 1986. An automata-theoretic approach to automatic program verification. In *Proceedings of the 1st Symposium in Logic in Computer Science (LICS'86)*. IEEE, Los Alamitos, CA, 332–344.

Moshe Y. Vardi and Pierre Wolper. 1994. Reasoning about infinite computations. *Information and Computation* 115, 1, 1–37.

Ralf Wimmer, Nils Jansen, Erika Ábrahám, Bernd Becker, and Joost-Pieter Katoen. 2012. Minimal critical subsystems for discrete-time Markov models. In *Tools and Algorithms for the Construction and Analysis of Systems*. Lecture Notes in Computer Science, Vol. 7214. Springer, 299–314.

Ralf Wimmer, Nils Jansen, Andreas Vorpahl, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. 2013. High-level counterexamples for probabilistic automata. In *Quantitative Evaluation of Systems*. Lecture Notes in Computer Science, Vol. 8054. Springer, 39–54.

He Zhu, Fei He, William N. N. Hung, Xiaoyu Song, and Ming Gu. 2009. Data mining based decomposition for assume-guarantee reasoning. In *Proceedings of the Conference on Formal Methods in Computer-Aided Design (FMCAD'09)*. IEEE, Los Alamitos, CA, 116–119.

Received August 2015; revised December 2015; accepted March 2016