International Journal of Networked and Distributed Computing
Vol. **8**(**2**); March (2020), *pp.* 94–107
DOI: https://doi.org/10.2991/ijndc.k.190918.001; ISSN 2211-7938; eISSN 2211-7946
https://www.atlantis-press.com/journals/ijndc

Research Article

Compositional Stochastic Model Checking Probabilistic Automata via Assume-guarantee Reasoning

Yang Liu*, Rui Li

School of Information Engineering, Nanjing University of Finance & Economics, Nanjing, Jiangsu 210046, China

ARTICLE INFO

Article History

Received 04 April 2019
Accepted 01 May 2019

Keywords

Stochastic model checking
assume-guarantee reasoning
symmetric assume-guarantee rule
learning algorithm
probabilistic automata

ABSTRACT

Stochastic model checking is the extension and generalization of the classical model checking. Compared with classical model checking, stochastic model checking faces more severe state explosion problem, because it combines classical model checking algorithms and numerical methods for calculating probabilities. For dealing with this, we first apply symmetric assume-guarantee rule symmetric (SYM) for two-component systems and symmetric assume-guarantee rule for n-component systems into stochastic model checking in this paper, and propose a compositional stochastic model checking framework of probabilistic automata based on the NL* algorithm. It optimizes the existed compositional stochastic model checking process to draw a conclusion quickly, in cases the system model does not satisfy the quantitative properties. We implement the framework based on the PRISM tool, and several large cases are used to demonstrate the performance of it.

1. INTRODUCTION

Formal verification can reveal the unexposed defects in a safety-critical system. As a prominent formal verification technique, model checking is an automatic and complete verification technique of finite state systems against correctness properties, which was pioneered respectively by Clarke and Emerson [1] and by Queille and Sifakis [2] in the early 1980's. Whereas model checking techniques focus on the absolute correctness of systems, in practice such rigid notions are hard, or even impossible, to ensure. Instead, many systems exhibit stochastic aspects [3] which are essential for among others: modeling unreliable and unpredictable system behavior (message garbling or loss), model-based performance evaluation (i.e., estimating system performance and dependability) and randomized algorithms (leader election or consensus algorithms). Automatic formal verification of stochastic systems by model checking is called stochastic model checking or probabilistic model checking [4].

Stochastic model checking algorithms rely on a combination of model checking techniques for classical model checking and numerical methods for calculating probabilities. So, stochastic model checking faces more severe state explosion problem, compared with classical model checking [5]. There are some works to deal with this problem through bounded probabilistic model checking [6], abstraction refinement [7], compositional verification [8] and so on. The crucial notion of compositional verification is "divide and conquer". It can decompose the whole system into separate components and conquer each component separately. The compositional verification techniques include assume-guarantee reasoning [9], contract-based methods [10] and invariant-based methods [11]. This paper focuses on assume-guarantee reasoning, which is an automatic method of compositional verification. To account for the relationship between the whole system and its different components, assume-guarantee reasoning gives some rules, which can change the global verification of a system into local verification of individual components.

Theoretically speaking, applying the assume-guarantee reasoning into stochastic model checking is a feasible way to solve the state explosion problem. There is some research work done in this direction [12–15]. We argue that applying the assume-guarantee reasoning into stochastic model checking should solve the following four issues, which is named as AG-SMC problem: (1) How to generate appropriate assumptions. (2) How to check the assume-guarantee triple. (3) How to construct a counterexample. (4) How to verify a stochastic system composed of n ($n \geq 2$) components.

1.1. Related Work

According to the generation type of assumptions, we divided the existed work into two categories.

1.1.1. Manual interactive assumption generation

On the existing theory of Markov Decision Process (MDP) model of combinatorial analysis [16], Kwiatkowska et al. [17] first gives

*Corresponding author. Email: yliu@nufe.edu.cn

out assume-guarantee reasoning for verifying probabilistic automaton (PA) model, including asymmetric assumption-guarantee rule (ASYM), circular assumption-guarantee rule (CRIC) and asynchronous assumption-guarantee rule (ASYNC). It solves the AG-SMC problem as follows: (1) It generates the assumptions through the manual interactive method. (2) In the triple of the form $\langle A\rangle_{\geq \mathrm{PA}} M \langle P\rangle_{\geq \mathrm{PG}}$, system model M is a PA, the assumption $\langle A\rangle_{\geq \mathrm{PA}}$ and guarantee $\langle P\rangle_{\geq \mathrm{PG}}$ are probabilistic safety properties, represented by deterministic finite automaton (DFA). When system component M satisfies assumptions A with minimum probability PA, it will be able to satisfy property P with minimum probability PG. Checking the triple can be reduced to multi-objective model checking [18], which is equivalent to a linear programming (LP) problem. (3) It does not involve to construct the counterexamples. (4) It verifies a stochastic system composed of $n \geq 2$ components through multi-component asymmetric assume-guarantee rule (ASYM-N). The core idea of ASYM-N rule is similar to CRIC rule, i.e., the component M_1 satisfies the guarantee $\langle A_1\rangle_{\geq \mathrm{PA}_{M_1}}$, then the guarantee $\langle A_1\rangle_{\geq \mathrm{PA}_{M_1}}$ as the assumption of the component M_2, let the component M_2 can satisfy the guarantee $\langle A_2\rangle_{\geq \mathrm{PA}_{M_2}}$, …, until the component M_n that satisfies the assumption $\langle A_{n-1}\rangle_{\geq \mathrm{PA}_{M_{n-1}}}$ can satisfy the guarantee $\langle P\rangle_{\geq \mathrm{PG}}$. If all above-mentioned conditions hold, the entire system model $M_1 || M_2 || \cdots || M_n$ will satisfy the guarantee $\langle P\rangle_{\geq \mathrm{PG}}$.

1.1.2. Automated assumption generation

Bouchekir and Boukala [19], He et al. [20], Komuravelli et al. [21], Feng et al. [22] and [23] are the automated assumption generation methods for solving the AG-SMC problem. They can be divided into the following three kinds further.

1.1.2.1. Learning-based assumption generation

Based on the learning-based assume-guarantee reasoning (LAGR) technology and the ASYM rule proposed in Segala [16], Feng et al. [22] proposes L*-based learning framework for PA model, which can be used to verify whether the given PA model satisfies the probabilistic safety property. Feng et al. [22] uses the cases to demonstrate the performance of its method, including the client–server, sensor network and the randomized consensus algorithm. For the AG-CSMC problem, Segala [16] can be specifically described in the following four aspects: (1) Through the L* learning algorithm, the process of generating an appropriate assumption $\langle A\rangle_{\geq \mathrm{PA}}$ is fully automated, i.e., we need to generate a closed and consistent observation table through membership queries, to generate a conjectured assumption, and then verify the correctness of the assumption through equivalence queries. (2) It checks the assume-guarantee triple through multi-objective model checking [18]. (3) In the whole learning process, Feng et al. [22] adopts the method proposed in Han et al. [24] to generate probabilistic counterexamples for refining the current assumption, i.e., the PRISM [25] is used to obtain the error state nodes in the model, and then the probabilistic counterexamples are constructed by using Eppstein's [26] algorithm. (4) The verification problem of a stochastic system composed of $n \geq 2$ components is not solved.

Feng et al. [23] makes further research based on Feng et al. [22] and uses several large cases to demonstrate the performance of it, including client–server, sensor network, randomized consensus algorithm and Mars Exploration Rovers (MER). For the AG-CSMC problem, compared with Feng et al. [23] and Feng et al. [22], the contribution of Feng et al. [23] is reflected in the better solution of the first sub-problem and the solution of the fourth sub-problem, which will be illustrated in the following two aspects: (1) Feng et al. [23] compares the assumption generation process between the L* learning algorithm and the NL* learning algorithm, and finds that NL* often needs fewer membership and equivalence queries than L* in large cases. (2) Based on Segala [16], Feng et al. [23] uses the ASYM-N rule to propose a learning framework for compositional stochastic model checking, and uses it to verify the multi-component stochastic system. So far, in the learning-based assumption generation method, four sub-problems of AG-CSMC problem have been solved basically.

1.1.2.2. Symbolic learning-based assumption generation

One deficiency of learning-based assumption generation method is that the learning framework is sound but incomplete. Based on ASYM rule, He et al. [20] proposes an assume-guarantee rule containing weighted assumption for the first time, and provides a sound and complete learning framework, which can verify whether the probabilistic safety properties are satisfied on the MDP model. Through randomized consensus algorithm, wireless LAN protocol, FireWire protocol and randomized dining philosophers, He et al. [20] demonstrates the performance of its method. For the AG-CSMC problem, He et al. [20] can be specifically described in the following four aspects: (1) The weighted assumption can be represented by Multi-terminal Binary Decision Diagrams (MTBDD). Based on the L* learning algorithm, He et al. [20] proposes an MTBDD learning algorithm to automatically generate the weighted assumption, which is represented by a k-Deterministic Finite Automaton (k-DFA). MTBDD learning algorithm can make membership queries on binary strings of arbitrary lengths and answer membership queries on valuations over fixed variables by the teacher. (2) Through the weighted extension of the classical simulation relation, He et al. [20] presents a verification method of the assume-guarantee triple containing the weighted assumption. (3) Similarly to Feng et al. [22], He et al. [20] also constructs the necessary probabilistic counterexamples in the learning process through Han et al. [24]. (4) The verification problem of a stochastic system composed of $n \geq 2$ components is not solved.

In Bouchekir and Boukala [19], the method realizes automatic assumption generation through the Symbolic Learning-based Assume-Guarantee Reasoning technology, also known as the Probabilistic Symbolic Compositional Verification (PSCV). The PSCV method provides a sound and complete symbolic assume-guarantee rule to verify whether the MDP model satisfies the Probabilistic Computation Tree Logic (PCTL) property. It is a new approach based on the combination of assume-guarantee reasoning and symbolic model checking techniques. Bouchekir and Boukala [19] uses randomized mutual exclusion, client–server, randomized dining philosophers, randomized self-stabilizing algorithm and Dice to demonstrate the performance of its method. For the AG-CSMC problem, Bouchekir and Boukala [19] can be specifically described in the following four aspects: (1) Appropriate assumptions are automatically generated by symbolic MTBDD learning algorithm, and represented by interval MDP (IMDP), thus ensuring the completeness of symbolic assume-guarantee rule. Moreover, in addition, to

reduce the size of the state space, the PSCV method encodes both system components and assumptions implicitly using compact data structures, such as BDD or MTBDD. (2) Bouchekir and Boukala [19] uses the method in He et al. [20] to verify assume-guarantee triple. (3) To refine assumptions, the PSCV method [27] uses the causality method to construct counterexamples, i.e., it uses K* algorithm [28] in the DiPro tool to construct counterexamples, and applies the algorithms in Debbi and Bourahla [29] to construct the most indicative counterexample. (4) Verification of a stochastic system composed of $n \geq 2$ components is not involved.

1.1.2.3. Assumption generation based on abstraction-refinement

The method in Komuravelli et al. [21] is similar to Counterexample Guided Abstraction Refinement (CEGAR) [30]. It uses the Assume-Guarantee Abstraction Refinement technology to propose an assume-guarantee compositional verification framework for Labeled Probabilistic Transition Systems (LPTSes), which can verify whether the given LPTS model satisfies the safe-PCTL property. Komuravelli et al. [21] uses the client–server, MER and wireless sensor network to demonstrate the performance of its method. For the AG-CSMC problem, Komuravelli et al. [21] can be specifically described in the following four aspects: (1) The method can use tree counterexamples from checking one component to refine the abstraction of another component. Then, it uses the abstraction as the assumptions for assume-guarantee reasoning, represented by LPTS. (2) It uses a strong simulation relationship to check the assume-guarantee triple. (3) The process of constructing tree counterexample can be reduced to check the Satisfiability Modulo Theories problem, and then solve it through Yices [31]. (4) It also verifies an n-component stochastic system ($n \geq 2$) by the ASYM-N rule.

1.2. Our Contribution

This paper presents some improvements based on the probabilistic assume-guarantee framework proposed in Feng et al. [23]. On one hand, our optimization is to verify each membership and equivalence query, to seek a counterexample, which can prove the property is not satisfied. If the counterexample is not spurious, the generation of the assumptions will stop, and the verification process will also terminate immediately. On the other hand, a potential shortage of the ASYM displays that the sole assumption A about M_1 is present, but the additional assumption about M_2 is nonexistent. We thus apply the SYM rule to the compositional verification of PAs and extend the rule to verify an n-component system ($n \geq 2$). Through several large cases, it is shown that our improvements are feasible and efficient.

1.3. Paper Structure

The rest of the paper is organized as follows. Section 2 introduces the preliminaries used in this paper, which include PAs, model checking and the NL* algorithm. Section 3 presents a compositional stochastic model checking framework based on the SYM rule and optimizes the learning framework. Then, the framework is extended to an n-component system ($n \geq 2$) in Section 4. Section 5 develops a prototype tool for the framework, and compares it with Feng et al. [23] by several large cases. Finally, Section 6 concludes the paper and presents direction for future research.

2. BACKGROUND

2.1. Probabilistic Automata

Probabilistic automata [3,17,32,33] can model both probabilistic and nondeterministic behavior of systems, which is a slight generalization of MDPs. The verification algorithms for MDPs can be adapted for PAs.

In the following, Dist(V) is defined as the set of all discrete probability distributions over a set V. η_v is defined as the point distribution on $v \in V$. $\mu_1 \times \mu_2 \in \mathrm{Dist}(V_1 \times V_2)$ is the product distribution of $\mu_1 \in \mathrm{Dist}(V_1)$ and $\mu_2 \in \mathrm{Dist}(V_2)$.

Definition 1. (probabilistic automaton) A probabilistic automaton (PA) is a tuple $M = (V, \bar{v}, \alpha_M, \delta_M, L)$ where V is a set of states, $\bar{v} \in V$ is an initial state, α_M is an alphabet for all the action, $\delta_M \subseteq V \times (\alpha_M \cup \{\tau\}) \times \mathrm{Dist}(V)$ is a probabilistic transition relation. τ is an invisible action, and L: $V \to 2^{AP}$ is a labeling function mapping each state to a set of atomic propositions taken from a set AP.

In any state v of a PA M, we use the transition $v \xrightarrow{\alpha} \mu$ to denote that $(v, \alpha, \mu) \in \delta_M$, where $\alpha \in \alpha_M \cup \{\tau\}$ is an action label. μ is a probability distribution over state v. All transitions are nondeterministic, and it will make a random choice according to the distribution μ. A trace through M is a (finite or infinite) sequence $v_0 \xrightarrow{\alpha_0, \mu_0} v_1 \xrightarrow{\alpha_1, \mu_1} \cdots$ where $v_0 = \bar{v}$, and for each $i \geq 0$, $v_i \xrightarrow{\alpha_i} \mu_i$ is a transition and $\mu_i(v_{i+1}) > 0$. The sequence of actions $\alpha_0, \alpha_1, \ldots$, after removal of any τ, from a trace t is also called a path. An adversary σ is sometimes referred to as scheduler, policy, or strategy, which maps any finite path to a sub-distribution over the available transitions in the last state of the path. This paper focuses on are finite-memory adversaries, which store information about the history in a finite-state automaton (see Baier and Katoen [3] Definition 10.97; pp. 848). We define Trace_M^σ as the set of all traces through M under the control of adversary σ, and Adv_M as the set of all potential adversaries for M. For an adversary, we define a probability space $\Pr_M^\sigma$ on Trace_M^σ, and the probability space can know the probability of the adversary σ.

Definition 2. (Parallel composition of PAs) If $M_1 = (V_1, \bar{v}_1, \alpha_{M_1}, \delta_{M_1}, L_1)$ and $M_2 = (V_2, \bar{v}_2, \alpha_{M_2}, \delta_{M_2}, L_2)$ are PAs, then their parallel composition is denoted as $M_1 \| M_2$. It is given by the $\mathrm{PA}(V_1 \times V_2, (\bar{v}_1, \bar{v}_2), \alpha_{M_1} \cup \alpha_{M_2}, \delta_{M_1 \| M_2}, L)$ where $\delta_{M_1 \| M_2}$ is defined such that $(v_1, v_2) \xrightarrow{\alpha} \mu_1 \times \mu_2$ if and only if one of the following holds:

$$v_1 \xrightarrow{\alpha} \mu_1, v_2 \xrightarrow{\alpha} \mu_2 \text{ and } \alpha \in \alpha_{M_1} \cap \alpha_{M_2} \tag{1}$$

$$v_1 \xrightarrow{\alpha} \mu_1, \mu_2 = \eta_{v_2} \text{ and } \alpha \in (\alpha_{M_1} \setminus \alpha_{M_2}) \cup \{\tau\} \tag{2}$$

$$v_2 \xrightarrow{\alpha} \mu_2, \mu_1 = \eta_{v_1} \text{ and } \alpha \in (\alpha_{M_2} \setminus \alpha_{M_1}) \cup \{\tau\} \tag{3}$$

and

$$L(v_1, v_2) = L_1(v_1) \cup L_2(v_2) \tag{4}$$

Definition 3. (Alphabet extension of PA) For any PA $M = (V, \bar{v}, \alpha_M, \delta_M, L)$ and set of actions y, we extend the alphabet of M to

y, denoted $M[y]$, as follows: $M[y] = (V, \bar{v}, \alpha_M \cup y, \delta_{M[y]}, L)$ where $\delta_{M[y]}$ is a probabilistic transition relation on $M[y]$, and $\delta_{M[y]} = \delta_M \cup \{(v, \alpha, \eta_v) | v \in V \wedge \alpha \in y \backslash \alpha_M\}$.

For any state $v = (v_1, v_2)$ of $M_1 || M_2$, the projection of v on M_i, denoted by $v \restriction_{M_i}$. Then, we extend it to distributions on the state space $V_1 \times V_2$ of $M_1 || M_2$. For each trace t on $M_1 || M_2$, the projection of t on M_i, denoted by $t \restriction_{M_i}$, i.e., the trace can be acquired from M_i by projecting each state of t onto M_i and removing all the actions not in the alphabet α_{M_i}.

Definition 4. (Adversary projections) Let us suppose that M_1 and M_2 are PAs, σ is an adversary of $M_1 || M_2$. The projection of σ on M_i is denoted as $\sigma \restriction_{M_i}$, which is the adversary on M_i, for any finite trace t_i of M_i, $\sigma \restriction_{M_i}(t)(\alpha, \mu_i)$ equals:

$$\frac{\sum \{|\Pr^{\sigma}(t) \cdot \sigma(t)(\alpha, \mu)| \, t \in \mathrm{Trace}^{\sigma}_{M_1 || M_2} \wedge t \restriction_{M_i} = t_i \wedge \mu \restriction_{M_i} = \mu_i |\}}{\Pr^{\sigma \restriction_{M_i}}(t_i)} \quad (5)$$

2.2. Model Checking for Probabilistic Automata

Here, we concentrate on action-based properties over PAs, defined regarding their traces. In essence, we use regular languages over actions to describe these properties. A regular safety property P signifies a set of infinite words ω, the usual notation is $\mathcal{L}(P)$, that is represented by a regular language of bad prefixes, because its finite words any (possibly empty) extension is not in $\mathcal{L}(P)$. Formally, we describe that set for P by a DFA $P^{\mathrm{err}} = (V, \bar{v}, \alpha_P, \delta_P, F)$, V is a set of states, $\bar{v} \in V$ is an initial state, α_P is an alphabet, transition function δ_P: $V \times \alpha_P \to V$ and a set of accepting states $F \subseteq V$, which can store the set of bad prefixes of infinite words ω. Formally, a regular safety language $\mathcal{L}(P)$ is defined as:

$$\mathcal{L}(P) = \{\omega \in (\alpha_P)^{\omega} | \text{no prefix of } \omega \text{ is in } \mathcal{L}(P^{\mathrm{err}})\} \quad (6)$$

Provided a PA M and regular safety property P, alphabet $\alpha_P \subseteq \alpha_M$, an infinite trace t of M satisfies P, denoted $t \vDash P$, if and only if $t \restriction \alpha_P \in \mathcal{L}(P)$. For a finite trace t' of M, if some infinite traces t of which t' is a prefix satisfies P, we denote as $t' \vDash P$. For an adversary $\sigma \in \mathrm{Adv}_M$, we define the probability of M under σ satisfying P as:

$$\Pr^{\sigma}_M(P) \stackrel{\mathrm{def}}{=} \Pr^{\sigma}_M \{t \in \mathrm{Trace}^{\sigma}_M | t \vDash P\} \quad (7)$$

That is to say $\Pr^{\sigma}_M(P)$ indicates the probability of a corresponding trace t (the trace t is included by the component M under adversary σ and satisfies the property P).

Next, we define the minimum probability of satisfying P as:

$$\Pr^{\min}_M(P) \stackrel{\mathrm{def}}{=} \inf_{\sigma \in \mathrm{Adv}_M} \Pr^{\sigma}_M(P) \quad (8)$$

$\inf_{\sigma \in \mathrm{Adv}_M} \Pr^{\sigma}_M(P)$ denotes that $\Pr^{\sigma}_M(P)$ of infimum is taken over by all adversaries σ for M.

A probabilistic safety property $\langle P \rangle_{\geq \mathrm{PG}}$ contains a safety property P and a sound probability bound PG. For example, the probability of a success happening is at least 0.98. We have a PA M satisfies this property, denoted $M \vDash \langle P \rangle_{\geq \mathrm{PG}}$, if and only if the probability of satisfying P is at least PG for any adversary:

$$M \vDash \langle P \rangle_{\geq PG} \Leftrightarrow \forall \sigma \in \mathrm{Adv}_M \cdot \Pr^{\sigma}_M(P) \geq \mathrm{PG} \Leftrightarrow \Pr^{\min}_M(P) \geq \mathrm{PG} \quad (9)$$

According to the above formulae, the verification of a probabilistic safety property $\langle P \rangle_{\geq \mathrm{PG}}$ on a PA M can be transformed into calculation of the minimum probability $\Pr^{\min}_M(P)$, i.e., we should calculate the maximum probability of reaching a set of accepting states in the product of $M \otimes P^{\mathrm{err}}$ (see Kwiatkowska et al. [33] Definition 6 for details), where the DFA P^{err} represents the safety property P. In fact, a finite-memory adversary is necessary, because such an adversary σ always exists, which leads to $\Pr^{\sigma}_M(P) = \Pr^{\min}_M(P)$. Particularly, this extreme case also holds:

$$M \vDash \langle P \rangle_{\geq 1} \Leftrightarrow \forall t \in \mathrm{Trace}_M \cdot t \vDash P \quad (10)$$

Definition 5. (Assume-guarantee triple) If $\langle A \rangle_{\geq \mathrm{PA}}$ and $\langle P \rangle_{\geq \mathrm{PG}}$ are probabilistic safety properties, M is a PA and alphabet $\alpha_P \subseteq \alpha_A \cup \alpha_M$, then:

$$\langle A \rangle_{\geq \mathrm{PA}} \mathrm{M} \langle \mathrm{P} \rangle_{\geq \mathrm{PG}} \Leftrightarrow \forall \sigma \in \mathrm{Adv}_{M[\alpha_A]}. \quad (11)$$
$$(\Pr^{\sigma}_{M[\alpha_A]}(A) \geq \mathrm{PA} \Rightarrow \Pr^{\sigma}_{M[\alpha_A]}(P) \geq \mathrm{PG})$$

where $\langle A \rangle_{\geq \mathrm{PA}}$ is also called as assumption and $M[\alpha_A]$ is, as described in Section 2.1, M with its alphabet extended to include α_A.

Determining whether an assume-guarantee triple holds can reduce to multi-objective probabilistic model checking [18,33]. In the absence of an assumption (denoted by ⟨true⟩), checking the triple can reduce to normal model checking:

$$\langle \mathrm{true} \rangle M \langle P \rangle_{\geq PG} \Leftrightarrow M \vDash \langle P \rangle_{\geq PG} \quad (12)$$

2.3. NL* Learning Algorithm

The NL* Learning algorithm [34] is a popular active learning algorithm (since they can ask queries actively) for Residual Finite-State Automata (RFSA) [35,36]. It is developed from L* algorithm, and has some similar features with L* algorithm. It also needs an automaton to accept each unknown regular language, and a Minimally Adequate Teacher (MAT) to answer membership and equivalence queries.

Generally, the RFSA may generate extra nondeterministic choices in the product PA [37] and it is a subclass of Nondeterministic Finite-state Automata (NFA). So, we must transform NFA A into a corresponding DFA A through the standard subset construction algorithm [38]. Although we cannot acquire more succinct assumptions because of the transform step, NL* algorithm may have a faster learning procedure than L* algorithm [23].

3. ASSUME-GUARANTEE REASONING WITH SYM RULE

3.1. Symmetric Rule

At present, compositional stochastic model checking is implemented based on the ASYM [22,23,33,39], which can generate the corresponding assumption for only one component of the system. We present the SYM for the compositional stochastic model checking PAs.

Theorem 1. *Let us suppose that M_1, M_2 are PAs and $\langle A_{M_1}\rangle_{\geq \mathrm{PA}_{M_1}}$, $\langle A_{M_2}\rangle_{\geq \mathrm{PA}_{M_2}}$, $\langle P\rangle_{\geq \mathrm{PG}}$ are probabilistic safety properties. Respectively, their alphabets satisfy $\alpha_{A_{M_1}} \subseteq \alpha_{M_2}$, $\alpha_{A_{M_2}} \subseteq \alpha_{M_1}$ and $\alpha_P \subseteq \alpha_{A_{M_1}} \cup \alpha_{A_{M_2}}$. $co\langle A_{M_1}\rangle_{\geq \mathrm{PA}_{M_1}}$ denote the co-assumption for M_1 which is the complement of $\langle A_{M_1}\rangle_{\geq \mathrm{PA}_{M_1}}$, similarly for $co\langle A_{M_2}\rangle_{\geq \mathrm{PA}_{M_2}}$, the following SYM rule holds:*

$$\frac{\begin{array}{ll} 1: & \langle A_{M_1}\rangle_{\geq \mathrm{PA}_{M_1}} M_1 \langle P\rangle_{\geq \mathrm{PG}} \\ 2: & \langle A_{M_2}\rangle_{\geq \mathrm{PA}_{M_2}} M_2 \langle P\rangle_{\geq \mathrm{PG}} \\ 3: & \mathcal{L}\left(co\langle A_{M_1}\rangle_{\geq \mathrm{PA}_{M_1}} \| co\langle A_{M_2}\rangle_{\geq \mathrm{PA}_{M_2}}\right) = \varnothing \end{array}}{\langle \mathrm{true}\rangle M_1 \| M_2 \langle P\rangle_{\geq \mathrm{PG}}}$$

Theorem 1 indicates that, if each assumption about corresponding component can be acquired, we will be able to decide whether the property $\langle P\rangle_{\geq \mathrm{PG}}$ holds on $M_1 \| M_2$. The particular interpretation of Theorem 1 is shown below.

The meaning of the premise 1 is "whenever M_1 satisfies A_{M_1} with probability at least PA_{M_1}, then it will satisfy P with probability at least PG", $\langle A_{M_1}\rangle_{\geq \mathrm{PA}_{M_1}}$ also indicates these traces with probability at least PA_{M_1} in A_{M_1}. So it can be represented by $\langle A^{\mathrm{err}}_{M_1}\rangle_{<1-\mathrm{PA}_{M_1}}$ (see Section 2.2, $A^{\mathrm{err}}_{M_1}$ is same as P^{err}). The premise 2 is similar to the premise 1.

In the premise 3, the assumption and its complement have the same alphabet. There is no common trace in the composition of the co-assumptions. Note that $co\langle A_{M_1}\rangle_{\geq \mathrm{PA}_{M_1}}$ (i.e., $\langle A_{M_1}\rangle_{<\mathrm{PA}_{M_1}}$) can be represented by $\langle A^{\mathrm{err}}_{M_1}\rangle_{\geq 1-\mathrm{PA}_{M_1}}$.

So an infinite trace can be accepted by $\mathcal{L}(co\langle A_{M_1}\rangle_{\geq \mathrm{PA}_{M_1}} \| co\langle A_{M_2}\rangle_{\geq \mathrm{PA}_{M_2}})$, which can convert into a prefix of the infinite trace is not accepted by $\mathcal{L}\left(\langle A^{\mathrm{err}}_{M_1}\rangle_{\geq 1-\mathrm{PA}_{M_1}} \| \langle A^{\mathrm{err}}_{M_2}\rangle_{\geq 1-\mathrm{PA}_{M_2}}\right)$.

Proof of Theorem 1. We provide the proof of Theorem 1 in the following. This requires Lemma 1, which derives from Kwiatkowska et al. [33].

Lemma 1. Let us suppose that M_1, M_2 are PAs, $\sigma \in \mathrm{Adv}_{M_1\|M_2}$, $y \subseteq \alpha_{M_1\|M_2}$ and $i = 1, 2$. If A is regular safety properties such that $\alpha_A \subseteq \alpha_{M_i[y]}$, then:

$$\Pr^{\sigma}_{M_1\|M_2}(A) = \Pr^{\sigma\lceil_{M_i[y]}}_{M_i[y]}(A) \tag{13}$$

Proof (of Theorem 1). The proof is by contradiction. Assume that the premise 1, 2 and 3 hold, but the conclusion does not. Since $M_1 \| M_2 \nvDash \langle P\rangle_{\geq \mathrm{PG}}$, we will be able to find an adversary $\sigma \in \mathrm{Adv}_{M_1\|M_2}$, such that $\Pr^{\sigma}_{M_1\|M_2}(P) < \mathrm{PG}$. Now, it follows that:

$$\Pr^{\sigma}_{M_1\|M_2}(P) < \mathrm{PG} \tag{14}$$

By Lemma 1 since $\alpha_P \subseteq \alpha_{A_{M_1}} \cup \alpha_{A_{M_2}} \subseteq \alpha_{M_1[\alpha_{A_{M_1}}]}$

$$\Rightarrow \Pr^{\sigma\lceil_{M_1[\alpha_{A_{M_1}}]}}_{M_1[\alpha_{A_{M_1}}]}(P) < \mathrm{PG} \tag{15}$$

by the premise 1 and Definition 5

$$\begin{aligned} &\forall \sigma \in \mathrm{Adv}_{M_1\|M_2}. \\ &(\Pr^{\sigma\lceil_{M_1[\alpha_{A_{M_1}}]}}_{M_1[\alpha_{A_{M_1}}]}(A_{M_1}) \geq \mathrm{PA}_{M_1} \\ &\Rightarrow \Pr^{\sigma\lceil_{M_1[\alpha_{A_{M_1}}]}}_{M_1[\alpha_{A_{M_1}}]}(P) \geq \mathrm{PG}) \end{aligned} \tag{16}$$

by modus tollens since (15) and (16)

$$\Rightarrow \Pr^{\sigma\lceil_{M_1[\alpha_{A_{M_1}}]}}_{M_1[\alpha_{A_{M_1}}]}(A_{M_1}) < \mathrm{PA}_{M_1} \tag{17}$$

Similarly

$$\Pr^{\sigma\lceil_{M_2[\alpha_{A_{M_2}}]}}_{M_2[\alpha_{A_{M_2}}]}(A_{M_2}) < \mathrm{PA}_{M_2} \tag{18}$$

by the premise 3

$$\begin{aligned} &\neg\exists \sigma \in \mathrm{Adv}_{M_1\|M_2} \\ &.\left(\Pr^{\sigma\lceil_{M_1[\alpha_{A_{M_1}}]}}_{M_1[\alpha_{A_{M_1}}]}(A_{M_1}) < \mathrm{PA}_{M_1} \wedge \Pr^{\sigma\lceil_{M_2[\alpha_{A_{M_2}}]}}_{M_2[\alpha_{A_{M_2}}]}(A_{M_2}) < \mathrm{PA}_{M_2}\right) \end{aligned} \tag{19}$$

Our assumption contradicts (19), so this adversary σ is nonexistent. Next, we will use a simple example to illustrate the rule (taken from Kwiatkowska et al. [33]).

Example 1. Figure 1 shows two PAs M_1 and M_2. The switch of a device M_2 is controlled by a controller M_1. Once the emergence of the detect signal, M_1 can send a warn signal before the shutdown signal, but the attempt may be not successful with probability 0.2. M_1 issues the shutdown signal directly, this will lead to the occurrence of a mistake in the device M_2 with probability 0.1 (i.e., M_2 will not shut down correctly). The DFA P^{err} indicates that action fail never occurs. We need to verify whether $M_1 \| M_2 \vDash \langle P\rangle_{\geq 0.98}$ holds.

For checking whether $\langle \mathrm{true}\rangle M_1 \| M_2\ \langle P\rangle_{\geq 0.98}$ holds, we use the rule (SYM) and two probabilistic safety properties $\langle A_{M_1}\rangle_{\geq 0.9}$ and $\langle A_{M_2}\rangle_{\geq 0.8}$ (see Section 3.2 for details) as the assumptions about M_1 and M_2. They are represented by DFA $A^{\mathrm{err}}_{M_1}$ and $A^{\mathrm{err}}_{M_2}$ in Figure 2 (since alphabet $\alpha_{A_{M_1}}$ is same as $\alpha_{A_{M_2}}$, $A^{\mathrm{err}}_{M_1}$ is also same as $A^{\mathrm{err}}_{M_2}$). Note that only state a_2 is in the set of accepting states F (see Section 2.2) and indicates that the safety property P is violated.

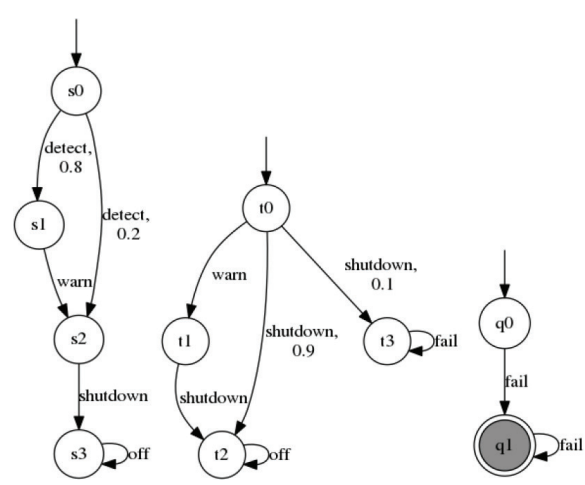

Figure 1 (a) Probabilistic automata M_1, (b) probabilistic automata M_2 and (c) DFA P^{err} for the safety property P.

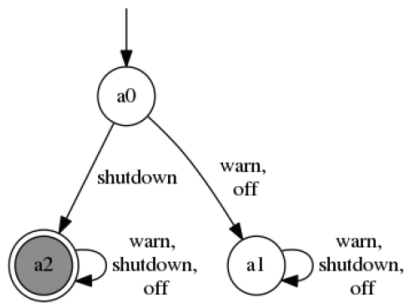

Figure 2 | Assumptions $A^{\text{err}}_{M_1}$, $A^{\text{err}}_{M_2}$ for M_1, M_2.

We can compute the probability of A_{M_1} and A_{M_2} in the premise 1 and 2, because we can solve these queries: $\langle A\rangle_{\geq \text{PA}}\ M\langle P\rangle_{I_G=?}$ and $\langle A\rangle_{I_A=?}\ M\langle P\rangle_{\geq \text{PG}}$, through multi-objective model checking, as shown in Etessami et al. [18] and Kwiatkowska et al. [33]. Actually, if there exists any adversary of the component M that satisfies the strongest assumption $\langle A\rangle_{\geq 1}$ but violate the probabilistic safety property $\langle P\rangle_{\geq \text{PG}}$, the interval I_A will be empty in the second question.

Through premise 3, in $\left\langle A^{\text{err}}_{M_1}\right\rangle_{\geq 0.1}$, we can find a counterexample cex(0.2, ⟨shutdown⟩), but corresponding counterexample in $\left\langle A^{\text{err}}_{M_2}\right\rangle_{\geq 0.2}$ is nonexistent (since action fail exists). So prefixes of all infinite traces in $\left\langle A^{\text{err}}_{M_1}\right\rangle_{\geq 0.1} \| \left\langle A^{\text{err}}_{M_2}\right\rangle_{\geq 0.2}$ can be accepted by $\mathcal{L}\left(\left\langle A^{\text{err}}_{M_1}\right\rangle_{\geq 0.1} \| \left\langle A^{\text{err}}_{M_2}\right\rangle_{\geq 0.2}\right)$ and we can think $M_1 \| M_2 \vDash \langle P\rangle_{\geq 0.98}$ holds. Note that if a trace in $\left\langle A^{\text{err}}_{M_2}\right\rangle_{\geq 0.2}$ corresponding to multiple traces in M_2, we give preference to the trace with action fail. Besides, we can find that the trace ⟨shutdown⟩ is a prefix of ⟨shutdown, warn⟩, ⟨shutdown, shutdown⟩ and ⟨shutdown, off⟩, so there is no need to consider for the last three traces.

3.2. Improved Learning Framework for SYM Rule

Inspired by assume-guarantee verification of PAs [23], we propose an improved learning framework that generates assumptions for compositional stochastic model checking two-component PAs with SYM. The inputs are components M_1, M_2, a probabilistic safety property $\langle P\rangle_{\geq \text{PG}}$ and the alphabets $\alpha_{A_{M_1}}$, $\alpha_{A_{M_2}}$. The aim is to verify whether $M_1 \| M_2 \vDash \langle P\rangle_{\geq \text{PG}}$ by learning assumptions. If these assumptions exist, it can conclude that the $\langle P\rangle_{\geq \text{PG}}$ holds on the system $M_1 \| M_2$. It outperforms [23] in cases the model does not satisfy the properties. Essentially, the original learning framework [23] only searches a counterexample after the conjectured assumption generation. Our method is to search a counterexample in each membership and equivalence query to prove $M_1 \| M_2 \nvDash \langle P\rangle_{\geq \text{PG}}$.

3.2.1. Overview

The NL*-based learning framework for compositional stochastic model checking with rule SYM is shown in Figure 3. Here, the MAT first answers a membership query: whether a given finite trace t_1 should be included in the assumption A_{M_1}. If t_1 is not in the assumption A_{M_1}, we will try to find corresponding traces in M_1 and M_2. If their probability violates the probabilistic safety property $\langle P\rangle_{\geq \text{PG}}$, t_1 will be not a spurious counterexample. We can think the model does not satisfy the property, otherwise continue to answer the next membership query after checking until the appearance of a conjectured assumption A_{M_1}. Then, the MAT answers an equivalence query. Through a multi-objective model checking technique [18,33], we can calculate the probability of a conjectured assumption, which is an interval I_{A_1}. If I_{A_1} is an empty interval, the framework will construct a probabilistic counterexample cex(σ, w, c). σ is an adversary for M_1 with $\Pr^{\sigma}_{M_1}(P) < \text{PG}$. w is a witness for $\langle A_{M_1}\rangle_{\geq \text{PA}_{M_1}}$ (PA_{M_1} is a lower bound of the interval I_{A_1}) in $M_1[\alpha_{A_{M_1}}]$, i.e., a set w of infinite traces in M_1^{σ} is defined as $\Pr(w) \geq \text{PA}_{M_1}$ and $t_1\restriction_{M_1} \vDash A_{M_1}$ for all $t_1 \in w$. A set c of finite traces in M_1^{σ} (i.e., $M_1^{\sigma,c}$) such that $\Pr(c) > 1 - \text{PG}$ and $t_1\restriction_{M_1} \nvDash P$ for all $t_1 \in c$. In short, probabilistic counterexamples are more complex than nonprobabilistic counterexamples. More details are provided in Feng et al. [22] and Ma et al. [40]. Next, we must check whether the appearance of a trace t_1 in the probabilistic counterexample cex(σ, w, c) causes the violation of $\langle P\rangle_{\geq \text{PG}}$ on $M_1 \| M_2$. If the trace exists, the execution of the learning algorithm will be terminated. Otherwise, the learning algorithm will refine the original conjecture and generate a new assumption. When all the conjectured assumptions are successful to be generated, we judge whether there exists any common trace that can be accepted by $\mathcal{L}(co\langle A_{M_1}\rangle_{I_{A_1}} \| co\langle A_{M_2}\rangle_{I_{A_2}})$. It requires us to do Counterexample Analysis. If counterexample does not exist, we can conclude that $M_1 \| M_2 \vDash \langle P\rangle_{\geq \text{PG}}$.

On the contrary, we need to check whether it is a spurious counterexample, let the conjectured assumption becomes stronger than necessary. If the spurious counterexample exists, the conjectured assumption must be refined once again. When the conjectured assumption is updated, the framework will return a lower and an upper bound on the minimum probability of safety property P holding. This measure means that it can provide some valuable information to the user, even if the framework could not produce an accurate judgment. More details are described in the following sections.

3.2.2. Answering membership queries

Minimally adequate teacher is responsible for the membership queries, i.e., checking $t_1 \| M_1 \vDash \langle P\rangle_{\geq \text{PG}}$. t_1 represents the trace in which each transition has probability 1. If trace $t_{M_1} \in M_1$, $t_{M_2} \in M_2$ and $t_{M_1}\restriction_{A_{M_1}} = t_{M_2}\restriction_{A_{M_1}} = t_1$, then P_1 and P_2 are the probability of trace t_{M_1} and t_{M_2} respectively. If the trace t_{M_1} or t_{M_2} has action fail and $P_1 * 1 > 1 - \text{PG}$ (i.e., $t_1 \| M_1 \nvDash \langle P\rangle_{\geq \text{PG}}$), t_1 will not be included in assumption A_{M_1} and it will be in $A^{\text{err}}_{M_1}$. Then, we use t_1 to verify $c \in \mathcal{L}(M_1 \| M_2)$. If $P_1 * P_2 > 1 - \text{PG}$, t_1 will be the counterexample c of $\mathcal{L}(M_1 \| M_2)$. We define cex(σ', c') as a probabilistic counterexample trace, and cex(σ', c') = cex($P_1 * P_2$, c) here. If t_1 is the counterexample c, we can conclude $M_1 \| M_2 \nvDash \langle P\rangle_{\geq \text{PG}}$. Then the learning algorithm is terminated and returns the probabilistic counterexample trace cex(σ', c'). Otherwise, the MAT continues to answer the membership queries, until it produces a conjectured assumption A_{M_1}, similarly for $t_2 \| M_2 \vDash \langle P\rangle_{\geq \text{PG}}$. Note that alphabet $\alpha_{A_{M_1}}$ is same as $\alpha_{A_{M_2}}$ in most cases, because $\alpha_{A_{M_1}}$ and $\alpha_{A_{M_2}}$ all reflect the same safety property P essentially. If $\alpha_{A_{M_1}}$ is same as $\alpha_{A_{M_2}}$, $t_2 \| M_2 \vDash \langle P\rangle_{\geq \text{PG}}$ can be omitted, and A_{M_1} is same as A_{M_2}.

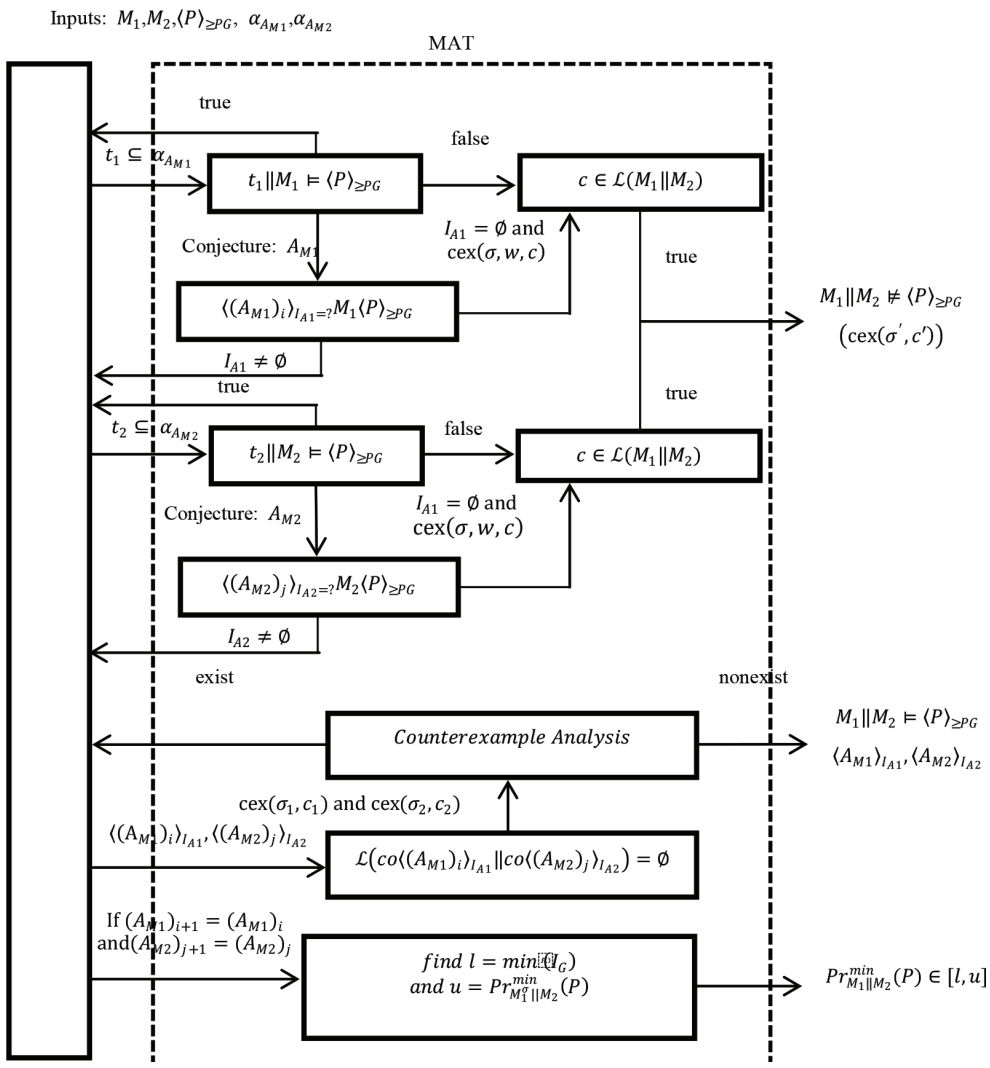

Figure 3 | NL*-based learning framework for the rule SYM.

Example 2. We execute the learning algorithm on PAs M_1, M_2 from Example 1, and the property is set as $\langle P\rangle_{\geq 0.99}$. The alphabet $\alpha_{A_{M_1}}$ is {warn, shutdown, off}, To build its the first conjectured assumption, the algorithm can generate some traces t_1:

⟨warn⟩, ⟨off⟩, ⟨shutdown⟩, ⟨shutdown, shutdown⟩, ⟨shutdown, warn⟩ and ⟨shutdown, off⟩.

The first two return true, i.e., they should be in the conjectured assumption. All of the others return false. Since t_{M_2} has action fail and $P_1 * 1 = 0.2 * 1 > 1 - 0.99 = 0.01$, trace ⟨shutdown⟩ returns false. We can find that the trace ⟨shutdown⟩ is a prefix of ⟨shutdown, shutdown⟩, ⟨shutdown, warn⟩ and ⟨shutdown, off⟩, so they all return false. Since $P_1 * P_2 = 0.2 * 0.1 > (1 - 0.99) = 0.01$, ⟨shutdown⟩ is a counterexample c of the target language $\mathcal{L}(M_1 \| M_2)$, the learning algorithm is terminated and returns the probabilistic counterexample trace cex(0.02, ⟨shutdown⟩).

3.2.3. Answering conjectures for each component

$\langle (A_{M_1})_i \rangle_{I_{A_1}=?} M_1 \langle P\rangle_{\geq PG}$ (i.e., $\langle A_{M_1}\rangle_{\geq PA_{M_1}} M_1 \langle P\rangle_{\geq PG}$ in SYM) can be calculated by multi-objective model checking [18,33]. The widest interval I_{A_1} is defined as $[PA_{M_1}, 1]$ and $PA_{M_1} = 1 - (1 - PG)/P_1$. P_1 is the probability of trace t_{M_1}, if the trace $t_{M_1} \in M_1$ or $t_{M_2} \in M_2$ has action fail and $t_{M_1} \upharpoonright A_{M_1} = t_{M_2} \upharpoonright A_{M_1} = t_1$, $t_1 \in A^{err}_{M_1}$. $i = 1$ indicates that this is the first conjectured assumption $\langle (A_{M_1})_1 \rangle_{I_{A_1}}$. If $I_{A_1} = \emptyset$, even under the conjectured assumption $\langle A_{M_1}\rangle_{\geq 1}$, M_1 still violates $\langle P\rangle_{\geq PG}$. We can construct a probabilistic counterexample cex(σ, w, c) [22,40] to indicate that $\langle A_{M_1}\rangle_{\geq 1} M_1 \langle P\rangle_{\geq PG}$ does not hold. Next, we consider whether the probabilistic counterexample cex(σ, w, c) also belongs to the language $\mathcal{L}(M_1 \| M_2)$, i.e., if cex(σ, w, c) is not a spurious counterexample (through checking $M_1^{\sigma,c} \| M_2 \nvDash \langle P\rangle_{\geq PG}$ [22]), it will prove the conclusion $M_1 \| M_2 \nvDash \langle P\rangle_{\geq PG}$. We can directly obtain a probabilistic counterexample trace cex(σ', c') from cex(σ, w, c). If cex(σ, w, c) is spurious, we need to acquire all traces in the set $T = c \upharpoonright_{A_{M_1}}$. Then, we should find out those traces, which are currently included in the conjectured assumption $\langle (A_{M_1})_1 \rangle_{I_{A_1}}$ but in fact should be excluded, because it violates the properties $\langle P\rangle_{\geq PG}$. In other words, we need to find some bad traces $t_1 = t_{M_1} \upharpoonright_{A_{M_1}}$, $t_{M_1} \in c$, which is not in $A^{err}_{M_1}$. All those traces t_1 will be provided to NL*, and it will produce a conjectured assumption $\langle (A_{M_1})_2 \rangle_{I_{A_1}}$ again. Similarly, we deal with the component M_2.

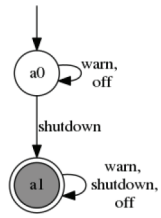

Figure 4 | The first conjectured assumptions $A^{\mathrm{err}}_{M_1}$, $A^{\mathrm{err}}_{M_2}$ for M_1, M_2.

Example 3. We still execute the learning algorithm on PAs M_1, M_2 and property $\langle P\rangle_{\geq 0.98}$ from Example 1. The first conjectured assumptions A_{M_1} and A_{M_2} are represented by $A^{\mathrm{err}}_{M_1}$ and $A^{\mathrm{err}}_{M_2}$ in Figure 4. We can calculate the result $I_{A_1} = [0.9, 1]$, since:

$t_{M_2} = \langle \text{shutdown, fail}\rangle$,

$t_{M_2}\restriction_{A_{M_1}} = \langle \text{shutdown}\rangle = t_{M_1}\restriction_{A_{M_1}}$,

$t_{M_1} = \langle \text{detect, shutdown}\rangle$,

$\mathrm{PA}_{M_1} = 1 - (1 - \mathrm{PG})/P_1 = 1 - (1 - 0.98)/0.2 = 0.9$.

Similarly, since:

$\mathrm{PA}_{M_2} = 1 - (1 - \mathrm{PG})/P_2 = 1 - (1 - 0.98)/0.1 = 0.8$, we can obtain $I_{A_2} = [0.8, 1]$. We cannot find any trace, which is not in $A^{\mathrm{err}}_{M_1}$ or $A^{\mathrm{err}}_{M_2}$, but actually violates the properties $\langle P\rangle_{\geq 0.98}$. So $\langle (A_{M_1})_1\rangle_{[0.9,\ 1]}$ and $\langle (A_{M_2})_1\rangle_{[0.8,\ 1]}$ will be returned to NL* algorithm.

3.2.4. Compositional verification of assumptions

If the interval I_{A_1} and I_{A_2} are nonempty, we will check premise 3 of SYM, we need to verify whether $\mathcal{L}(co\langle (A_{M_1})_i\rangle_{I_{A_1}} \,||\, co\ \langle (A_{M_2})_j\rangle_{I_{A_2}}) = \emptyset$. Here, the conjectured assumption A_{M_1} is the one derived after i iterations of learning, similarly for j. PA_{M_1} is the lower bound of the interval I_{A_1}, similarly for PA_{M_2}.

So $\mathcal{L}(co\langle (A_{M_1})_i\rangle_{I_{A_1}} \,||\, co\ \langle (A_{M_2})_j\rangle_{I_{A_2}})$ can simplify to $\mathcal{L}(co\langle A_{M_1}\rangle_{\geq \mathrm{PA}_{M_1}} \,||\, co\ \langle A_{M_2}\rangle_{\geq \mathrm{PA}_{M_2}})$, which can convert into the problem whether a prefix of the infinite trace is not accepted by $\mathcal{L}\left(\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 1-\mathrm{PA}_{M_1}} \,||\, \left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 1-\mathrm{PA}_{M_2}}\right)$.

Then, counterexample is analyzed by the following process. If the trace $t_1 \in A^{\mathrm{err}}_{M_1}$, we need to find the probability P_{M_1} of the trace t_{M_1}, if and only if $t_{M_1} \in M_1$ and $t_{M_1}\restriction_{A_{M_1}} = t_1$. If t_{M_1} is not unique, we will first return the trace with action fail. If it is nonexistent, we will return the trace with minimum probability for all t_{M_1}. When the returned trace has action fail, the spurious counterexample trace $\mathrm{cex}(\sigma_1, c_1) = \mathrm{cex}(P_{M_1}, t_1)$ will not exist in $\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 1-\mathrm{PA}_{M_1}}$, otherwise it will exist. Note that $\mathrm{cex}(\sigma_1, c_1)$ cannot prove $M_1||M_2 \not\models \langle P\rangle_{\geq \mathrm{PG}}$ and it indicates that a trace satisfies the property $\langle P\rangle_{\geq \mathrm{PG}}$ in $\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 1-\mathrm{PA}_{M_1}}$ essentially. So we call it as spurious counterexample trace. Similarly, we return the $\mathrm{cex}(\sigma_2, c_2) = \mathrm{cex}(P_{M_2}, t_2)$ as spurious counterexample trace in $\left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 1-\mathrm{PA}_{M_2}}$. When $\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 1-\mathrm{PA}_{M_1}}$ and $\left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 1-\mathrm{PA}_{M_2}}$ all have spurious counterexample trace, the spurious counterexample trace $\mathrm{cex}(\sigma, c) = \mathrm{cex}(P_{M_1} * P_{M_2}, t_1||t_2)$ will may exist in $\mathcal{L}\left(\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 1-\mathrm{PA}_{M_1}} \,||\, \left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 1-\mathrm{PA}_{M_2}}\right)$.

Next, if $P_{M_1} * P_{M_2} > 1 - \mathrm{PG}$, a prefix of the infinite trace is not accepted by $\mathcal{L}\left(\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 1-\mathrm{PA}_{M_1}} \,||\, \left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 1-\mathrm{PA}_{M_2}}\right)$. So we need to use the spurious counterexample traces $\mathrm{cex}(\sigma_1, c_1)$ and $\mathrm{cex}(\sigma_2, c_2)$ to weaken the corresponding assumptions, i.e., t_1 and t_2 will be added in the assumption A_{M_1} and A_{M_2} respectively, then the conjectured assumptions must be refined once again. Otherwise, if $P_{M_1} * P_{M_2} \leq 1 - \mathrm{PG}$, it will be not a spurious counterexample trace in $\mathcal{L}\left(\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 1-\mathrm{PA}_{M_1}} \,||\, \left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 1-\mathrm{PA}_{M_2}}\right)$.

Finally, if any spurious counterexample trace in $\mathcal{L}\left(\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 1-\mathrm{PA}_{M_1}} \,||\, \left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 1-\mathrm{PA}_{M_2}}\right)$ is nonexistent, we can obtain two assumptions $\langle A_{M_1}\rangle_{I_{A_1}}$ and $\langle A_{M_2}\rangle_{I_{A_2}}$ to prove $M_1||M_2 \models \langle P\rangle_{\geq \mathrm{PG}}$.

Example 4. We continue the execution of the algorithm from Example 3. We must do counterexample analysis for it. Intuitively, we can find a spurious counterexample trace $\mathrm{cex}(0.8, \langle \text{warn, shutdown}\rangle)$ in $\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 0.1}$ and $\mathrm{cex}(1, \langle \text{warn, shutdown}\rangle)$ in $\left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 0.2}$.

Since $0.8 * 1 = 0.8$, we can find that the spurious counterexample trace in $\mathcal{L}\left(\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 1-\mathrm{PA}_{M_1}} \,||\, \left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 1-\mathrm{PA}_{M_2}}\right)$ may be $\mathrm{cex}(0.8, \langle \text{warn, shutdown}\rangle)$. Since $0.8 > 1 - 0.98 = 0.02$, $\mathrm{cex}(0.8, \langle \text{warn, shutdown}\rangle)$ is the spurious counterexample trace of $\mathcal{L}\left(\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 0.1} \,||\, \left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 0.2}\right)$ and the trace $\langle \text{warn, shutdown}\rangle$ cannot be accepted by $\mathcal{L}\left(\left\langle A^{\mathrm{err}}_{M_1}\right\rangle_{\geq 0.1} \,||\, \left\langle A^{\mathrm{err}}_{M_2}\right\rangle_{\geq 0.2}\right)$. So we use the spurious counterexample trace to weaken the corresponding assumption, i.e., the trace $\langle \text{warn, shutdown}\rangle$ needs to be added to the corresponding assumption. The second conjectured assumption A_{M_1} (A_{M_2} is same as A_{M_1}) is shown in Figure 2, which can prove $M_1||M_2 \models \langle P\rangle_{\geq 0.98}$.

3.2.5. Generation of lower and upper bounds

In each iteration of the NL* algorithm, we can obtain the tightest bounds from the iterative process of assumptions (show in the bottom of Figure 3). If the learning framework cannot provide a definitive result (i.e., the runtime is more than the waiting time), some valuable quantitative information will be returned. For each conjectured assumption, we have a lower bound $\mathrm{lb}(A, P)$ and an upper bound $\mathrm{ub}(A, P)$ on the probabilistic safety property P.

We can calculate $p^*_A = \min(\mathrm{Pr}^{\min}_{M_1}(A_{M_1}), \mathrm{Pr}^{\min}_{M_2}(A_{M_2}))$ and generate a corresponding adversary $\sigma \in \mathrm{Adv}_M$ (M is the component about selected assumption), then we compute $\langle A\rangle_{\geq p^*_A} M\langle P\rangle_{I_G=?}$ through multi-objective model checking [18,33].

For the interval $\mathrm{lb}(A, P) \leq \mathrm{Pr}^{\min}_{M_1||M_2}(P) \leq \mathrm{ub}(A, P)$, we have:

$$\mathrm{lb}(A, P) = \min(I_G) \tag{20}$$

$$\mathrm{ub}(A, P) = \mathrm{Pr}^{\min}_{M_1^{\sigma}||M_2}(P), (\text{if } \sigma \in \mathrm{Adv}_{M_1}) \tag{21}$$

The proof of the tightest bounds is similar to Feng et al. [22]. Note that information generation of bounds may lead to little extra work.

4. ASSUME-GUARANTEE REASONING WITH SYM-N RULE

4.1. Symmetric Rule

We present a symmetric assume-guarantee rule SYM in the previous section, which can solve the problem of verification of a stochastic system about two components. Here, we will make an extension to it. Let it can be used to verify a stochastic system composed of $n \geq 2$ components: $M_1 \| M_2 \| \cdots \| M_n$.

Theorem 2. *Let $M_1, M_2, \ldots, M_n$ are PAs, for $i \in \{1, 2, \ldots, n\}$, $\langle A_{M_i} \rangle_{\geq \mathrm{PA}_{M_i}}$ is an assumption for the corresponding component M_i, $\langle P \rangle_{\geq \mathrm{PG}}$ is a probabilistic safety property. Their alphabets satisfy $\alpha_{A_{M_i}} \subseteq \alpha_{M_1} \cup \cdots \cup \alpha_{M_{i-1}} \cup \alpha_{M_{i+1}} \cup \cdots \cup \alpha_{M_n}$, and $\alpha_P \subseteq \alpha_{A_{M_1}} \cup \alpha_{A_{M_2}} \cup \cdots \cup \alpha_{A_{M_n}}$ respectively. $co\langle A_{M_i} \rangle_{\geq \mathrm{PA}_{M_i}}$ denotes the co-assumption for M_i which is the complement of $\langle A_{M_i} \rangle_{\geq \mathrm{PA}_{M_i}}$, the following SYM-N rule holds:*

$$
\begin{array}{ll}
1: & \left\langle A_{M_1} \right\rangle_{\geq \mathrm{PA}_{M_1}} M_1 \langle P \rangle_{\geq \mathrm{PG}} \\
2: & \left\langle A_{M_2} \right\rangle_{\geq \mathrm{PA}_{M_2}} M_2 \langle P \rangle_{\geq \mathrm{PG}} \\
\vdots & \\
n: & \left\langle A_{M_n} \right\rangle_{\geq \mathrm{PA}_{M_n}} M_n \langle P \rangle_{\geq \mathrm{PG}} \\
n+1: & \mathcal{L}\left(co\left\langle A_{M_1} \right\rangle_{\geq \mathrm{PA}_{M_1}} \| co\left\langle A_{M_2} \right\rangle_{\geq \mathrm{PA}_{M_2}} \| \cdots \| co\left\langle A_{M_n} \right\rangle_{\geq \mathrm{PA}_{M_n}} \right) = \varnothing \\
\hline
 & \langle \mathrm{true} \rangle M_1 \| M_2 \| \cdots \| M_n \langle P \rangle_{\geq \mathrm{PG}}
\end{array}
$$

Proof by contradiction. Assume that the premise 1, 2, …, $n + 1$ hold, but the conclusion does not. We can obtain an adversary $\sigma \in \mathrm{Adv}_{M_1 \| M_2 \| \cdots \| M_n}$, such that $\Pr^{\sigma}_{M_1 \| M_2 \| \cdots \| M_n}(P) < \mathrm{PG}$. Now, it follows that:

$$\Pr^{\sigma}_{M_1 \| M_2 \| \cdots \| M_n}(P) < \mathrm{PG} \tag{22}$$

by Lemma 1 since $\alpha_P \subseteq \alpha_{A_{M_1}} \cup \alpha_{A_{M_2}} \cup \cdots \cup \alpha_{A_{M_n}} \subseteq \alpha_{M_1[\alpha_{A_{M_1}}]}$

$$\Rightarrow \Pr^{\sigma\restriction_{M_1[\alpha_{A_{M_1}}]}}_{M_1[\alpha_{A_{M_1}}]}(P) < \mathrm{PG} \tag{23}$$

by the premise 1 and Definition 5

$$\forall \sigma \in \mathrm{Adv}_{M_1 \| M_2 \| \cdots \| M_n} \cdot \left(\Pr^{\sigma\restriction_{M_1[\alpha_{A_{M_1}}]}}_{M_1[\alpha_{A_{M_1}}]}(A_{M_1}) \geq \mathrm{PA}_{M_1} \Rightarrow Pr^{\sigma\restriction_{M_1[\alpha_{A_{M_1}}]}}_{M_1[\alpha_{A_{M_1}}]}(P) \geq \mathrm{PG} \right) \tag{24}$$

by modus tollens since (23) and (24)

$$\Rightarrow \Pr^{\sigma\restriction_{M_1[\alpha_{A_{M_1}}]}}_{M_1[\alpha_{A_{M_1}}]}(A_{M_1}) < \mathrm{PA}_{M_1} \tag{25}$$

Similarly

$$\Pr^{\sigma\restriction_{M_i[\alpha_{AM_i}]}}_{M_i[\alpha_{AM_i}]}(A_{M_i}) < \mathrm{PA}_{M_i}, i \in \{2, 3, \ldots, n\} \tag{26}$$

by the premise $n + 1$

$$\neg \exists \sigma \in \mathrm{Adv}_{M_1 \| M_2 \| \cdots \| M_n} \cdot \left(\Pr^{\sigma\restriction_{M_1[\alpha_{A_{M_1}}]}}_{M_1[\alpha_{A_{M_1}}]}(A_{M_1}) < \mathrm{PA}_{M_1} \wedge \cdots \wedge \Pr^{\sigma\restriction_{M_n[\alpha_{A_{M_n}}]}}_{M_n[\alpha_{A_{M_n}}]}(A_{M_n}) < \mathrm{PA}_{M_n} \right) \tag{27}$$

Our assumption contradicts (27), so this adversary σ is nonexistent. Next, we will use Example 5 to explain the rule.

Example 5. The example is the extension of Example 1. Figure 5 shows three PAs M_1, M_2, M_3 and a probabilistic safety property $\langle P \rangle_{\geq 0.98}$. The component M_2 indicates that the time signal may reappear with probability 0.5 before the shutdown signal. We will show the verification process by the method of SYM-N rule.

Similar to Example 1, through multi-objective model checking [18,33], we can acquire three assumptions $\langle A \rangle_{M_1 \geq 0.9}$, $\langle A \rangle_{M_2 \geq 1}$ and $\langle A \rangle_{M_3 \geq 0.8}$, which are represented by DFA $A^{\mathrm{err}}_{M_1}$, $A^{\mathrm{err}}_{M_2}$ and $A^{\mathrm{err}}_{M_3}$ in Figure 6.

Through premise $n + 1$, we can find a spurious counterexample trace cex(0.2, ⟨shutdown⟩) in $\left\langle A^{\mathrm{err}}_{M_1} \right\rangle_{\geq 0.1}$ and cex(1, ⟨shutdown⟩) in $\left\langle A^{\mathrm{err}}_{M_2} \right\rangle_{\geq 0}$, but corresponding spurious counterexample trace in $\left\langle A^{\mathrm{err}}_{M_3} \right\rangle_{\geq 0.2}$ is nonexistent (since action fail exists). So prefixes of all infinite traces in $\left\langle A^{\mathrm{err}}_{M_1} \right\rangle_{\geq 0.1} \| \left\langle A^{\mathrm{err}}_{M_2} \right\rangle_{\geq 0} \| \left\langle A^{\mathrm{err}}_{M_3} \right\rangle_{\geq 0.2}$ can be accepted by $\mathcal{L}\left(\left\langle A^{\mathrm{err}}_{M_1} \right\rangle_{\geq 0.1} \| \left\langle A^{\mathrm{err}}_{M_2} \right\rangle_{\geq 0} \| \left\langle A^{\mathrm{err}}_{M_3} \right\rangle_{\geq 0.2} \right)$ and we can think $M_1 \| M_2 \| M_3 \models \langle P \rangle_{\geq 0.98}$ holds.

4.2. Improved Learning Framework for SYM-N Rule

The NL*-based learning framework in Figure 7 can be used for verifying a stochastic system composed of $n \geq 2$ components: $M_1 \| M_2 \| \cdots \| M_n$. We first answer membership queries through solving the problem $t_j \| M_j \models \langle P \rangle_{\geq \mathrm{PG}}$, for $j \in \{1, 2, ..., n\}$. The process is similar to Section 3.2.2 but it is a little different. In Counterexample Analysis for Membership Queries, if $t_j \| M_j \not\models \langle P \rangle_{\geq \mathrm{PG}}$, the framework will verify whether t_j is a counterexample c of the target language

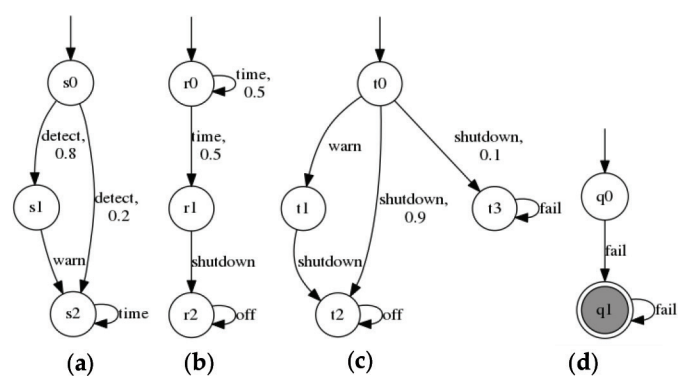

Figure 5 | (a) Probabilistic automata M_1, (b) Probabilistic automata M_2, (c) Probabilistic automata M_3 and (d) DFA P^{err} for the property P.

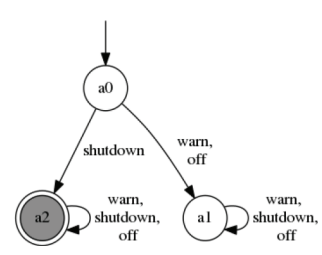

Figure 6 | Assumptions $A^{\mathrm{err}}_{M_1}$, $A^{\mathrm{err}}_{M_2}$, $A^{\mathrm{err}}_{M_3}$ for M_1, M_2, M_3.

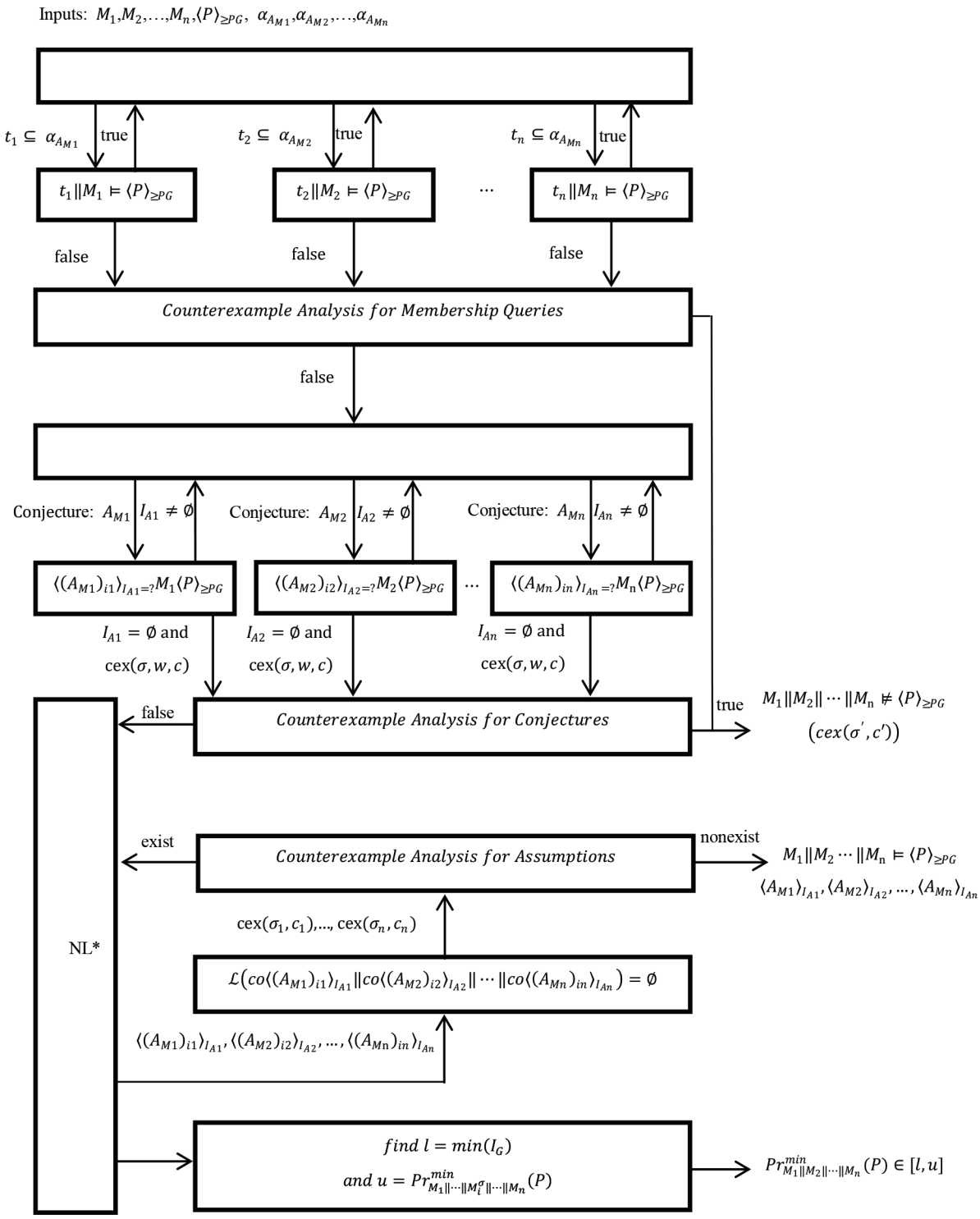

Figure 7 | NL*-based learning framework for the rule SYM-N.

$\mathcal{L}(M_1 \| M_2 \| \cdots \| M_n)$. If t_j is the counterexample c, the framework will return the trace t_j and the product of the probabilities of corresponding traces in all components as $\mathrm{cex}(\sigma', c')$, and we can find that the property is violated, i.e., $M_1 \| M_2 \| \cdots \| M_n \nvDash \langle P\rangle_{\geq \mathrm{PG}}$. Then, we need to answer equivalence queries through tackling the problem $\langle (A_{M_j})_{ij}\rangle_{I_{Aj}=?} M_j \langle P\rangle_{\geq \mathrm{PG}}$, ij indicates the number of iterations about the assumption A_{M_j} and the process of solving the problem shows in Section 3.2.3.

In Counterexample Analysis for Conjectures, the framework will check if the counterexample $\mathrm{cex}(\sigma, w, c)$ belongs to the target language $\mathcal{L}(M_1 \| M_2 \| \cdots \| M_n)$. The problem can transform into checking whether $M_1 \| \cdots \| M_j^{\sigma, c} \| \cdots \| M_n \nvDash \langle P\rangle_{\geq \mathrm{PG}}$ holds, similarly to Feng et al. [22]. Next, the framework needs to verify $\mathcal{L}(co\langle (A_{M_1})_{i_1}\rangle_{I_{A_1}} \| co\langle (A_{M_2})_{i_2}\rangle_{I_{A_2}} \| \cdots \| co\langle (A_{M_n})_{i_n}\rangle_{I_{A_n}}) = \emptyset$. It can simplify to find a trace that can be accepted by: $\mathcal{L}(co\langle A_{M_1}\rangle_{\geq \mathrm{PA}_{M_1}} \| co\langle A_{M_2}\rangle_{\geq \mathrm{PA}_{M_2}} \| \cdots \| co\langle A_{M_n}\rangle_{\geq \mathrm{PA}_{M_n}})$, and convert into finding a prefix of the infinite trace is not accepted by:

$$\mathcal{L}\left(\left\langle A_{M_1}^{\mathrm{err}}\right\rangle_{\geq 1-\mathrm{PA}_{M_1}} \| \left\langle A_{M_2}^{\mathrm{err}}\right\rangle_{\geq 1-\mathrm{PA}_{M_2}} \| \cdots \| \left\langle A_{M_n}^{\mathrm{err}}\right\rangle_{\geq 1-\mathrm{PA}_{M_n}}\right)$$

In Counterexample Analysis for Assumptions, if we cannot find any spurious counterexample trace, $\mathcal{L}(co\langle(A_{M_1})_{i_1}\rangle_{I_{A_1}} \| co\langle(A_{M_2})_{i_2}\rangle_{I_{A_2}} \| \cdots \| co\langle(A_{M_n})_{i_n}\rangle_{I_{A_n}})$ will be empty and the framework will return assumptions $\langle A_{M_1}\rangle_{I_{A_1}}, \langle A_{M_2}\rangle_{I_{A_2}}, \cdots, \langle A_{M_n}\rangle_{I_{A_n}}$ to prove that the property is satisfied, *i.* $M_1 \| M_2 \| \cdots \| M_n \vDash \langle P\rangle_{\geq \mathrm{PG}}$. On the contrary, we need to use the spurious counterexample traces to weaken the corresponding assumptions. We no longer go into details here.

The framework also can return the tightest bounds of the property P satisfied over the system $M_1 \| M_2 \| \cdots \| M_n$ from the iterative process of assumptions. We can calculate:

$$p_A^* = \min(\mathrm{Pr}_{M_1}^{\min}(A_{M_1}), \mathrm{Pr}_{M_2}^{\min}(A_{M_2}), \ldots, \mathrm{Pr}_{M_n}^{\min}(A_{M_n}))$$

and generate a corresponding adversary $\sigma \in \mathrm{Adv}_{M_i}$, for $i \in \{1, 2, \ldots, n\}$. Then, we compute $\langle A\rangle_{\geq p_A^*} M \langle P\rangle_{I_G=?}$ through multi-objective model checking [18,33]. In the end, the lower bound $\mathrm{lb}(A, P)$ is $\min(I_G)$ and the upper bound $\mathrm{ub}(A, P)$ is $\mathrm{Pr}_{M_1 \| \cdots \| M_i^\sigma \| \cdots \| M_n}^{\min}(P)$.

5. RESULTS

As shown in Figure 8, we have developed a prototype tool for our learning framework. It accepts a model and corresponding property as inputs and returns the verification result. Verification result can be classified into three categories:

(1) Some assumptions are provided to prove that model satisfies the property.

(2) Counterexample trace $\mathrm{cex}(\sigma', c')$ is provided to prove that model violates the property.

(3) Bounds of which the property P holds are provided, if the appropriate assumption or counterexample cannot be obtained.

We use PRISM [25] and counterexample construction algorithm (i.e., particle swarm optimization algorithm [40]) to form a MAT. Then through the libalf [41] learning library, we can implement the NL* algorithm and pose membership and equivalence queries to a MAT. The MAT uses the PRISM modeling language to describe models and probabilistic safety properties. In the interior of the MAT, PRISM can provide the transition matrix (indicate that the transition relation of states in the model) and failure states (indicate that a property is violated) to counterexample construction algorithm. The algorithm can find all shortest paths of the same length and calculate the probability of each path, to construct probabilistic counterexamples. Through constructed counterexamples, we can respond to these queries of libalf. All experiments are run on a 3.3 GHz PC with 8 GB RAM. Feng et al. [22] uses the L* learning algorithm to produce the probabilistic assumptions. On this basis, Feng et al. [23] proves that NL* learning algorithm has more efficient than L* in large-scale cases. Our method thus is based on NL*. We use several large cases to demonstrate our learning framework and compare with the method of Feng et al. [23]. We adopt the first two cases form [23], and modify them a little, because we focus on the conditions that the model does not satisfy the properties. To ensure the correctness of the experimental results, we change the cases through different means. The first case is a network of N sensors. In the network, a channel can issue some data to a processor, but it may crash because some data packets are lost. Through the SYM rule, we make the composition of the N sensors and a channel as a component M_1, the processor as the other component M_2. We will verify the probabilistic safety property, i.e., network never crashes with a certain probability. We will increase the probability of probabilistic safety property to satisfy our experimental requirements, and the verified property is $\langle P\rangle_{\geq 0.994}$. Table 1 shows experimental results for the sensor network.

The second case is the client–server model studied from Pasareanu et al. [42]. Feng et al. [23] injects (probabilistic) failures into one or more of the N clients and changes the model into a stochastic system. In client–server model, each client can send requests for

Table 1 | Sensor network experimental results

Case study [sensor network]	Sensor numbers	Component sizes		SYM		ASYM [23]	
		$\lvert M_1\rvert$	$\lvert M_2\rvert$	*MQ*	Time(s)	*MQ*	Time(s)
	1	72	32	16	1.5	25	2.7
	2	1184	32	16	1.8	25	2.9
	3	10662	32	16	2.4	25	3.9

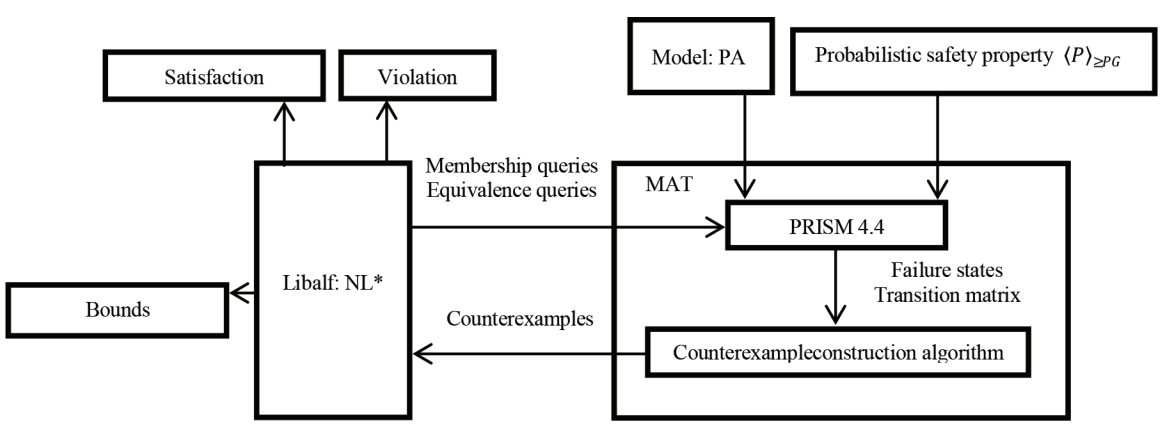

Figure 8 | Prototype tool.

reservations to use a common resource, the server can grant or deny a client's request, and the model must satisfy the mutual exclusion property (i.e., conflict in using resources between clients) with certain minimum probability. Through the SYM rule, we make the server as a component M_1 and the composition of N clients as the other component M_2. The verified property is $\langle P \rangle_{\geq 0.9}$. We use the method of Feng et al. [23] to inject (nonprobabilistic and probabilistic) failures into the server respectively. Table 2 shows experimental results for the client–server.

To consider the case where the model satisfies the properties, the last case is randomized consensus algorithm from Feng et al. [23] without modification. The algorithm models N distributed processes trying to reach consensus and uses, in each round, a shared coin protocol parameterized by K. The verified property is $\langle P \rangle_{\geq 0.97504}$, and 0.97504 is the minimum probability of consensus being reached within R rounds. Through the SYM rule, the system is decomposed into two PA components: M_1 for the coin protocol and M_2 for the interleaving of N processes.

In Tables 1 and 2, the component sizes of the M_1 and M_2 are denoted as $|M_1|$ and $|M_2|$, and the performance is measured by the total number of Membership Queries (MQ) and runtimes (Time). Note that Time includes counterexample construction, NFA translation and the learning process. Moreover, for the accuracy of the results, we select the counterexamples in the same order as Feng et al. [23] in each equivalence query. Note that Feng et al. [23] has included comparisons with non-compositional verification, so this paper only compares with Feng et al. [23].

As shown in Tables 1 and 2, the experiment results show that our framework is more efficient than Feng et al. [23]. Obviously, we can observe that, for all cases, runtimes and the number of the membership queries in our framework are less than Feng et al. [23]. Moreover, the runtimes need less in our framework, when the model has a large scale. A larger size model may have less runtimes and the number of membership queries than a smaller model. However, this is not proportion with the model size. The efficiency of our framework depends only on the time of a counterexample (indicate that the probabilistic safety property is violated) appears in conjectured assumptions. The earlier a counterexample appears, the more efficient our framework performs.

In Table 3, the component sizes of the M_1 and M_2 is also denoted as $|M_1|$ and $|M_2|$. The performance is measured only by total runtimes (Time), because both methods have the same amount of MQ if the model satisfies the properties. Because of the cost of early detection, we can find that our methods need to spend more time than Feng et al. [23] and cost grows with the model size. But compared with acquirement of optimization in Tables 1 and 2, the cost is acceptable in Table 3.

Table 4 compares the performance of the rule (SYM) and the rule (SYM-N). We impose a time-out of 5 h. Sensor network model has N sensors and client–server model has N clients. In client–server model, each client and server all have a (probabilistic) failure. For the use of rule (SYM-N), we decompose M_1 into separate sensor and compose each sensor and a channel as a component in sensor network model, and decompose M_2 further into separate client in client–server model. Moreover, the performance is measured by the total runtimes (Time). In all large cases, the rule (SYM-N) has more advantage than the rule (SYM). For example, in the case of sensor network model with four sensors, the component M_1 has 72776 states and the component M_2 has 32 states. The total runtime of the compositional verification by the rule (SYM) more than 5 h, but the use of the rule (SYM-N) only needs 16.6 s. This is because the size of the component M_1 is too large for the rule (SYM), and the counterexample construction algorithm needs more time to give the conclusion.

Table 3 | Randomized consensus algorithm experimental results

Case study [consensus]	[N R K]	Component sizes		SYM	ASYM [23]
		$\|M_1\|$	$\|M_2\|$	Time (s)	Time (s)
	2 3 20	3217	389	12.1	11.6
	2 4 4	431649	571	82.2	80.7
	3 3 20	38193	8837	355.8	350.2

Table 4 | Performance comparison of the rule (SYM) and the rule (SYM-N)

Case study [parameters]		Component sizes		SYM	SYM-N
		$\|M_1\|$	$\|M_2\|$	Time (s)	Time (s)
Sensor network [N]	4	72776	32	Time-out	16.6
	5	428335	32	Time-out	40.7
Client–server [N]	6	49	15625	Time-out	20.4
	7	64	78125	Time-out	80.9

Table 2 | Client–server experimental results

Case study [client–server]	Client numbers	Component sizes		SYM		ASYM [23]	
		$\|M_1\|$	$\|M_2\|$	*MQ*	Time (s)	*MQ*	Time (s)
Server (nonprobability) Client (1 failure)	3	16	45	100	2.5	161	5.2
	5	36	405	325	6.9	519	12.4
	7	64	3645	833	63.1	1189	140.1
Server (nonprobability) Client (N failures)	3	16	125	175	4.6	213	5.9
	4	25	625	336	8.3	393	11.4
	5	36	3125	226	4.9	648	18.1
Server (probability) Client (1 failure)	3	16	45	120	0.31	187	5.7
	5	36	405	379	7.8	583	16.4
	7	64	3645	937	28.1	1308	45.5
Server (probability) Client (N failure)	3	16	125	176	3.9	265	6.6
	4	25	625	337	7.4	507	12.2
	5	36	3125	568	66.2	839	90.3

6. DISCUSSION

We first present a sound SYM for compositional stochastic model checking. Then, we propose a learning framework for compositional stochastic model checking PAs with rule SYM, based on the optimization of LAGR techniques. Our optimization can terminate the learning process in advance, if a counterexample appears in any membership and equivalence query. We also extend the framework to support the assume-guarantee rule SYM-N which can be used for reasoning about a stochastic system composed of $n \geq 2$ components: $M_1||M_2|| \cdots ||M_n$. Experimental results show that our method can improve the efficiency of the original learning framework [23]. Similar to Feng et al. [22] and Kwiatkowska et al. [33], it can return the tightest bounds for the safety property as a reference as well.

In the future, we intend to develop our learning framework to produce richer classes of probabilistic assumption (for example weighted automata as assumptions [39]) and extend it to deal with more expressive types of probabilistic models.

CONFLICTS OF INTEREST

The author declare they have no conflicts of interest.

ACKNOWLEDGMENTS

This work was supported by the Six Talent Peaks Project of Jiangsu (No. RJFW-014), National Natural Science Foundation of China (61303022), Natural Science Major Project of Jiangsu Higher Education Institutions (17KJA520002), and Nanjing Scientific & Technological Innovation Project for Outstanding Overseas Returnees.

REFERENCES

[1] E.M. Clarke, E.A. Emerson, Design and synthesis of synchronization skeletons using branching time temporal logic, in: D. Kozen (Eds.), Workshop on logics of programs, Lecture Notes in Computer Science, vol. 131, Springer, Berlin, Heidelberg, 1981, pp. 52–71.

[2] J.P. Queille, J. Sifakis, Specification and verification of concurrent systems in CESAR, in: M. Dezani-Ciancaglini, U. Montanari (Eds.), International Symposium on Programming, Lecture Notes in Computer Science, vol. 137, Springer, Berlin, Heidelberg, 1982, pp. 337–351.

[3] C. Baier, J-P. Katoen, Principles of Model Checking, MIT Press, Cambridge, UK, 2008.

[4] M. Kwiatkowska, G. Norman, D. Parker, Stochastic model checking, in: M. Bernardo, J. Hillston (Eds.), Formal Methods for Performance Evaluation (SFM), Lecture Notes in Computer Science, vol. 4486, Springer, Berlin, Heidelberg, 2007, pp. 220–270.

[5] V. Forejt, M. Kwiatkowska, G. Norman, D. Parker, Automated verification techniques for probabilistic systems, in: M. Bernardo, V. Issarny (Eds.), Formal Methods for Eternal Networked Software Systems (SFM), Lecture Notes in Computer Science, vol. 6659, Springer, Berlin, Heidelberg, 2011, pp. 53–113.

[6] G.D. Penna, B. Intrigila, I. Melatti, E. Tronci, M.V. Zilli, Bounded probabilistic model checking with the murα verifier, in: A.J. Hu, A.K. Martin (Eds.), Formal Methods in Computer-Aided Design (FMCAD), Lecture Notes in Computer Science, vol. 3312, Springer, Berlin, Heidelberg, 2004, pp. 214–229.

[7] E. Clarke, O. Grumberg, S. Jha, et al., Counterexample-guided abstraction refinement, in: E.A. Emerson, A.P. Sistla (Eds.), Computer Aided Verification (CAV), Lecture Notes in Computer Science, vol. 1855, Springer, Berlin, Heidelberg, 2000, pp. 154–169.

[8] H. Barringer, R. Kuiper, A. Pnueli, Now you may compose temporal logic specifications, in: Sixteenth Annual ACM Symposium on the Theory of Computing (STOC), ACM, New York, NY, USA, 1984, pp. 51–63.

[9] A. Pnueli, In transition from global to modular temporal reasoning about programs, in: K.R. Apt (Eds.), Logics and models of Concurrent Systems, NATO ASI Series (Series F: Computer and Systems Sciences), vol. 13, Springer, Berlin, Heidelberg, 1985, pp. 123–144.

[10] B. Meyer, Applying 'Design by Contract', Computer 25 (1992), 40–51.

[11] S. Bensalem, M. Bogza, A. Legay, T.H. Nguyen, J. Sifakis, R. Yan, Incremental component-based construction and verification using invariants, in: Formal Methods in Computer Aided Design (FMCAD), IEEE, Piscataway, NJ, 2010, pp. 257–256.

[12] H. Barringer, C.S. Păsăreanu, D. Giannakopolou, Proof rules for automated compositional verification through learning, in: Proc. of the 2nd International Workshop on Specification and Verification of Component Based Systems, 2003, pp. 14–21.

[13] M.G. Bobaru, C.S. Păsăreanu, D. Giannakopoulou, Automated assume-guarantee reasoning by abstraction refinement, in: A. Gupta, S. Malik (Eds.), Computer Aided Verification (CAV), Lecture Notes in Computer Science, vol. 5123, Springer, Berlin, Heidelberg, 2008, pp. 135–148.

[14] J.M. Cobleigh, D. Giannakopoulou, C.S. Păsăreanu, Learning assumptions for compositional verification, in: H. Garavel, J. Hatcliff (Eds.), Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lecture Notes in Computer Science, vol. 2619, Springer, Berlin, Heidelberg, 2003, pp. 331–346.

[15] O. Grumberg, D.E. Long, Model checking and modular verification, ACM Trans. Program. Lang. Syst. 16 (1994), 843–871.

[16] R. Segala, Modeling and verification of randomized distributed real-time systems, Department of Electrical Engineering and Computer Science, MIT, 1995 (Also appears as Technical Report MIT/LCS/TR–676).

[17] M. Kwiatkowska, G. Norman, D. Parker, H. Qu, Assume-guarantee verification for probabilistic systems, in: J. Esparza, R. Majumdar (Eds.), Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lecture Notes in Computer Science, vol. 6015, Springer, Berlin, Heidelberg, 2010, pp. 23–37.

[18] K. Etessami, M. Kwiatkowska, M.Y. Vardi, M. Yannakakis, Multi-objective model checking of Markov decision processes, in: O. Grumberg, M. Huth (Eds.), Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lecture Notes in Computer Science, vol. 4424, Springer, Berlin, Heidelberg, 2007, pp. 50–65.

[19] R. Bouchekir, M.C. Boukala, Learning-based symbolic assume-guarantee reasoning for Markov decision process by using interval Markov process, Innov. Syst. Softw. Eng. 14 (2018), 229–244.

[20] F. He, X. Gao, B-Y. Wang, L. Zhang, Leveraging Weighted Automata in Compositional Reasoning about Concurrent Probabilistic Systems, 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, 2015, pp. 503–514.

[21] A. Komuravelli, C.S. Păsăreanu, E.M. Clarke, Assume-guarantee abstraction refinement for probabilistic systems, in: P. Madhusudan, S.A. Seshia (Eds.), Computer Aided Verification,

(CAV) Lecture Notes in Computer Science, vol. 7358, Springer, Berlin, Heidelberg, 2012, pp. 310–326.

[22] L. Feng, M. Kwiatkowska, D. Parker, Compositional verification of probabilistic systems using learning, in: 2010 Seventh International Conference on the Quantitative Evaluation of Systems, IEEE, Williamsburg, VA, USA, 2010, pp. 133–142.

[23] L. Feng, M. Kwiatkowska, D. Parker, Automated learning of probabilistic assumptions for compositional reasoning, in: D. Giannakopoulou, F. Orejas (Eds.), Fundamental Approaches to Software Engineering (FASE), Lecture Notes in Computer Science, vol. 6603, Springer, Berlin, Heidelberg, 2011, pp. 2–17.

[24] T. Han, J.P. Katoen, D. Berteun, Counterexample generation in probabilistic model checking, IEEE Trans. Softw. Eng. 35 (2009), 241–257.

[25] A. Hinton, M. Kwiatkowska, G. Norman, D. Parker, PRISM: a tool for automatic verification of probabilistic systems, in: H. Hermanns, J. Palsberg (Eds.), Tools and Algorithms for the Construction and Analysis of Systems (TACAS), Lecture Notes in Computer Science, vol. 3920, Springer, Berlin, Heidelberg, 2006, pp. 441–444.

[26] D. Eppstein, Finding the k shortest paths, SIAM J. Comput. 28 (1998), 652–673.

[27] H. Debbi, A. Debbi, M. Bourahla, Debugging of probabilistic systems using structural equation modelling, Int. J. Critic. Comput. Based Syst. 6 (2017), 250–274.

[28] H. Aljazzar, S. Leue, K*: a heuristic search algorithm for finding the k shortest paths, Artif. Intell. 175 (2011), 2129–2154.

[29] H. Debbi, M. Bourahla, Generating diagnoses for probabilistic model checking using causality, Comput. Inform. Technol. 21 (2013), 13–22.

[30] H. Hermanns, B. Wachter, L. Zhang, Probabilistic CEGAR, in: A. Gupta, S. Malik, Computer Aided Verification (CAV), Lecture Notes in Computer Science, vol. 5123, Springer, Berlin, Heidelberg, 2008, pp. 162–175.

[31] B. Dutertre, L. de Moura, The Yices SMT Solver, Technical Report, SRI International, 2006.

[32] M.O. Rabin, Probabilistic automata, Inform. Control. 6 (1963), 230–245.

[33] L. Feng, On Learning Assumptions for Compositional Verification of Probabilistic Systems, Ph.D. thesis, University of Oxford, 2013.

[34] B. Bollig, P. Habermehl, C. Kern, M. Leucker, Angluin-style learning of NFA, in: Boutilier, Craig, Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI), AAAI Press, Pasadena, CA, USA, 2009, pp. 1004–1009.

[35] F. Denis, A. Lemay, A. Terlutte, Residual finite state automata, Fund. Inform. 51 (2002), 339–368.

[36] F. Denis, A. Lemay, A. Terlutte, Learning regular languages using RFSAs, Theor. Comput. Sci. 313 (2004), 267–294.

[37] L. de Alfaro, Formal Verification of Probabilistic Systems, Stanford University, 1997.

[38] M.O. Rabin, D.S. Scott, Finite automata and their decision problems, IBM Journal of Research and Development, IBM J. Res. Dev. 3 (1959), 114–125.

[39] F. He, X. Gao, M. Wang, B-Y. Wang, L. Zhang, Learning weighted assumptions for compositional verification of Markov decision processes, ACM Trans. Softw. Eng. Meth. 25 (2016), 1–39.

[40] Y. Ma, Z. Cao, Y. Liu, Counterexample generation in stochastic model checking based on pso algorithm with heuristic, Int. J. Softw. Eng. Knowl. Eng. 26 (2016), 1117–1143.

[41] B. Bollig, J.P. Katoen, C. Kern, M. Leucker, D. Neider, D.R. Piegdon, libalf: The automata learning framework, in: T. Touili, B. Cook, P. Jackson (Eds.), Computer Aided Verification (CAV), Lecture Notes in Computer Science, vol. 6174, Springer, Berlin, Heidelberg, 2010, pp. 360–364.

[42] C. Păsăreanu, D. Giannakopoulou, M. Bobaru, J.M. Cobleigh, H. Barringer, Learning to divide and conquer: applying the L* algorithm to automate assume-guarantee reasoning, Formal Methods Syst Des. 32 (2008), 175–205.