Innovations in Systems and Software Engineering (2018) 14:229–244
https://doi.org/10.1007/s11334-018-0316-7

ORIGINAL PAPER

Learning-based symbolic assume-guarantee reasoning for Markov decision process by using interval Markov process

Redouane Bouchekir[1] · Mohand Cherif Boukala[1]

Received: 24 March 2017 / Accepted: 24 May 2018 / Published online: 1 June 2018

Abstract

Many real-life critical systems are described with large models and exhibit both probabilistic and non-deterministic behaviour. Verification of such systems requires techniques to avoid the state space explosion problem. *Symbolic model checking* and compositional verification such as *assume-guarantee reasoning* are two promising techniques to overcome this barrier. In this paper, we propose a probabilistic symbolic compositional verification approach (PSCV) to verify probabilistic systems where each component is a Markov decision process (MDP). PSCV starts by encoding implicitly the system components using compact data structures. To establish the symbolic compositional verification process, we propose a sound and complete symbolic assume-guarantee reasoning rule. To attain completeness of the symbolic assume-guarantee reasoning rule, we propose to model assumptions using interval MDP. In addition, we give a symbolic MTBDD-learning algorithm to generate automatically the symbolic assumptions. Moreover, we propose to use causality to generate small counterexamples in order to refine the conjecture assumptions. Experimental results suggest promising outlooks for our probabilistic symbolic compositional approach.

Keywords Probabilistic model checking · Compositional verification · Symbolic model checking · Assume-guarantee paradigm · Model learning

1 Introduction

1.1 Context and motivation

With the increase in complexity of modern computing systems, the chances of introducing errors have increased significantly. Detecting and fixing every error is a very important task, especially for safety-critical applications. The main approach to verify such systems is *model checking* [14,15], which consists to verify systems by exhaustively searching the complete state space.

Many safety-critical systems exhibit non-deterministic stochastic behaviour [44], for example, probabilistic protocol [20], randomized distributed algorithms [28], fault tolerant systems [43], composition of inter-organizational workflow [7]. *Probabilistic verification* is a set of techniques for formal modelling and analysis of such systems. *Probabilistic model checking* [1,4,5] involves the construction of a finite-state model augmented with probabilistic information, such as Markov chains or probabilistic automaton [24,34]. This is then checked against properties specified in probabilistic extensions of temporal logic, such as probabilistic computation tree logic (PCTL) [23]. As with any formal verification technique, one of the main challenges for probabilistic model checking is the *state space explosion problem*: the number of system states grows exponentially in the number of concurrent components. Indeed, even for small models, the size of the state space may be massive; and this can cause the failure of the verification process.

To cope with the state space explosion problem, several approaches have been proposed such as (1) *compositional verification* [21,22,25,36] and (2) *symbolic model checking* [9,26,38,39]. Compositional verification suggests a "divide and conquer" strategy to reduce the verification task into simpler subtasks. A popular approach is the *assume-guarantee* paradigm [11,13,41], in which individual system components are verified under *assumptions* about their environment. Once

✉ Redouane Bouchekir
rbouchekir@usthb.dz

Mohand Cherif Boukala
mboukala@usthb.dz

1 MOVEP, Computer Science Department, USTHB, Algiers, Algeria

it has been verified that the other system components do indeed satisfy these assumptions, proof rules can be used to combine individual verification results, establishing correctness properties of the overall system. The success of assume-guarantee reasoning approach depends on discovering appropriate assumptions. The process of generating *automatically* useful assumptions can be solved by using an active model learning [13], such as L^* learning algorithm [3].

Symbolic model checking is also a useful technique to cope with the state explosion problem. In symbolic model checking, system states are implicitly represented by predicates, as well as the initial states and transition relation of the system. Using advanced data structures such as binary decision diagrams (BDD) or multi-terminal BDD (MTBDD) [19], a large number of states could efficiently be stored and explored simultaneously.

The enormous success of assume-guarantee reasoning and symbolic model checking techniques to cope with the state space explosion problem motivated us to propose a new approach based on the combination of assume-guarantee reasoning and symbolic model checking techniques. For that, we propose a new approach named *probabilistic symbolic compositional verification* (PSCV) to take advantages of both approaches. The PSCV is based on a symbolic assume-guarantee reasoning rule. The main characteristics of our symbolic assume-guarantee reasoning rule are soundness and completeness. The completeness is guaranteed by the use of interval MDP to represent assumption. In addition, to reduce the size of the state space, PSCV encodes both system components and assumptions implicitly using compact data structures, such as BDD and MTBDD. We also adapt the L^* learning algorithm to generate series of conjecture symbolic assumptions.

1.2 Contributions

Probabilistic symbolic compositional verification aims to avoid the state space explosion problem for probabilistic systems composed of MDP components. Different from the monolithic verification, in PSCV each system component is verified in isolation under an assumption about its contextual environment. To illustrate the PSCV, we consider a system S (Fig. 1) composed of two MDP components M_0 and M_1. In order to guarantee that S satisfies a probabilistic safety property, we first encode the components (M_0 and M_1) using symbolic data structures (symbolic MDP); many works such as [9,19,38,39] show that the symbolic representation is often more efficient than the explicit representation. The second step aims to generate a symbolic assumption S_{I_i}, where M_0 *is embedded in* S_{I_i}. The notion of embedded, denoted by $M_0 \preceq I$, means that the symbolic assumption S_{I_i} should be expressive enough to capture the abstract behaviour of M_0 and amenable to automatic generation via algorithmic learning. If the size of the symbolic assumption S_{I_i} is much smaller than the size of the corresponding component M_0, then we can expect significant gains of the verification performance. In addition, we propose a sound and complete symbolic assume-guarantee reasoning proof rule to define and establish the verification process of $S_{I_i} \parallel M_1$. Moreover, we propose a symbolic MTBDD-learning algorithm to generate automatically the symbolic assumptions S_{I_i}. We developed a prototype tool PSCV4MDP (probabilistic symbolic compositional verification for MDP), which implements the symbolic MTBDD-learning algorithm. We evaluate our approach on a several case studies derived from the PRISM benchmarks, and we have compared the results of our approach with the symbolic monolithic probabilistic verification [33]. Experimental results suggest promising outlooks.

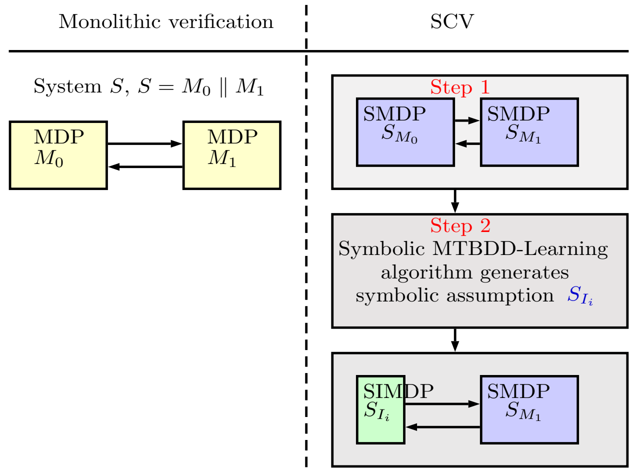

Fig. 1 An overview of our approach, step 1 aims to encode MDP using symbolic MDP and step 2 aims to learn a symbolic assumption S_{I_i}, then the verification process will be established using a symbolic assume-guarantee reasoning proof rule

The remainder of this paper is organized as follows: In Sect. 2 we provide some background knowledge about MDP, interval MDP, the parallel composition MDP $\parallel$ IMDP and the symbolic data structures used to encode MDP and interval MDP. In Sect. 3, we present the PSCV approach, where we detail our symbolic assume-guarantee reasoning proof rule, the encoding process of MDP and interval MDP and the symbolic MTBDD-learning algorithm. Section 4 reports the experimental results of several case studies. Section 5 describes the most relevant works to ours, and Sect. 6 concludes the paper and talks about future works.

2 Preliminaries

In this section, we give some background knowledge about MDP, interval MDP, the parallel composition and the sym-

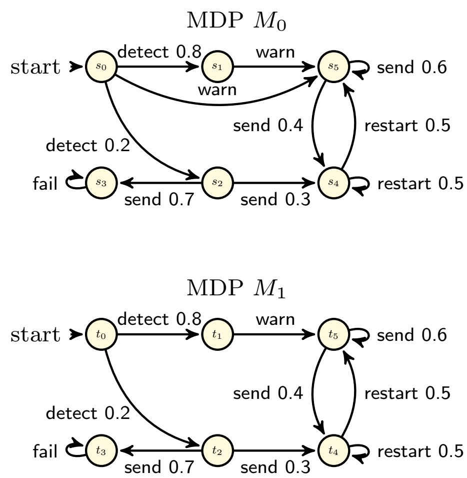

Fig. 2 Example of two MDP, M_0 (above) and M_1 (below)

bolic data structures used to encode implicitly MDP and interval MDP.

MDP are often used to describe and study systems exhibit non-deterministic and stochastic behaviour.

Definition 1 *Markov decision process* (MDP) is a tuple $M = (S_M, s_0^M, \Sigma_M, \delta_M)$ where S_M is a finite set of states, $s_0^M \in S$ is an initial state, Σ_M is a finite set of alphabets and $\delta_M \subseteq S \times (\Sigma_M \cup \{\tau\}) \times Dist(S)$ is a probabilistic transition relation.

In a state s of MDP M, one or more transitions, denoted $(s, a) \rightarrow \mu$, are available, where $a \in \Sigma_M$ is an action label, μ is a probability distribution over states and $(s, a, \mu) \in \delta_M$. A path through MDP is a (finite or infinite) sequence $(s_0, a_0, \mu_0) \rightarrow (s_1, a_1, \mu_1) \rightarrow \cdots$ we denote by *FPath* a finite path through MDP M. To reason about MDP, we use the notion of adversaries, which resolve the non-deterministic choices in MDP, based on its execution history. We denote by σ_M an adversary of MDP M. Formally, an adversary σ_M maps any finite path *FPath* to a distribution over the available transitions in the last state on the part, i.e. mapping every *FPath* to an element $\sigma_M(FPath)$ of δ_M. Indeed, we distinguish several classes of adversaries: (1) *memoryless adversary*, where it always pick the same choice in a state, i.e. the choice depends only on the current state, (2) *finite-memory adversary*, it stores information about the history in a finite-state automaton, (3) *deterministic adversary*, an adversary is deterministic if it always selects a single transition, or (3) *randomized adversary*, where it maps finite paths *FPath* in MDP to a probability distribution over element of δ_M. In our case, the class of deterministic adversaries are sufficient for our problem.

An example of two MDP M_0 and M_1 is shown in Fig. 2.

Interval Markov chains (IMDP) generalize ordinary MDP by using interval-valued transition probabilities rather than just probability value. In this paper, we use interval MDP

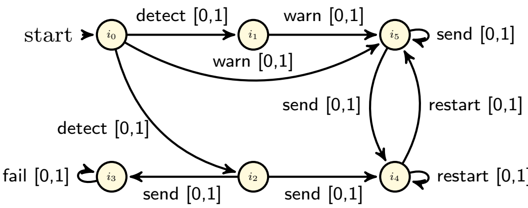

Fig. 3 Example of interval MDP I

to represent the assumptions used on the PSCV verification process.

Definition 2 *Interval Markov Chain* (IMDP) is a tuple $I = (S_I, s_0^I, \Sigma_I, P^l, P^u)$ where S_I, s_0^I and Σ_I are defined as for MDP. $P^l, P^u : S \times \Sigma_I \times S \mapsto [0, 1]$ are matrices representing the lower/upper bounds of transition probabilities such that: $P^l(s, a)(s') \leq P^u(s, a)(s')$ for all states $s, s' \in S$ and $a \in \Sigma_I$.

An example of interval MDP I is shown in Fig. 3.

In Definition 3, we describe how MDP and interval MDP are composed together. This is done by using the asynchronous parallel operator ($\|$) defined by [42], where MDP and interval MDP synchronize over shared actions and interleave otherwise.

Definition 3 *Parallel composition* MDP $\|$ IMDP
Let M and I be MDP and interval MDP, respectively. Their parallel composition, denoted by $M \| I$, is an interval MDP MI, where $MI = M \| I$.
$MI = \{S_M \times S_I, (s_0^M, s_0^I), \Sigma_M \cup \Sigma_I, P^l, P^u\}$, where P^l, P^u are defined such that:
$(s_i, s_j) \xrightarrow{a} [P^l(s_i, a)(s_j) \times \mu_i, P^u(s_i, a)(s_j) \times \mu_i]$ if and only if one of the following conditions holds: let $s_i, s_i' \in S_M$ and $s_j, s_j' \in S_I$.

- $s_i \xrightarrow{a, \mu_i} s_i', s_j \xrightarrow{P^l(s_j,a)(s_j'), P^u(s_j,a)(s_j')} s_j'$, where $a \in \Sigma_M \cap \Sigma_I$,
- $s_i \xrightarrow{a, \mu_i} s_i'$, where $a \in \Sigma_M \backslash \Sigma_I$,
- $s_j \xrightarrow{P^l(s_j,a)(s_j'), P^u(s_j,a)(s_j')} s_j'$, where $a \in \Sigma_M \backslash \Sigma_I$.

Example 1 To illustrate the parallel composition, we consider the example of MDP M_0 and interval MDP I shown in Figs. 2 and 3, respectively. The product $MI = M_0 \| I$ obtained from their parallel composition is shown in Fig. 4. M_0 and I synchronize over their shared actions $\Sigma_{M_0} \cap \Sigma_I = \{detect, warn, send, restart, fail\}$.

MDP and interval MDP can be implicitly encoded using compact data structures such as BDD and MTBDD. Let B denote the *Boolean domain* $\{0, 1\}$. Fix a finite ordered set of Boolean variables $X = \langle x_1, x_2, \ldots, x_n \rangle$. A *valuation* $v =$

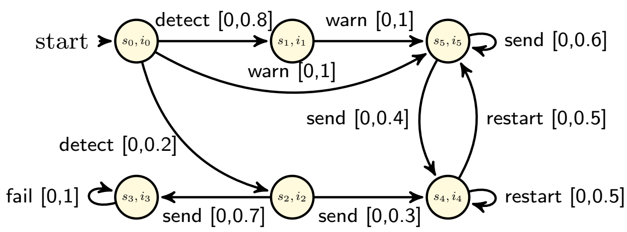

Fig. 4 Interval MDP MI result of the parallel composition $M_0 \parallel I$

$\langle v_1, v_2, \ldots, v_n \rangle$ of X assigns the Boolean value v_i to the Boolean variable x_i.

Definition 4 A *binary decision diagram* (BDD) is a rooted, directed acyclic graph with its vertex set partitioned into non-terminal and terminal vertices (also called nodes). A non-terminal node d is labelled by a variable $var(d) \in X$. Each non-terminal node has exactly two children nodes, denoted $then(d)$ and $else(d)$. A terminal node d is labelled by a Boolean value $val(d)$ and has no children. The Boolean variable ordering $<$ is imposed onto the graph by requiring that a child d' of a non-terminal node d is either terminal, or is non-terminal and satisfies $var(d) < var(d')$.

Definition 5 A *multi-terminal binary decision diagram* (MTBDD) is a BDD where the terminal nodes are labelled by a real number.

In this work, we consider the verification of probabilistic safety properties specified using PCTL in the form of $P_{\leq\rho}[\psi]$ with $\rho \in [0, 1]$ and

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

where a is an atomic proposition, ϕ is a state formula, ψ is a path formula and U is the *Until* temporal operator.

We use also the operator $\Diamond$ (diamond) operator to specify probabilistic safety property. Intuitively, a property of the form $\Diamond\phi$ means that ϕ is eventually satisfied. This operator can be expressed in terms of the PCTL until as follows:

$$\Diamond\phi \equiv true U \phi.$$

3 Probabilistic symbolic compositional verification

In this paper, we propose a probabilistic symbolic compositional verification approach (PSCV) to verify whether a system S composed of MDP components satisfies or not a probabilistic safety property $P_{\leq\rho}[\psi]$. The PSCV process is based on our proposed symbolic assume-guarantee reasoning proof rule, where assumptions are represented using interval MDP.

Definition 6 Let $M = (S_M, s_0^M, \Sigma_M, \delta_M)$ and $I = (S_I, s_0^I, \Sigma_I, P^l, P^u)$ be MDP and interval MDP, respectively. We say M is embedded in I, written $M \preceq I$, if and only if: (1) $S_M = S_I$, (2) $s_0^M = s_0^I$, (3) $\Sigma_M = \Sigma_I$, and (4) $P^l(s,a)(s') \leq \mu(s,a)(s') \leq P^u(s,a)(s')$ for every $s, s' \in S_M$ and $a \in \Sigma_M$.

Example 2 Consider the MDP M_0 shown in Fig. 2 and interval MDP I shown in Fig. 3. They have the same state space, identical initial state (s_0, i_0) and the same set of actions $\{detect, warn, send, restart, fail\}$. In addition, the transition probability between any two states in M_0 lies within the corresponding transition probability interval in I by taking the same action. For example, the transition probability between s_0 and s_1 is $s_0 \xrightarrow{detect,0.8} s_1$, which falls into the interval $[0, 1]$ labelled the transition $i_0 \xrightarrow{detect,[0,1]} i_1$ in I, formally $P^l(i_0, detect)(i_1) \leq \mu(s_0, detect)(s_1) \leq P^u(i_0, detect)(i_0')$. Thus, we have $M_0 \preceq I$.

Theorem 1 *Let M_0, M_1 be MDP and $P_{\leq\rho}[\psi]$ a probabilistic safety property, then the following proof rule is sound and complete:*

$$\frac{\begin{array}{lr} M_0 \preceq I & (1) \\ I \parallel M_1 \models P_{\leq\rho}[\psi] & (2) \end{array}}{\begin{array}{lr} M_0 \parallel M_1 \models P_{\leq\rho}[\psi] & (3) \end{array}}$$

This proof rule means if we have a system S composed of two components M_0 and M_1, where $S = M_0 \parallel M_1$, then we can check the correctness of a probabilistic safety property $P_{\leq\rho}[\psi]$ over S without constructing and verifying the full state space. Instead, we first generate an appropriate assumption I, where I is an interval MDP, then we check if this assumption could be used to verify S by checking the two promises:

(1) *Check if M_0 is embedded in I, $M_0 \preceq I$,*
(2) *Check if $I \parallel M_1$ satisfies the probabilistic safety property $P_{\leq\rho}[\psi]$, $I \parallel M_1 \models P_{\leq\rho}[\psi]$.*

If the two promises are satisfied, then we can conclude that $M_0 \parallel M_1$ satisfies $P_{\leq\rho}[\psi]$.

Proof Let M_0 and M_1 be MDP, where $M_0 = (S_{M_0}, s_0^{M_0}, \Sigma_{M_0}, \delta_{M_0})$, $M_1 = (S_{M_1}, s_0^{M_1}, \Sigma_{M_1}, \delta_{M_1})$, and interval MDP I, $I = (S_I, s_0^I, \Sigma_I, P^l, P^u)$. If $M_0 \preceq I$ and based on Definition 6 we have $S_M = S_I, s_0^M = s_0^I, \Sigma_M = \Sigma_I$, and $P^l(s,a)(s') \leq \mu(s,a)(s') \leq P^u(s,a)(s')$ for every $s, s' \in S_{M_0}$ and $a \in \Sigma_{M_0}$. Based on Definitions 3 and 6, $M_0 \parallel M_1$ and $I \parallel M_1$ have the same state space, initial state and actions. Since $P^l(s,a)(s') \leq \mu(s,a)(s') \leq P^u(s,a)(s')$, and we suppose the transition probability of $M_0 \parallel M_1$ as: $\mu_{M_0 \parallel M_1}((s_i, s_j), a)(s_i', s_j') =$

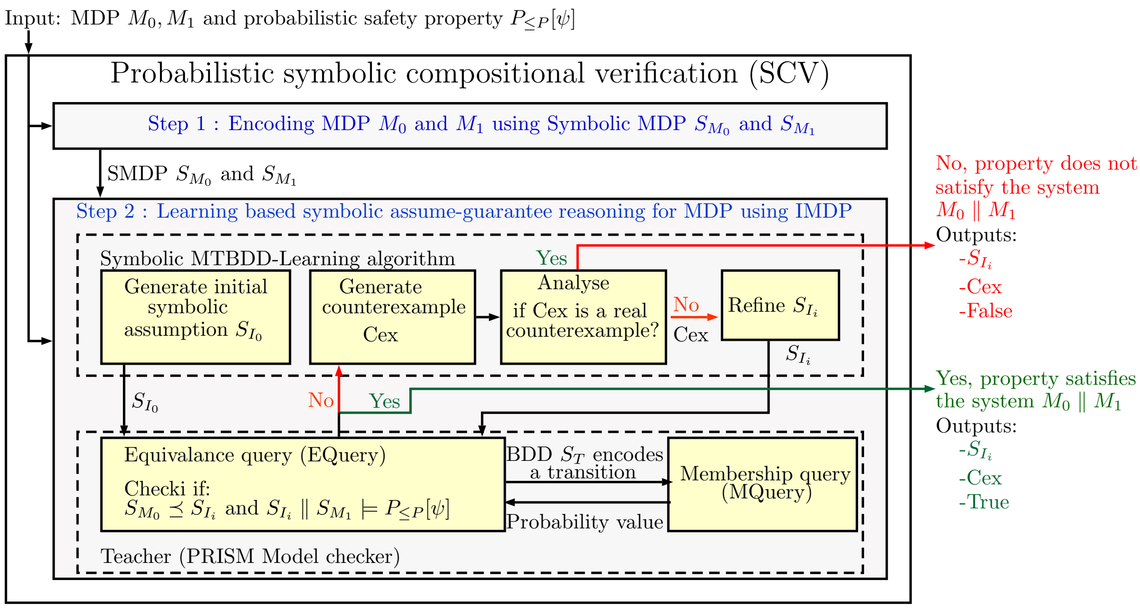

Fig. 5 An overview of our approach probabilistic symbolic compositional verification approach (PSCV)

$\mu_{M_0}((s_i), a)(s'_i) \times \mu_{M_1}((s_j), a)(s'_j)$ for any state $s_i, s'_i \in S_{M_0}$ and $s_j, s'_j \in S_{M_1}$. Thus, $P^l((s_i, s_j), a)(s'_i, s'_j) \leq \mu_{M_0 \| M_1}((s_i, s_j), a)(s'_i, s'_j) \leq P^u((s_i, s_j), a)(s'_i, s'_j)$ for the probability between two states (s_i, s'_i) and (s_j, s'_j). In $I \parallel M_1$ the probability interval between any two states (s_i, s_j) and (s'_i, s'_j) is restricted by the interval $[P^l((s_i, a)(s'_i) \times \mu_{M_1}(s_j), a)(s'_j), P^u((s_i, a)(s'_i) \times \mu_{M_1}(s_j), a)(s'_j)]$, this implies, if $M_0 \preceq I$ and $I \parallel M_1 \models P_{\leq\rho}[\psi]$ then $M_0 \parallel M_1 \models P_{\leq\rho}[\psi]$ is guaranteed. □

The main characteristic of our PSCV approach is completeness:

– If $M_0 \parallel M_1 \models P_{\leq\rho}[\psi]$, then PSCV returns *true* and an assumption I, where $I \parallel M_1 \models P_{\leq\rho}[\psi]$,
– If $M_0 \parallel M_1 \not\models P_{\leq\rho}[\psi]$, then PSCV returns *false*, assumption I and a counterexample showing the reason why $P_{\leq\rho}[\psi]$ is violated.

Proof In our approach PSCV, the symbolic learning algorithm targets the component M_0, and it will infer an assumption I eventually. If $M_0 \parallel M_1 \models P_{\leq\rho}[\psi]$, the MTBDD-learning algorithm always infers I, where $M_0 \preceq I$. In the worst case, the upper probability value of the final assumption I will be equal to transition probability value of M_0. Otherwise, if $M_0 \parallel M_1 \not\models P_{\leq\rho}[\psi]$, the MTBDD-learning algorithm will start by generating an initial assumption I_0, if I_0 is too strong (the upper probability value is too big), the MTBDD-learning algorithm will refine I based on a counterexample. The refinement process will lead to generate a real counterexample showing the reason why $P_{\leq\rho}[\psi]$ is violated. In the worst case, the refinement process will set the upper probability value of the final assumption I as for transition probability value in M_0. □

Figure 5 presents an overview of the PSCV approach. PSCV is based on the symbolic assume-guarantee reasoning proof rule. In our approach, assumptions are represented using interval MDP instead of DFA, and the state space is encoded using compact data structures. In addition, the L^* learning algorithm is adapted to accept the implicit representation of the state space. Furthermore, our approach always terminate. The choice of using interval MDP to represent assumptions is motivated by: (1) MDP can be embedded in an interval MDP (Definition 6), this how make our assume-guarantee reasoning rule sound, where interval MDP captures the abstract behaviour of the original MDP; (2) it is amenable to generate assumption represented by interval MDP using the L^* learning algorithm; (3) the implicit representation of interval MDP can be more efficient in size than the corresponding MDP; (4) completeness, it is always possible to find an interval MDP, where MDP is embedded on it; and (5) by using interval MDP, we can easily extend the PSCV to verify properties of the form $P_{\geq\rho}[\psi]$.

The first step of PSCV aims to encode components (M_0 and M_1) by means of SMDP. Instead of representing the state space of MDP explicitly (using explicit representation), we encode the state space, transitions and actions using symbolic MDP (SMDP).

In Definition 7, we introduce SMDP and we provide the different data structures used to encode implicitly MDP.

Definition 7 Symbolic MDP (SMDP) is a tuple $S_M = (X, Init_M, Y, f_{S_M}(yxx'z), Z)$ where X, Y and Z are finite ordered set of Boolean variables with $X \cap Y \cap Z = \emptyset$. $Init(X)$

is an initial predicate over X and $f_{S_M}(yxx'z)$ is a transition predicate over $Y \cup X \cup X' \cup Z$ where y, x, x', z are valuations of receptively, Y, X, X' and Z. The set X encodes the states of S, X' next states, Y encodes alphabets, and Z encodes the non-deterministic choice.

More concretely, let $M = (S_M, s_0, \Sigma_M, \delta_M)$ be MDP, $n = |S_M|$, $m = |\Sigma_M|$ and $k = \lceil log_2(n) \rceil$. We can see δ_M as a function of the form $S_M \times \Sigma_M \times \{1, 2, \ldots, r\} \times S_M \rightarrow [0, 1]$, where r is the number of non-deterministic choice of a transition. We use a function $enc : S_M \rightarrow \{0, 1\}^k$ over $X = \langle x_1, x_2, \ldots, x_k \rangle$ to encode states in S_M and $X' = \langle x'_1, x'_2, \ldots, x'_k \rangle$ to encode next states. We use also $Y = \langle y_1, y_2, \ldots, y_m \rangle$ to encode actions and we represent the non-deterministic choice using $Z = \langle z_1, z_2, \ldots, z_r \rangle$. Let x, x', y, z be valuations of X, X', Y and Z, respectively. A valuation x of X or X' encodes a state s by $enc(s)$, $s \in S_M$. A valid (true) valuation y encodes an action a by $enc(a)$, $a \in \Sigma_M$.

Example 3 We consider the MDP M_0 (Fig. 2). M_0 contains the set of states $S_{M_0} = \{s_0, s_1, s_2, s_3, s_4, s_5\}$ and the set of actions $\Sigma_{M_0} = \{detect, warn, send, restart, fail\}$. We use $X = \langle x_0, x_1, x_2 \rangle$ to encode the set of states in S_{M_0}: $enc(s_0) = (000), enc(s_1) = (001), enc(s_2) = (010), enc(s_3) = (011), enc(s_4) = (100), enc(s_5) = (101)$; and we use the set $Y = \langle d, w, s, r, f \rangle$ to encode the actions $\{detect, warn, send, restart, fail\}$, respectively. Table 1 summarizes the process of encoding the transition function δ_{M_0}, and its corresponding MTBDD is shown in Fig. 6. In this figure, the terminal node labelled by the value 0 and its incoming edges are removed, solid (doted) edges are labelled by the value 1 (0), respectively.

Table 1 Encoding the set of states and the transition function of MDP M_0 (Fig. 2)

$s \in S_{M_0}$	s_0	s_1	s_2	s_3	s_4	s_5	$enc(s_i)$
s_0	0	0.8	0.2	0	0	1	000
s_1	0	0	0	0	0	1	001
s_2	0	0	0	0.7	0.3	0	010
s_3	0	0	0	1	0	0	011
s_4	0	0	0	0	0.5	0.5	100
s_5	0	0	0	0	0.4	0.6	101

δ_{M_0}	y	x_0	x'_0	x_1	x'_1	x_2	x'_2	z
$s_0 \xrightarrow{detect} s_1$	d	0	0	0	0	0	1	0
$s_0 \xrightarrow{detect} s_2$	d	0	0	0	1	0	0	0
$s_0 \xrightarrow{warn} s_5$	w	0	1	0	0	0	1	1
$s_1 \xrightarrow{warn} s_5$	w	0	1	0	0	1	1	0
$s_2 \xrightarrow{send} s_3$	s	0	0	1	1	0	1	0
$s_2 \xrightarrow{send} s_4$	s	0	1	1	0	0	0	0
$s_3 \xrightarrow{fail} s_3$	f	0	0	1	1	1	1	0
$s_4 \xrightarrow{restart} s_4$	r	1	1	0	0	0	0	0
$s_4 \xrightarrow{restart} s_5$	r	1	1	0	0	0	1	0
$s_5 \xrightarrow{send} s_4$	s	1	1	0	0	0	1	0
$s_5 \xrightarrow{send} s_5$	s	1	1	0	0	1	1	0

Following the same process to encode MDP implicitly as SMDP, we can encode interval MDP as SIMDP.

Definition 8 Symbolic interval MDP (SIMDP) is a tuple $S_I = (X, Init_I, Y, f^l_{S_I}(yxx'z), f^u_{S_I}(yxx'z), Z)$ where X, Y and Z, $Init_I$ are defined as for interval MDP. $f^l_{S_I}(yxx'z)$ and $f^u_{S_I}(yxx'z)$ are a transition predicates over $Y \cup X \cup X' \cup Z$ where y, x, x', z are valuations of receptively, Y, X, X' and Z. In practice, $f^l_{S_I}(yxx'z)$ and $f^u_{S_I}(yxx'z)$ are MTBDD encoding the interval MDP, respectively, with lower and upper probability values.

As described in proof of Theorem 1, $M_0 \parallel M_1$ and $I \parallel M_1$ have the same state space, initial state and actions. The implicit representation is introduced to reduce the size of the state space. Indeed, the implicit representation of $M_0 \parallel M_1$ and $I \parallel M_1$ may be different. This is due to the probability values. Assumption I uses interval-valued transition probabilities rather than probability value in the original component M_0, and if the upper/lower bound is better uniformed than the probabilities in M_0, then we can expect a gain in the size of the state space. This is achieved because the final nodes in MTBDD encoding I are less than MTBDD encoding M_0, for that many non-terminal nodes will be merged together. In practice, and to improve the performance of our approach, in term of the size of the state space, our approach generates the first assumption with transitions, where the interval probability value is equal to [0, 1], between all states of the model, this will lead to merge many non-terminal nodes.

The second step in our approach, PSCV, aims to generate series of conjecture symbolic assumption S_{I_i}. Since we use symbolic data structures to encode MDP, in the aim of reducing the size of the state space, the symbolic assume-guarantee reasoning rule could be rephrased as follows.

Let S_{M_0}, S_{M_1} be SMDP and $P_{\leq \rho}[\psi]$ a probabilistic safety property; the following proof rule is sound and complete:

$$\begin{array}{lr} S_{M_0} \preceq S_{I_i} & (1) \\ S_{I_i} \parallel S_{M_1} \models P_{\leq \rho}[\psi] & (2) \\ \hline S_{M_0} \parallel S_{M_1} \models P_{\leq \rho}[\psi] & (3) \end{array}$$

The proof of the rephrased symbolic assume-guarantee reasoning rule follows the same proof of the initial assume-reasoning rule.

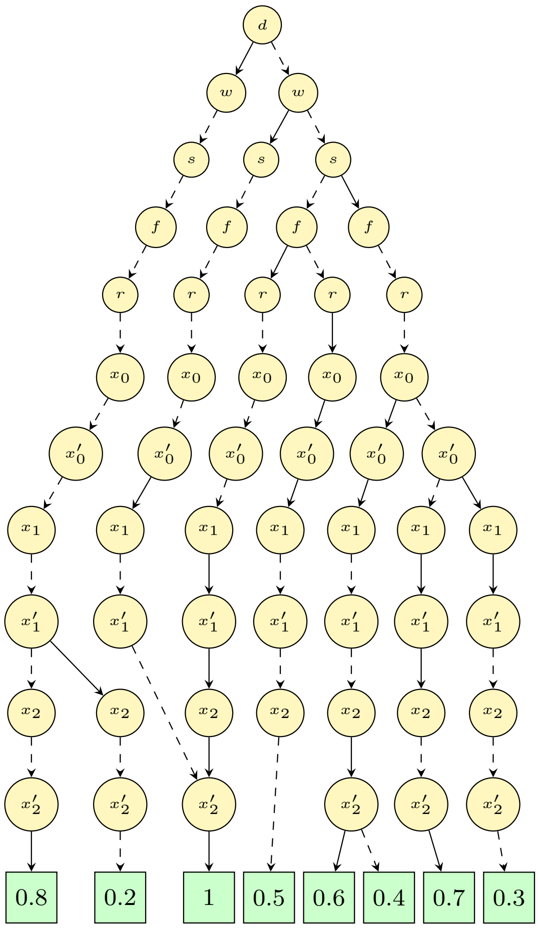

Fig. 6 MTBDD encoding the transition function δ_{M_0} (M_0 in Fig. 2)

3.1 Symbolic MTBDD-learning framework

According to the symbolic assume-guarantee rule for MDP, given an appropriate symbolic assumption S_{I_i}, we can check the correctness of a probabilistic safety property $P_{\leq\rho}[\psi]$ on $S_{M_0} \parallel S_{M_1}$, without constructing and analysing the full model. This rule is sound and complete. However, the main challenge consists of the automatic generation of an appropriate symbolic assumption. The aim of the second step, which is the symbolic MTBDD-learning framework is to learn such assumption. Our framework takes the symbolic representation of M_0 and M_1 result of the first step as input and uses the symbolic MTBDD-learning algorithm to generate a series of conjecture symbolic assumptions S_{I_i}. In Sect. 3.1.1, we detail the process of generating S_{I_i}.

3.1.1 The symbolic MTBDD-learning algorithm

The L^* learning algorithm [3] is a formal method to learn a deterministic finite automata (DFA) with the minimal number of states that accepts an unknown language L over an alphabet Σ (target language). During the learning process, the L^* algorithm interacts with a *Teacher* to make two types of queries: (i) membership queries and (ii) equivalence queries. A *membership queries* are used to check whether some words w are in the target language or not. *Equivalence queries* are used to check whether a conjectured DFA A accepts the target language. If the conjectured DFA is not correct, the teacher would return a counterexample to L^* to refine the automaton being learnt.

The L^* learning algorithm has been widely used in compositional verification of non-probabilistic systems. However, for probabilistic systems such as MDP, it was demonstrated that it is undecidable to infer MDP under a version of L^* learning algorithm, and a learning algorithm for general probabilistic systems may not exist after all [31]. In our work, we propose to use interval MDP to model assumptions instead of MDP or DFA. Moreover, we adapted the L^* learning algorithm to accept the symbolic representation of MDP (SMDP) and learns a symbolic assumption S_{I_i} encoded using SIMDP. Our proposed symbolic MTBDD-learning algorithm is shown in Algorithm 1.

Algorithm 1 Symbolic MTBDD-learning algorithm

Input: $MDP\, M_0$, $SMDP\, S_{M_0}$, S_{M_1} and $\varphi = P_{\leq\rho}[\psi]$
output: $SIMDP\, S_{I_i}$, set of counterexamples and a Boolean value
Begin
$SIMDP\, S_{I_0} = Generate\_S_{I_0}$ (M_0,S_{M_0});
Boolean $b = EQuery(S_{I_0}, S_{M_0}, S_{M_1}, \varphi)$;
if b == true **then**
  return (S_{I_0}, null,true);
else
  $i = 1$;
  $S_{I_i} = S_{I_0}$;
  repeat
    Cex = $GenerateCex(S_{I_0}, S_{M_1}, \varphi)$;
    isRealCex = $AnalyseCex$(Cex, S_{M_0}, S_{M_1}, φ);
    if isRealCex == true **then**
      return (S_{I_i}, Cex, false);
    else
      $i++$;
      $S_{I_i} = Refine\_S_{I_i}(S_{I_i}$, Cex);
      $b = EQuery(S_{I_i}, S_{M_0}, S_{M_1}, \varphi)$;
    end if
  until b == true;
  return (S_{I_i}, null, true);
end if
End

The symbolic MTBDD-learning algorithm starts by generating an initial assumption S_{I_0}. Since we use symbolic data structures to represent the system components as well as assumptions, the function $Generate\_S_{I_0}$ accepts MDP M_0 and SMDP S_M as inputs, and returns SIMDP S_{I_0}. The process of generating S_{I_0} is described in Algorithm 2. According to the symbolic assume-guarantee reasoning proof rule, S_{I_0} has the same state space as M_0, i.e. the same initial state, set of states and actions. Thus, the function $Generate\_S_{I_0}$ initializes the same data structures to X^I, $Init_I$ and Y^I as for S_{M_0}.

Algorithm 2 $Generate\_S_{I_0}$

1: **Input:** $MDPM_0$, $SMDPS_{M_0}$
2: **output:** $SIMDPS_{I_0}$
3: **BEGIN**
4: We consider $S_{M_0} = \{X^M, Init_M, Y^M, f_{S_M}(yxx'z), Z^M\}$ and let $S_{I_0} = \{X^I, Init_I, Y^I, f^l_{S_I}(yxx'z), f^u_{S_I}(yxx'z), Z^I\}$ be SIMDP.
5: Create a new Interval MDP I_0 equivalent to M_0, with transition equal to [0, 1] between all states. The set of actions in M_0 are hold in each transition of I_0.
6: $X^I = X^M$;
7: $Init_I = Init_M$;
8: $Y^I = Y^M$;
9: $f^l_{S_I}(yxx'z) =$ MTBDD with one final node labelled 0;
10: Convert the transition function of I_0 to MTBDD $f^u_{S_I}$;
11: return S_{I_0};
12: **End**

Table 2 Comparison between the size of MTBDD encoding the transition function of M_0 and I_0 for randomized dining philosophers

No.	S_{M_0}		S_{I_0}	
	T4MC	M. Nodes	T4MC	M. Nodes
6	0.09	5008	0.25	2832
16	7091	37,523	0.301	20,127
18	11,042	43,472	0.372	25,470
20	16,563	58,565	0.75	31,441
30	64,207	131,260	1.385	70,716
40	169,215	232,655	3.016	125,691

Since the initial assumption have the same state space as S_{M_0}, and to optimize the implicit representation of the transition function of S_{I_0}, $Generate\_S_{I_0}$ creates a new interval MDP I_0 equivalent to M_0, with transition equal to [0, 1] between all states, where the set of actions are hold in each transition, then it converts the transition function of I_0 to MTBDD $f^u_{S_I}$. The aim behind the generation of S_{I_0} with transition equal to [0, 1] between all states is to reduce the size of the implicit representation of the state space. Indeed, for large probabilistic system, when we use uniform probabilities, 0 and 1 in our case, this will reduce the number of terminal nodes as well as non-terminal nodes. Adding transition between all states, will keep our assume-guarantee verified for the initial assumption, since M_0 is embedded in I_0, in addition, this process will help to reduce the size of the implicit representation of I_0 and this by combining any isomorphic sub-tree into a single tree, and eliminating any nodes whose left and right children are isomorphic. To illustrate the size gain of using this process, we consider the example of *randomized dining philosophers* [18,35]. (This example is described in Sect. 4.) The results are reported in Table 2. In this table we compare the size of MTBDD encoding the transition function of M_0 and I_0 for *randomized dining philosophers* model, where $T4MC$ describes the time for model construction and *M. Nodes* illustrates the number of model nodes.

Example 4 To illustrate the PSCV approach, we propose the verification of the system $S = M_0 \parallel M_1$ (Fig. 2) against the probabilistic safety property: $\varphi = P_{\leq 0.1}[\Diamond "fail"]$, where $"fail"$ stands for the state $\langle s_3 t_3 \rangle$. The property φ means that the maximum probability that the system S should never fails, over all possible adversaries, is less than 0.1. Initially, PSCV starts by encoding the system components M_0 and M_1 using SMDP. Then, it calls the symbolic MTBDD-learning algorithm to generate an appropriate symbolic assumption S_{I_i}. The symbolic MTBDD-learning algorithm generates an initial conjecture symbolic assumption S_{I_0}.

An equivalence query is made in line 5 (Algorithm 1) to check whether the initial conjectured symbolic assumption S_{I_0} could be used in the compositional verification. Like the L^* learning algorithm, our symbolic MTBDD-learning algorithm interacts with a *teacher* to answer two types of queries: *equivalence queries* and *membership queries*. However, in our case, the interpretation of these latter is different.

3.1.2 Equivalence queries

In our approach, equivalence queries are used to check whenever a symbolic conjecture assumption S_{I_i} can be used to establish the two promises of the symbolic assume-guarantee rule. The first step is to check if $S_{M_0} \preceq S_{I_i}$, this is done by checking if: (1) $S_M = S_{I_i}$, (2) $s^M_0 = s^I_0$, (3) $\Sigma_M = \Sigma_I$, and (4) $P^l(s, a)(s') \leq \mu(s, a)(s') \leq P^u(s, a)(s')$ for every $s, s' \in S_M$ and $a \in \Sigma_M$ (see Definition 6). Since we use symbolic data structures to encode MDP and interval MDP, the checking process can be rephrased as follows:
Let $S_M = (X^M, Init_M, Y^M, f_{S_M}(yxx'z), Z^M)$ and $S_{I_i} = (X^I, Init_I, Y^I, f^l_{S_I}(yxx'z), f^u_{S_I}(yxx'z), Z^I)$ be SMDP and SIMDP, respectively. We say that $S_{M_0} \preceq S_{I_i}$ if and only if: (1) $X^M = X^I$, (2) $Init_M = Init_I$, (3) $Y^M = Y^I$, and (4) $MQuery(S_T, f^u_{S_I}) \leq MQuery(S_T, f_{S_M}) \leq MQuery(S_T, f^l_{S_I})$ for every $s \in S_M$, where S_T is a BDD encoding a transition $s_i \xrightarrow{a} s_j$ ($s_i, s_j \in S_M$). $EQuery$ calls the function $Threshold$ if:
$Prob_{S_M} < Prob_{SI^l}$ or $Prob_{S_M} > Prob_{SI^u}$. $Threshold$ changes the final nodes in $f^l_{S_{SI}}$ (or $f^u_{S_{SI}}$) to $Prob_{S_M}$ following the BDD S_T. Otherwise, if the first promise is valid, then the function $EQuery$ would check the second promise of the symbolic assume-guarantee rule, i.e. Does $S_{I_i} \parallel S_{M_2}$ satisfy $P_{\leq \rho}[\psi]$?
The second step is done by applying a symbolic probabilistic model checking. In practice, to check whether $S_{I_i} \parallel S_{M_2} \models P_{\leq \rho}[\psi]$, the model checker needs first to compute the parallel composition $S_{I_i} \parallel S_{M_2}$, which results an SIMDP. The symbolic model checking algorithm SPMC used in the function $EQuery$ (line 12) is described in Sect. 3.1.4. The function $EQuery$ is illustrated in Algorithm 3.

Algorithm 3 $EQuery$

```
Input: SIMDPS_{I_i}, SMDPS_{M_0}, SMDPS_{M_1}, P_{≤ρ}[ψ]
output: Boolean value
Begin
for all transition t in S_M do
    BDD S_T = Convert t to BDD;
    Double Prob_{SI^u} = MQuery(S_T, f^u_{S_I});
    Double Prob_{S_M} = MQuery(S_T, f_{S_M});
    Double Prob_{SI^l} = MQuery(S_T, f^l_{S_I});
    if Prob_{S_M} < Prob_{SI^l} then
        Threshold(f^l_{S_{SI}}, <, S_T, Prob_{S_M});
    end if
    if Prob_{S_M} > Prob_{SI^u} then
        Threshold(f^u_{S_I}, >, S_T, Prob_{S_M});
    end if
end for
Boolean sat = SPMC(SI, f_{S_M}, P_{≤ρ}[ψ]);
return sat;
End
```

3.1.3 Membership queries

Membership queries can be seen as a function $MQuery$ $(BDDS_T, MTBDDf_{S_M})$:$Double$. $MQuery$ tacks a BDD S_T and MTBDD f_{S_M} as inputs, where the BDD S_T encodes a transition $s_i \xrightarrow{a} s_j$ over $Y \cup X \cup X'$, and returns the probability value between $s_i \xrightarrow{a} s_j$ in f_{S_M} by applying the function $Apply(*, S_T, f_{S_M})$, where apply returns an MTBDD representing the function $S_T * f_{S_M}$. Multiplying the BDD S_T with the MTBDD f_{S_M} removes all transitions from f_{S_M} which do not connect states of S_T, i.e. the result MTBDD encodes only $s_i \xrightarrow{a} s_j$ with the original probability value in f_{S_M}.

Algorithm 4 $MQuery$

```
Input: BDDS_T, MTBDDf_{S_M}
output: double
Begin
MTBDDR = Apply(*, S_T, f_{S_M});
d = get the terminal node of R;
return val(d);
End
```

Example 5 It is clear that the initial assumption S_{M_0} is embedded in S_{I_0}. Indeed, the main step in the equivalence query is the verification if $S_{I_i} \parallel S_{M_1}$ satisfies the property φ or not. To model checking the system $S_{I_0} \parallel S_{M_1} \models^? P_{\leq\rho}[\psi]$, the function $EQuery$ calls the symbolic probabilistic model checking SPMC. In our example, $EQuery$ calls the function SPMC the first time with S_{I_0}, the MTBDD $f_{S_{M_1}}$ and the property φ. After model checking the system $S_{I_0} \parallel S_{M_1} \models^?$ $P_{\leq\rho}[\psi]$, SPMC returns $false$, this means that the system $S_{I_0} \parallel S_{M_1}$ does not satisfy the property φ.

The function SPMC used in this paper is described in the next section (Sect. 3.1.4).

3.1.4 Symbolic probabilistic model checking

Model checking algorithm for interval MDP was considered in [6,10], where it was demonstrated that the verification of interval MDP is often more consume, in time as well as in space, than the verification of MDP. In this work, our ultimate goal is reducing the size of the state space. Therefore, the verification of interval MDP needs to be avoided. Thus, we propose rather than verifying interval MDP $S_{I_i} \parallel S_{M_1}$, we verify only a restricted SIMDP RI, which is an MTBDD contains the upper probability value of the probability interval associate in each transition of S_{I_i}. This can be done by taking the MTBDD $f^u_{S_{I_i}}$ of S_{I_i}. Then, the verification of $RI \parallel S_{M_1}$ can be done using the standard probabilistic model checking proposed in [25]. The symbolic probabilistic model checking used in this work was proposed in [39].

When $EQuery$ returns $false$, this means either the symbolic assumption is too strong (the upper probability value is big) or the system S does not satisfy the property φ. The symbolic MTBDD-learning algorithm calls the functions $GenerateCex$ and $AnalyseCex$ to generate and analyse the counterexamples.

3.1.5 Generate probabilistic counterexamples

The probabilistic counterexamples are generated when a probabilistic property φ is not satisfied. They provide a valuable feed back about the reason why φ is violated.

Definition 9 The probabilistic property $\varphi = P_{\leq\rho}[\psi]$ is refuted when the probability mass of the path satisfying φ exceeds the bound ρ. Therefore, the counterexample can be formed as a set of paths satisfying φ, whose combined measure is greater than or equal to ρ.

As denoted in Definition 9, the probabilistic counterexample is a set of finite paths, for example, the verification of the property "*a fail state is reached with probability at most 0.01*" is refused by a set of paths whose total probability exceeds 0.01. The main techniques used for the generation of counterexamples are described in [29]. The probabilistic counterexamples are a crucial ingredient in our approach, since they are used to analyse and refine the conjecture symbolic assumptions. Thus, our need consist to find the most indicative counterexample. A most indicative counterexample is the minimal counterexample (which has the least number of paths). A recent work [16] proposed to use causality in order to generate small counterexamples. In this paper, we used the tool DiPro[1] to generate counterexamples. DiPro employs many algorithms to generate counterexamples, among these algorithms we use the K^* algorithm [2].

[1] https://se.uni-konstanz.de/research1/tools/dipro/.

In addition, we apply the algorithms in [17] to generate the most indicative counterexample, denoted by Cex.

3.1.6 Analysis probabilistic counterexamples

The function $AnalyseCex$ aims to check whether the probabilistic counterexample Cex is *real* or not. Cex is a *real* counterexample of the system S if and only if $SM_0^{Cex} \parallel S_{M_1} \models P_{\leq\rho}[\psi]$ does not hold. In practice, the function $AnalyseCex$ creates a fragment of the MDP M_0 based on the probabilistic counterexample Cex, where the MDP fragment M_0^{Cex} contains only transitions present in Cex. Thus, the fragment M_0^{Cex} is obtained by removing from M_0 all states and transitions not appearing in any path of the set Cex.

Since we use symbolic data structures to encode the state space, we encode the MDP fragment using SMDP SM_0^{Cex} (following the same process to encode MDP). $AnalyseCex$ returns $true$ if and only if the symbolic probabilistic model cheeking of $SM_0^{Cex} \parallel S_{M_1} \models P_{\leq\rho}[\psi]$ returns $false$, or $false$ otherwise. The computing of the product $SM_0^{Cex} \parallel S_{M_1}$ follows the same process as the parallel composition of $MDP \parallel IMPD$. The function $AnalyseCex$ is described in Algorithm 5.

Algorithm 5 $AnalyseCex$

Input: a set of probabilistic counterexample Cex, S_{M_0}, S_{M_1}, $\varphi = P_{\leq\rho}[\psi]$
output: Boolean value
Begin
M_0^{Cex} = remove from M_0 all states and transitions not appearing in any path of the set Cex;
SMDP SM_0^{Cex} = Encode M_0^{Cex} as SMDP;
Boolean sat = $SPMC(SM_0^{Cex}, f_{S_{M_1}}, \varphi)$;
return ¬ sat;
End

Example 6 The equivalence query for the initial assumption S_{I_0} returns $false$. The PSCV calls the function $GenerateCex$ to generate the most indicative probabilistic counterexample. For our example, since the system $S_{I_0} \parallel S_{M_1}$ does not satisfy the property φ.
$GenerateCex$ returns the set Cex, where $Cex = \{(s_0, t_0) \xrightarrow{detect,0.2} (s_2, t_2) \xrightarrow{send,0.7} (s_3, t_3)\}$. To check whether the probabilistic counterexample Cex is real or not, the function $AnalyseCex$ generates an MDP fragment M_0^{Cex} (see Fig. 7), and calls the SPMC to model checking the system $SM_0^{Cex} \parallel S_{M_1} \models P_{\leq 0.1}[\psi]$. SPMC returns $false$, this means that Cex is a real counterexample, and the property φ does not satisfy the system S.

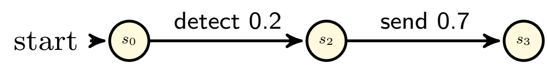

Fig. 7 MDP fragment M_0^{Cex}

3.1.7 Refinement process of the conjecture symbolic assumption S_{I_i}

If the probabilistic counterexample is not real, the symbolic MTBDD-learning algorithm calls the function $Refine\_S_{I_i}$ to refine S_{I_i}. $Refine\_S_{I_i}$ takes the conjecture symbolic assumption S_{I_i} and the probabilistic counterexample Cex as input. First, $Refine\_S_{I_i}$ searches the maximum probability value Max_p in all paths of Cex. Then, it sets the upper probability value for all probability intervals of S_{I_i} presenting in Cex to Max_p. The choice of using Max_p to refine S_{I_i} is to keep the upper bounds of the interval probability value more uniform, i.e. to reduce the number of terminal nodes of the MTBDD encoding S_{I_i}. As described in Algorithm 6, the refinement process of S_{I_i} is based on the set of paths present in Cex. This process could bring the S_{I_i} closer to the original competent M_0. In practice, the use of Max_p could lead to generate a spurious counterexample. To resolve this, and to guarantee the completeness of our approach, we store the last set of counterexample $Last\_Cex$ and the current set Cex, and if a state s is present in the two sets of counterexample, Cex and $Last\_Cex$, then we set all the upper bounds of the outgoing transitions to the corresponding probability value in δ_{M_0}. These states are stored to not change their outgoing transitions in the next refinement iterations.

Algorithm 6 $Refine\_S_{I_i}$

Input: S_{I_i}, Cex
output: S_{I_i}
Begin
Double Max_p = search the maximum probability value in all paths of Cex
Set the upper probability value in $f^u_{S_{I_i}}$ to Max_p if and only if $path \in Cex$
return S_{I_i};
End

Our approach PSCV as well as our algorithm, symbolic MTBDD-learning algorithm, is characterized by:

– *Soundness*, this means if our symbolic MTBDD-learning algorithm generates a symbolic assumption S_{I_i}, where $S_{M_0} \preceq S_{I_i}$ and $S_{I_i} \parallel S_{M_1} \models P_{\leq\rho}[\psi]$, then we are sure that $S_{M_0} \parallel S_{M_1} \models P_{\leq\rho}[\psi]$, the soundness of our approach is based on Theorem 1, this proof rule has been precisely proven in Sect. 3.
– *Completeness*, this means that our symbolic MTBDD-learning algorithm will generate eventually a symbolic

Table 3 Experimental results for case studies randomized mutual exclusion and client–server

Case study	No.	Symbolic monolithic verification			Symbolic compositional verification								Size red (%)
		$S_{M_0} \parallel S_{M_1}$		S_{M_0}	S_{I_0}	$S_{I_f} \parallel S_{M_1}$		SM_0^{Cex}		$SM_0^{Cex} \parallel S_{M_1}$		TT	
		T4MC	Size	Size	Size	T4MC	Size	T4MC	Size	T4MC	Size		
Mutual	6	0.357	9409	1578	736	0.173	4981	0.015	601	0.072	3166	2.547	47
	7	0.556	13,182	3591	1370	0.308	8163	0.021	1229	0.128	4692	2.987	38
	8	0.867	17,531	5658	2197	0.487	11,953	0.066	2367	0.163	6974	3.214	32
	11	2.270	34,034	7882	3217	1.648	26,779	0.199	7350	0.272	15,587	4.129	21
	13	4.010	47,916	10,263	4430	2.739	39,543	0.470	11,867	0.994	22,524	5.214	17
Client–server	2	0.007	616	616	379	0.012	531	0.002	62	0.005	502	0.125	13
	3	0.020	1428	1428	987	0.016	1220	0.003	104	0.007	1213	0.185	14
	4	0.051	2979	2979	1528	0.031	2506	0.005	152	0.009	1401	0.213	15
	5	0.052	4997	4997	3241	0.051	4176	0.009	206	0.012	1600	0.354	16
	6	0.069	7439	7439	4987	0.066	6180	0.010	266	0.014	1810	0.398	17
	7	0.093	10,684	10,684	6893	0.092	8849	0.014	332	0.015	2031	4.218	17

assumption S_{I_i}. Our symbolic MTBDD-learning algorithm always generates a new assumption either by learning an initial assumption S_{I_0} or by refine S_{I_i}.

- *Termination*, this means that our algorithm will always terminate. Our symbolic MTBDD-learning algorithm targets a SIMDP S_{I_i}, where $S_{M_0} \preceq S_{I_i}$. Based on Theorem 1, we can always generate assumptions such $S_{M_0} \preceq S_{I_i}$; thus, the learning algorithm will always terminate, either by true if $S_{M_0} \parallel S_{M_1} \models P_{\leq \rho}[\psi]$ or false otherwise. In the worst case, our algorithm will generate S_{M_0} as a symbolic assumption.
- *Complexity*, another important point is the complexity of our symbolic MTBDD-learning algorithm. Indeed, if we want to apply our algorithm to verify real-life systems then it needs to check these systems within a polynomial number of queries, thus, in a polynomial time. Our algorithm infers the symbolic assumption with at most n equivalence queries, where $n = |\delta_{M_0}|$.

As described before, the main goal of our work is to cope with the state space explosion problem. In the next section (Sect. 4), we apply our PSCV approach in a several case studies and we compare its performance against the symbolic monolithic probabilistic verification.

4 Implementation and experimental results

We have implemented a prototype tool named *PSCV4MDP* (probabilistic symbolic compositional verification for MDP). Our tool accepts MDP specified using PRISM code and a probabilistic safety property as input and returns either *true* if the MDP satisfies the probabilistic safety property, or *false* and counterexample otherwise. In this section, we give the results obtained for the application of our approach in a several case studies derived from the PRISM benchmark.[2] For each case study, we check the model against a probabilistic safety property using: (1) symbolic monolithic probabilistic model checking and (2) symbolic probabilistic compositional verification. In addition, we compare, for each technique, the time for model construction *T4MC*, and the size of the state space, i.e. the number of nodes. We have summarized and compared the results in Tables 3 and 4. In Table 3 we report the results of the case studies *randomized mutual exclusion* and *Client–Server*; for these two case studies, it was necessary to generate counterexamples to refine the initial assumption, contrary to Table 4, where the initial assumption was sufficient for the verification process. For both tables, the column (No.) shows the number of components and the column [Red (%)] shows the reduction of model size of our PSCV to the symbolic monolithic verification. The tests were carried on a personal computer with Linux as operating system, 2.30 GHz CPU and 4GB RAM.

In addition, for each case study, we report the size of $S_{M_0} \parallel S_{M_1}$ for the symbolic monolithic verification or $S_{I_f} \parallel S_{M_1}$ for the symbolic compositional verification, where S_{I_f} is the final symbolic assumption. We also report in Table 3 the T4MC and the size of $SM_0^{Cex} \parallel S_{M_2}$ and S_{M_2}. We observe that the PSCV successfully infers symbolic assumptions and $S_{I_f} \parallel S_{M_1}$ are much more compact than $S_{M_0} \parallel S_{M_1}$ in all cases. For example, in the first case study (mutual $N = 11$), $S_{I_f} \parallel S_{M_1}$ has 26,779 nodes while $S_{M_0} \parallel S_{M_1}$ has 34,034 nodes; in R. S. stab. ($N = 17$), $|S_{M_0} \parallel S_{M_1}| = 969$ while $|S_{I_f} \parallel S_{M_1}| = 53$. The column TT illustrates the total time for PSCV to check if the case study satisfies the property or not, i.e. $TT = T4MC$ (for PSCV) + Time to generate S_{I_f}.

[2] http://www.prismmodelchecker.org/casestudies/index.php.

Table 4 Experimental results for the case studies randomized dining philosophers, randomized self-stabilizing algorithm and dice

Case study	No.	Symbolic monolithic verification			Symbolic compositional verification				Size red (%)
		$S_{M_0} \parallel S_{M_1}$		S_{M_0}	S_{I_0}	$S_{I_f} \parallel S_{M_1}$		TT	
		T4MC	Size	Size	Size	T4MC	Size		
R.D. Philos	6	0.090	5008	956	705	0.125	1887	0.192	62
	16	7.091	37,523	9215	5035	0.344	20,127	0.521	46
	18	11.042	43,472	11,749	6372	0.437	25,470	0.982	41
	20	16.563	58,568	14,570	7866	0.50	31,441	1.320	47
	30	64.207	131,260	32,980	17,691	1.657	70,716	2.520	46
	40	169.215	232,655	58,565	31,441	2.704	125,691	3.391	45
R.S. Stab.	6	0.003	177	117	20	0.003	20	0.010	88
	8	0.005	285	285	26	0.004	26	0.010	89
	13	0.006	625	625	41	0.005	41	0.012	93
	17	0.012	969	969	53	0.007	53	0.015	94
	19	0.013	1165	1165	59	0.009	59	0.015	94
Dice	2	0.003	437	71	11	0.009	240	0.011	45
	3	0.005	793	203	167	0.011	597	0.015	24
	4	0.007	1181	378	268	0.020	986	0.025	17
	5	0.029	1601	616	487	0.023	1407	0.029	12
	8	0.081	3053	917	770	0.050	2862	0.059	6
	12	0.120	5437	1218	1117	0.108	5250	0.102	3

4.1 Randomized mutual exclusion

Our first case study, randomized mutual exclusion (Mutual), is based on Pnueli and Zuck's [40] probabilistic symmetric solution to the n-process mutual exclusion problem. The model is represented as an MDP. We let No. denotes the number of processes. We check the system against the property: φ_1 = *The probability that two or more processes are simultaneously in their remainder phases is at most 0.999*. The results for this case study are reported in Fig. 8. The results show that PSCV performs better then the symbolic monolithic verification and reduces the model size by 31% on average. For time to model checking this case study, the symbolic monolithic verification performs better than PSCV, and this is due to time to generate counterexample and refine assumptions.

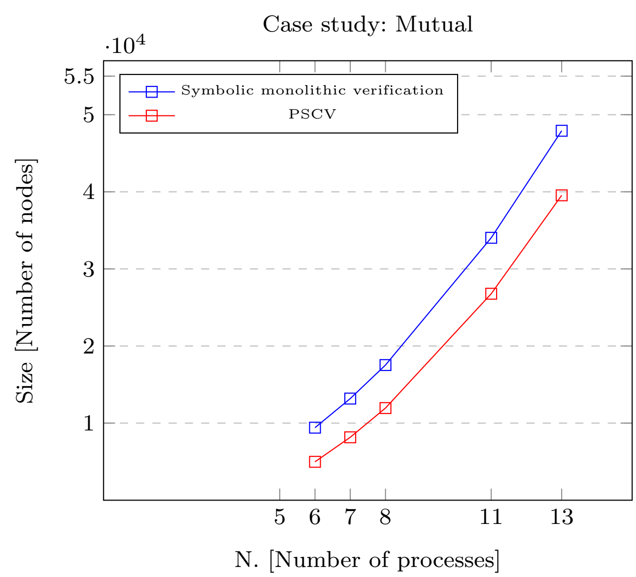

Fig. 8 Case study results from the application of symbolic monolithic verification and PSCV: Mutual

4.2 Client–server

This case study is a variant of the Client–Server model from [41]. It models a server and N clients. The server can grant or deny a client's request for using a common resource, once a client receives permission to access the resource, it can either use it or cancel the reservation. Failures might occur with certain probability in one or multiple clients, causing the violation of the mutual exclusion property (i.e. conflict in using resources between clients). In this case study, we consider the property: φ_5 = *the probability a failure state is reached is at most 0.98*. For this case study, PSCV reduces the size by 15.33% on average (Fig. 9).

4.3 Randomized dining philosophers

The third case study is the randomized dining philosophers (R.D. Philos). This case study models a randomized solution

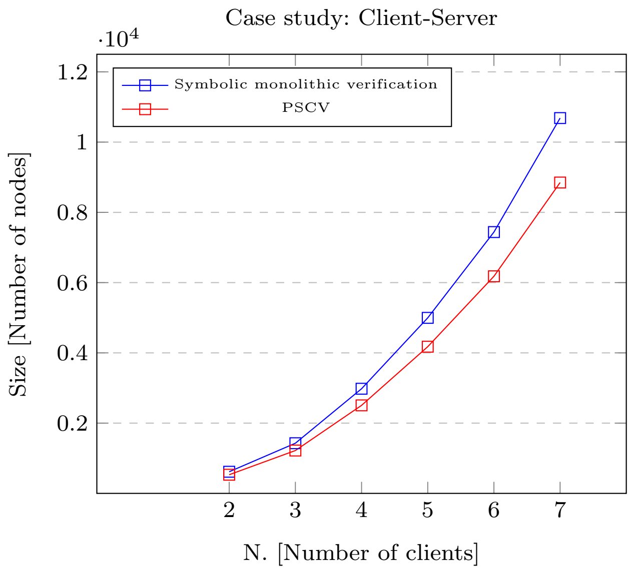

Fig. 9 Case study results from the application of symbolic monolithic verification and PSCV: client–server

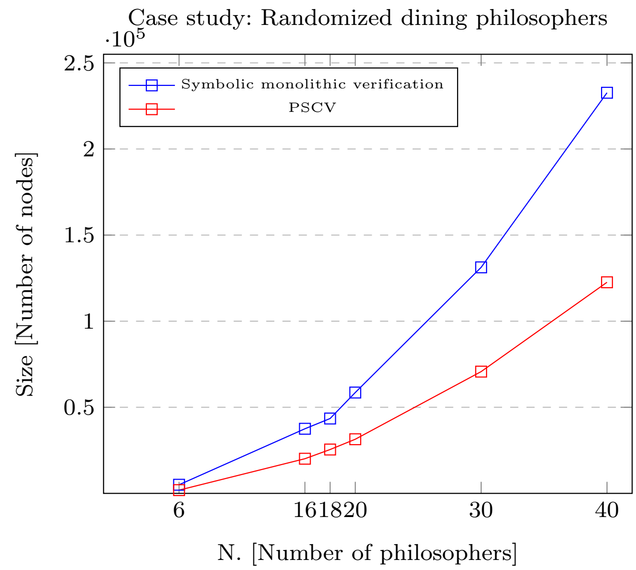

Fig. 10 Case study results from the application of symbolic monolithic verification and PSCV: randomized dining philosophers

to the dining philosophers problem, proposed by [18,35]. R.D. Philos concerns the problem of resources allocation between processes. Several philosophers sit around a circular table. There is a fork between each pair of neighbouring philosophers. A philosopher can eat if and only if he obtains the resources from both sides. In this case study, No. denotes the number of philosophers. We analyse the solution proposed in [18,35] by using the property: $\varphi_2 =$ *the probability that philosophers do not obtain their shared resource simultaneously is at most 0.980*, formally: $P_{\leq 0.980}[\Diamond "err"]$, where label *"err"* sands for every states satisfy: $[(s_N \geq 8)\&(s_N \leq 9)]$, and N is the component number. Results are reported in Fig. 10. In this case study, PSCV reduces the model size by 47.66% on average and improves the verification time as well.

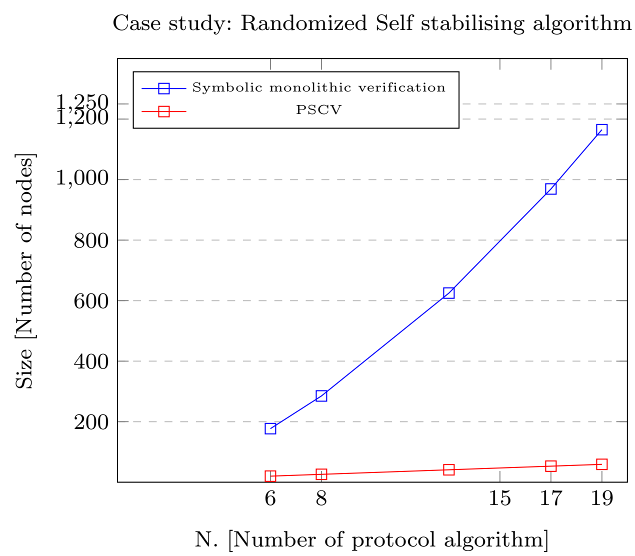

Fig. 11 Case study results from the application of symbolic monolithic verification and PSCV: randomized self-stabilizing algorithm

4.4 Randomized self-stabilizing algorithm

In the fourth case study we consider a number of randomized self-stabilizing algorithms (R.S. Stab.). A randomized self-stabilizing protocol for a network of processes is a protocol which, when started from some possibly illegal start configuration, returns to a legal/stable configuration without any outside intervention within some finite number of steps. In this paper, we consider the solution of Israeli and Jalfon [28] and we analyse the protocol through the following property: $\varphi_3 =$ *the probability to reach a stable configuration for all algorithms is at most 0.999*. Experimental results are reported in Fig. 11. For this case study, the verification time is improved and the model size is reduced by 91.60% on average.

4.5 Dice

This case study considers probabilistic programs, due to Knuth and Yao [30], which model fair dice using only fair coins. At the implementation, we have used a reimplementation of the work done by [27]. The probabilistic programs are modelled using discrete time Markov chain (DTMC), which constitute a subclass of MDP. We consider the analysis of the property: $\varphi_4 =$ *the probability of reaching a state with* $s = 7\,\&\,d = k$ *is at most 0.01, where* $k \in [1\ldots 6]$. Performing the verification of this case study, the PSCV reduces the size by 17.83% on average (Fig. 12).

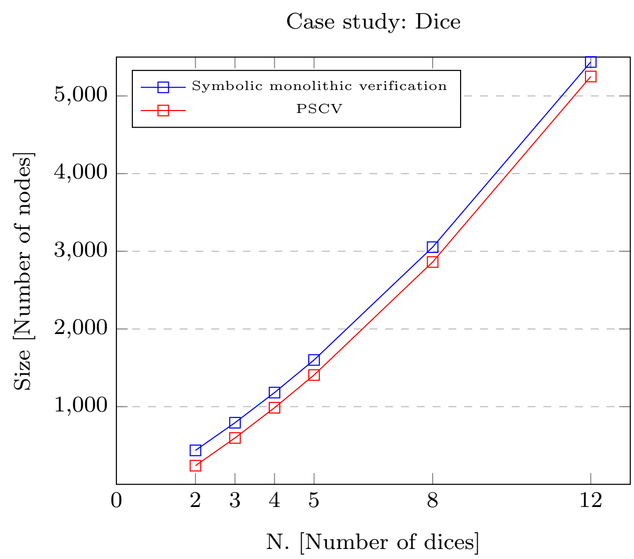

Fig. 12 Case study results from the application of symbolic monolithic verification and PSCV: dice

4.6 Discussion

Table 3 reports the results for case studies *Mutual* and *Client–Server*. The average size reduction for these case studies is 22.54%. For *Mutual* model, the reduction size decreases by the number of components. As described in our paper, the size reduction depends on the number of terminal nodes and the combining process of isomorphic sub-tree. For this case study, the initial assumption S_{I_0} was too string for PSCV to check if $M_0 \parallel M_1 \models P_{\leq \rho}[\psi]$; thus, PSCV refined S_{I_0}. At each refinement iteration, assumptions converge to the original component M_0, and this could lead to add other terminal nodes and many non-terminal nodes. Contrary to the first case study, the size reduction for *Client–Server* model is stable. For the model checking time, the symbolic monolithic verification performs better than PSCV, and this is due essentially to the time needed to analyse and refine the assumptions. The refinement process and counterexample generation consume more time than the monolithic verification. In Table 4, we report the results for case studies *R.D. Philos*, *R.S. Stab.* and *Dice*. For these case studies, S_{I_0} was sufficient for the PSCV verification process. The average size reduction for these examples is 50.11. The reduction size for *R.S. Stab* model is stable, contrary to *R.D. Philos* and *Dice*, where the size reduction decrease by the number of components. Indeed, in these examples the size reduction depends on the number of terminal nodes and the combining process of isomorphic sub-tree.

The overall results show that PSCV is more effective than the symbolic monolithic verification. The PSCV successfully generates a symbolic assumption that establish the verification process. Moreover, the model size is reduced by 39.25% on average for the 28 cases (reported in Tables 3 and 4). The success of our approach, comparing with the symbolic monolithic verification, comes from the fact that PSCV avoids the construction of the whole model.

5 Related works

In this section, we review some research works related to the symbolic probabilistic model checking, compositional verification and assume-guarantee reasoning. Verification of probabilistic systems has been addressed by Vardi and Wolper [44–46], then by Pnueli and Zuck [40] and by Baier and Kwiatkowska [5]. The symbolic probabilistic model checking algorithms have been proposed by [12,39]. These algorithms have been implemented in a symbolic probabilistic model checker PRISM [33]. Model checking algorithm for interval MDP was considered in [6,10]. An important step in our approach is the generation of small counterexample. The main techniques used to generate counterexamples were detailed in [29]. A recent work [16] proposed to use causality in order to generate small counterexamples; the authors of this work propose to use the tool DiPro to generate counterexamples; then they applied an aided-diagnostic method to generate the most indicative counterexample [17]. For the compositional verification of non-probabilistic systems, several frameworks have been developed using the assume-guarantee reasoning approach [11,13,41]. The compositional verification of probabilistic systems has been a significant progress in these last years [8,21,22,25,32]. Our approach is inspired by the work of [21,22]. In this work, assumptions are represented as deterministic finite automata (DFA) and the classical L^* learning algorithm has been applied to infer assumptions. However, this work used a non-complete assume-guarantee reasoning rule, and the generation of an assumption to establish the compositional verification is not guaranteed. Another work relevant to ours is [25]. This work proposed the first sound and complete learning-based composition verification technique for probabilistic safety properties, where they used an adapted L^* learning algorithm to learn weighted automata as assumptions, then they transformed them into MTBDD. In [37] authors proposed an algorithm for automatically learning a deterministic labelled Markov decision process model from the observed behaviour of a reactive system. The proposed learning algorithm adopt a passive learning model, which was adapted from algorithms for learning deterministic probabilistic finite automata, and extended to include both probabilistic and non-deterministic transitions.

6 Conclusion

In this paper, we proposed a fully automated probabilistic symbolic compositional verification approach (PSCV) to verify probabilistic systems, where each component is an MDP. The PSCV is based on our proposed symbolic assume-guarantee reasoning rule, which describes the symbolic compositional verification process. The main characteristics of our symbolic assume-guarantee reasoning rule are completeness and soundness. The completeness comes from the use of IMDP to represent the assumptions. Since we aim to overcome the state space explosion problem when verifying probabilistic safety properties, the system components and assumptions are represented by a compact symbolic data structures, i.e. SMDP and SIMDP. In addition, we proposed a symbolic MTBDD-learning algorithm to construct automatically these assumptions, where it used causality to generate small counterexamples in order to refine the conjecture symbolic assumptions. The experimental results are encouraging and demonstrated that our approach can reduce the state space by learning symbolic assumptions.

Our actual approach can be applied to sequential probabilistic systems. We plan to propose other assume-guarantee reasoning rule such as asymmetric rule or circular rule to verify n-components. Since we use interval MDP to represent assumptions, our approach can be easily extended to verify properties of the form $\varphi = P_{\geq P}[\psi]$. Moreover, we plan to extend the PSCV to verify other probabilistic properties such as liveness.

References

1. Abate A, Prandini M, Lygeros J, Sastry S (2008) Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. Automatica 44(11):2724–2734
2. Aljazzar H, Leue S (2011) K^*: a heuristic search algorithm for finding the k shortest paths. Artif Intell 175(18):2129–2154
3. Angluin D (1987) Learning regular sets from queries and counterexamples. Inf Comput 75(2):87–106
4. Baier C, Katoen JP, Larsen KG (2008) Principles of model checking. MIT Press, Cambridge
5. Baier C, Kwiatkowska M (1998) Model checking for a probabilistic branching time logic with fairness. Distrib Comput 11(3):125–155
6. Benedikt M, Lenhardt R, Worrell J (2013) LTL model checking of interval Markov chains. In: International conference on tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 32–46
7. Bouchekir R, Boukhedouma S, Boukala MC (2016) Symbolic probabilistic analysis and verification of inter-organizational workflow. In: International conference on information technology for organizations development (IT4OD). IEEE, pp 1–8
8. Bouchekir R, Boukhedouma S, Boukala M (2016) Automatic compositional verification of probabilistic safety properties for inter-organisational workflow processes. In: Proceedings of the 6th international conference on simulation and modeling methodologies, technologies and applications—vol 1: SIMULTECH, ISBN:978-989-758-199-1, pp 244–253
9. Burch JR, Clarke EM, McMillan KL, Dill DL, Hwang LJ (1992) Symbolic model checking: 10^{20} states and beyond. Inf Comput 98(2):142–170
10. Chatterjee K, Sen K, Henzinger TA (2008) Model-checking ω-regular properties of interval Markov chains. In: International conference on foundations of software science and computational structures. Springer, Berlin, pp 302–317
11. Chen YF, Clarke M, Farzan A, Tsai M, Tsay YK, Wang BY (2010) Automated assume-guarantee reasoning through implicit learning. In: International conference on computer aided verification. Springer, Berlin, pp 511–526
12. Ciesinski F, Baier C, Grober M, Parker D (2008) Generating compact MTBDD-representations from Probmela specifications. In: International SPIN workshop on model checking of software. Springer, Berlin, pp 60–76
13. Cobleigh JM, Giannakopoulou D, Pasareanu CS (2003) Learning assumptions for compositional verification. In: International conference on tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 331–346
14. Clarke EM, Emerson EA (1981) Design and synthesis of synchronization skeletons using branching time temporal logic. In: Workshop on logic of programs. Springer, Berlin, pp 52–71
15. Clarke EM, Emerson EA, Sistla AP (1986) Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans Program Lang Syst (TOPLAS) 8(2):244–263
16. Debbi H, Debbi A, Bourahla M (2016) Debugging of probabilistic systems using structural equation modelling. Int J Crit Comput Based Syst 6(4):250–274
17. Debbi H, Bourahla M (2013) Generating diagnoses for probabilistic model checking using causality. CIT. J Comput Inf Technol 21(1):13–22
18. Duflot M, Fribourg L, Picaronny C (2004) Randomized dining philosophers without fairness assumption. Distrib Comput 17(1):65–76
19. Fujita M, McGeer PC, Yang JY (1997) Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. Form Methods Syst Des 10(2–3):149–169
20. Fehnker A, Gao P (2006) Formal verification and simulation for performance analysis for probabilistic broadcast protocols. In: International conference on ad-hoc networks and wireless. Springer, Berlin, pp 128–141
21. Feng L, Kwiatkowska M, Parker D (2010) Compositional verification of probabilistic systems using learning. In: Seventh international conference on the quantitative evaluation of systems (QEST). IEEE, pp 133–142
22. Feng L (2013) On learning assumptions for compositional verification of probabilistic systems. Doctoral dissertation, University of Oxford
23. Hasson H, Jonsson B (1994) A logic for reasoning about time and probability. Form Asp Comput 6:512–535
24. Hart S (1984) Probabilistic temporal logics for finite and bounded models. In: Proceedings of the sixteenth annual ACM symposium on theory of computing. ACM, pp 1–13
25. He F, Gao X, Wang M, Wang BY, Zhang L (2016) Learning weighted assumptions for compositional verification of Markov decision processes. ACM Trans Softw Eng Methodol 25(3):21
26. Hermanns H, Kwiatkowska M, Norman G, Parker D, Siegle M (2003) On the use of MTBDDs for performability analysis and verification of stochastic systems. J Log Algebr Program 56(1–2):23–67
27. Hurd J (2003) Formal verification of probabilistic algorithms (No. UCAM-CL-TR-566). University of Cambridge, Computer Laboratory
28. Israeli A, Jalfon M (1990) Token management schemes and random walks yield self-stabilizing mutual exclusion. In: Proceedings

of the ninth annual ACM symposium on principles of distributed computing. ACM, pp 119–131
29. Jansen N, Wimmer R, Abraham E, Zajzon B, Katoen JP, Becker B, Schuster J (2014) Symbolic counterexample generation for large discrete-time Markov chains. Sci Comput Program 91:90–114
30. Knuth D (1979) The complexity of nonuniform random number generation. In: Algorithm and complexity, new directions and results. Academic press, pp 357–428
31. Komuravelli A, Pasareanu CS, Clarke EM (2012) Learning probabilistic systems from tree samples. In: Proceedings of the 2012 27th annual IEEE/ACM symposium on logic in computer science. IEEE Computer Society, pp 441–450
32. Kwiatkowska M, Norman G, Parker D, Qu H (2010) Assume-guarantee verification for probabilistic systems. In: International conference on tools and algorithms for the construction and analysis of systems. Springer, Berlin, pp 23–37
33. Kwiatkowska M, Norman G, Parker D (2011) PRISM 4.0: verification of probabilistic real-time systems. In: International conference on computer aided verification. Springer, Berlin, pp 585–591
34. Lehmann D, Shelah S (1982) Reasoning with time and chance. Inf Control 53(3):165–198
35. Lehmann D, Rabin MO (1981) On the advantages of free choice: a symmetric and fully distributed solution to the dining philosophers problem. In: Proceedings of the 8th ACM SIGPLAN-SIGACT symposium on principles of programming languages. ACM, pp 133–138
36. Larsen KG, Pettersson P, Yi W (1995) Compositional and symbolic model-checking of real-time systems. In: 16th IEEE proceedings of real-time systems symposium, 1995. IEEE, pp 76–87
37. Mao M, Chen Y, Jaeger M, Nielsen TD, Larsen KG, Nielsen B (2012) Learning Markov decision processes for model checking. arXiv preprint arXiv:1212.3873
38. McMillan KL (1993) Symbolic model checking. In: Symbolic model checking. Springer, New York, pp 25–60
39. Parker DA (2002) Implementation of symbolic model checking for probabilistic systems. Doctoral dissertation, University of Birmingham
40. Pnueli A, Zuck L (1986) Verification of multiprocess probabilistic protocols. Distrib Comput 1(1):53–72
41. Pasareanu CS, Giannakopoulou D, Bobaru MG, Cobleigh JM, Barringer H (2008) Learning to divide and conquer: applying the L^* algorithm to automate assume-guarantee reasoning. Form Methods Syst Des 32(3):175–205
42. Segala R (1995) Modelling and verification of randomized distributed real time systems. Ph.D. thesis, Massachusetts Institute of Technology
43. Steel G (2006) Formal analysis of PIN block attacks. Theor Comput Sci 367(1–2):257–270
44. Vardi MY (1985) Automatic verification of probabilistic concurrent finite state programs. In: 26th annual symposium on foundations of computer science. IEEE, pp 327–338
45. Vardi MY, Wolper P (1994) Reasoning about infinite computations. Inf Comput 115(1):1–37
46. Vardi MY (1999) Probabilistic linear-time model checking: an overview of the automata-theoretic approach. In: International AMAST workshop on aspects of real-time systems and concurrent and distributed software. Springer, Berlin, pp 265–276