Automated Compositional Verification for Probabilistic Systems Through Implicit Learning

Redouane Bouchekir
MOVEP, Computer Science Department
USTHB
BP 32 El-Alia, Algiers, Algeria
Email: rbouchekir@usthb.dz

Mohand Cherif Boukala
MOVEP, Computer Science Department
USTHB
BP 32 El-Alia, Algiers, Algeria
Email: mboukala@usthb.dz

***Abstract*—In this paper, we propose an automated computational verification to verify probabilistic systems, where each system component is a Markov Decision Process (MDP). Our approach starts by encoding implicitly the system components using Boolean functions. Different from the monolithic verification, in our compositional verification, each system component is verified in isolation under an assumption about its contextual environment. To establish this compositional verification process, we propose a novel sound and complete symbolic assume-guarantee reasoning rule. This rule uses CDNF learning algorithm to generate the series of symbolic probabilistic assumptions automatically.**

***Keywords*—Probabilistic model checking · Compositional verification · Symbolic model checking · Assume-guarantee paradigm · Machine learning.**

I. Introduction

A. Context and Motivation

An important feature of probability systems is their complexity, i.e. their propensity to have a very large number of possible behaviours. This is due mainly to the concurrent execution and the multiple interactions of the different components that make up a complex system. This makes their design, implementation and verification extremely difficult. This difficulty is enhanced by the often critical role of this type of system (avionics control process, nuclear power plants, etc.). These particular characteristics of complex probabilistic systems make the verification phases crucial. *Probabilistic verification* is a set of techniques for formal modelling and analysis of such systems. *Probabilistic model checking*[1],[2],[3] involves the construction of a finite-state model augmented with probabilistic information, such as Markov chains or probabilistic automaton [20],[16]. This is then checked against properties specified in probabilistic extensions of temporal logic, such as Probabilistic Computation Tree Logic (PCTL) [15].

Formal methods, including the Probabilistic Model Checking [1],[2],[3], have been successfully applied to probabilistic systems of reasonable size, which means that they can be managed by the main memory. However, the most of critical probabilistic systems are complex and possess large state spaces. Indeed, the state space represents all the possible behaviours of the probabilistic system. Generally, the number of states increases exponentially. This important evolution in the number of states commuted under the name of the state space explosion problem. This problem constitutes, even after several years of research, the main obstacle of probabilistic model checking.

To cope with the state space explosion problem, several approaches have been proposed such as: (i) *Compositional verification* [21],[12],[13],[14] and (ii) *Symbolic model checking* [6],[22]. Compositional verification suggests a "divide and conquer" strategy to reduce the verification task into simpler subtasks. A popular approach is the *assume-guarantee* paradigm [9],[25],[7], in which individual system components are verified under *assumptions* about their environment. Once 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 machine learning [7], [13], such as $CDNF$ learning algorithm [4]. 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 Boolean functions, as well as the initial states and transition relation of the system. To model checking probabilistic systems encoded using Boolean function, the Boolean function should be converted to another data structures such as Binary Decision Diagrams(BDD) or Multi Terminal BDD(MTBDD) [11], this is due to the absence of SAT-based model checking for probabilistic systems.

In this paper, we present a novel approach for the compositional verification for probabilistic systems through implicit learning. Our aim is to reduce the size of the state space. For that, we propose to encode the system components using Boolean function. This encoding allows to store and explore a large number of states efficiently [7]. We use this Boolean functions as input of the CDNF learning algorithm. This algorithm generates an assumption which simulates a set of components. The idea is to use this assumption for the verification instead of the real system components. To establish the verification process i.e. to guarantee that the generated assumption simulates all the possible behaviour of the set of components, we proposed a sound and complete symbolic assume-guarantee reasoning rule. In this rule, the assumption are represented using Interval MDP. We have illustrated our approach using a simple example.

B. Contributions

Our contributions are summarized as follows: (i) We propose a full automated compositional verification for probabilistic systems using CDNF learning algorithm, (ii) We give a sound and complete symbolic assume-guarantee reasoning rule, (iii) We describe the process of encoding MDP using Boolean functions, (iiii) We apply our approach for the verification of a simple example.

The remainder of this paper is organized as follows: In section II we provide the most relevant works to our work. Section III provides some background knowledge about MDP, Interval MDP and the parallel composition MDP || IMDP. In Section IV, we present our approach, where we detail the process of encoding MDP using Boolean function, our symbolic assume-guarantee reasoning proof rule and the application of the CNDF learning algorithm to generate assumptions. Section V concludes the paper and talks about future works.

II. 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 have been addressed by Vardi and Wolper [27], [28],[29], and then by Pnueli and Zuck [26], and by Baier and Kwiatkowska [3]. The symbolic probabilistic model checking algorithms have been proposed by [24], [8]. These algorithms have been implemented in a symbolic probabilistic model checker PRISM [18]. The main techniques used to generate counterexamples was detailed in [17]. A recent work [10] proposed to use causality in order to generate small counterexamples, the authors of this work propose to used the tool DiPro to generate counterexamples, then they applied an aided-diagnostic method to generate the most indicative counterexample. For the compositional verification of non-probabilistic systems, several frameworks have been developed using the assume-guarantee reasoning approach [9], [25], [7]. The compositional verification of probabilistic systems has been a significant progress in these last years [19],[12],[13], [14]. Our approach is inspired by the work of [12],[13]. In this work, they consider the verification of Discrete Time Markov Chains DTMC, and they proposed to use CDNF learning algorithm to infer assumptions. Another work relevant to ours is [14]. 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.

III. Preliminaries

In this section, we give some background knowledge about MDP and IMDP. 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, $\delta_M \subseteq S \times (\Sigma_M \cup \{\tau\}) \times Dist(S)$ 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 $(s_0, a_0, \mu_0) \rightarrow (s_1, a_1, \mu_1) \rightarrow ....$ To reason about MDP, we use the notion of adversaries, which resolve the non deterministic choices in MDP, based on its execution history. Formally, an adversary maps any finite path to a distribution over the available transitions in the last state on the part. An example of two MDP M_0 and M_1 is shown in Fig. 5.

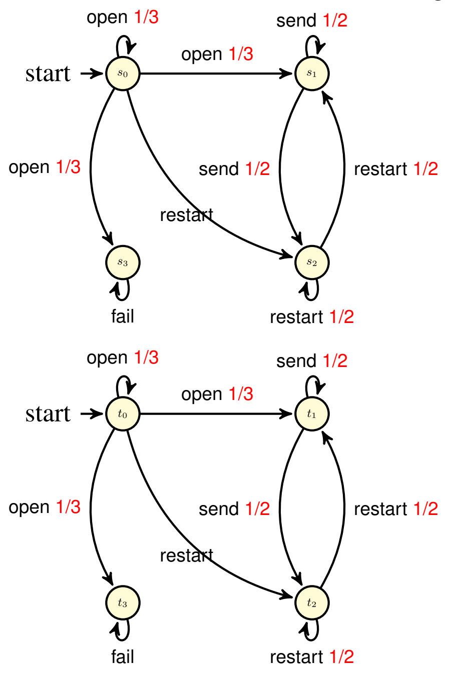

Fig. 1: Example of two MDP, M_0 (above) and M_1 (below).

Interval Markov Chains (IMDP) generalize ordinary MDP by having interval-valued transition probabilities rather than just probability value. In this paper, we use IMDP to represent the assumptions used in our compositional verification.

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: $P^l(s, s') \leq P^u(s, s')$ for all states $s, s' \in S$.

An example of IMDP I is shown in Fig. 2.

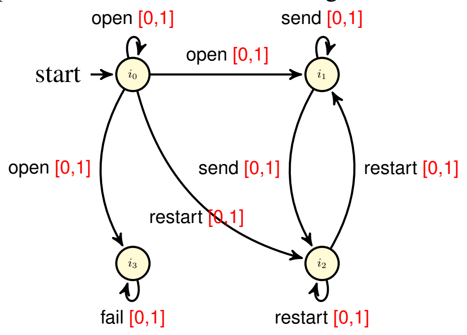

Fig. 2: Example of IMDP I.

In Definition 3, we describe how MDP and IMDP are composed together. This is done by using the asynchronous parallel operator (‖) defined by [23], where MDP and IMDP synchronise over shared actions and interleave otherwise.

Definition 3: Parallel composition $MDP \parallel IMDP$ Let M and I be MDP and IMDP, respectively. Their parallel composition, denoted by $M \parallel I$, is an IMDP MI, where $MI = M \parallel I$.
$MI = \{S_M \times S_I, (s_0^M, s_0^I), \Sigma_M \cup \Sigma_I, P^l, P^u, L\}$, where P^l, P^u are defined such that:
$(s_i, s_j) \xrightarrow{a} [P^l(s_i, s_j) \times \mu_1, P^u(s_i, s_j) \times \mu_2]$ if and only if one of the following holds: let $s_i,\ s_i' \in \Sigma_M$ and $s_j,\ s_j' \in \Sigma_I$.

- $s_i \xrightarrow{a,\mu_1} s_i', s_j \xrightarrow{a,[P^u(s_j),P^l(s_j)]} s_j'$, where $a \in \Sigma_M \cap \Sigma_I$,
- $s_i \xrightarrow{a,\mu_1} s_i'$, where $a \in \Sigma_M \setminus \Sigma_I$,
- $s_j \xrightarrow{a,[P^u(s_j),P^l(s_j)]} s_j'$, where $a \in \Sigma_M \setminus \Sigma_I$.

Example 1: To illustrate the parallel composition, we consider the example of MDP M_0 and IMDP I shown in Fig. 5 and 2, respectively. The product $MI = M_0 \parallel I$ obtained from their parallel composition is shown in Fig. 3. M_0 and I synchronise over their shared actions $\Sigma_{M_0} \cap \Sigma_I = \{open, send, restart, fail\}$.

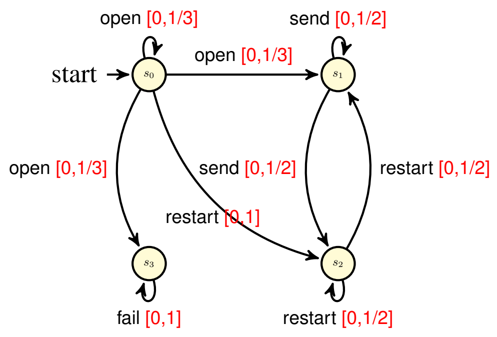

Fig. 3: IMDP MI result of the parallel composition $M_0 \parallel I$.

In this work we use the symbolic model checking to verify if a system $M_0 \parallel I$ satisfies a probabilistic safety property. The symbolic Model checking uses BDD and MTBDD to encode the state space. It is straightforward to convert a Boolean function to a BDD/MTBDD.

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.

IV. Probabilistic Symbolic Compositional Verification

Probabilistic symbolic compositional verification aims to mitigate the state explosion problem. Figure 4 illustrates an overview of our approach. The first step consists to encode the system components as a Boolean function. This encoding allows to store and explore a large number of states efficiently. We use this Boolean functions as input of the CDNF learning algorithm. This algorithm generates a conjecture assumption I, which simulates the component M_0. The idea is to use this assumption I for the verification instead of M_0. To establish the verification process i.e. to guarantee that I simulates all the possible behaviour of M_0, we propose a sound and complete symbolic assume-guarantee reasoning rule. Indeed, the conjecture assumption needs to verify both premises of our symbolic assume-guarantee rule, otherwise, we generate a counterexample (Ctx) which we use it to refine I. In this section, we detail all steps of our approach as well as our symbolic assume-guarantee rule.

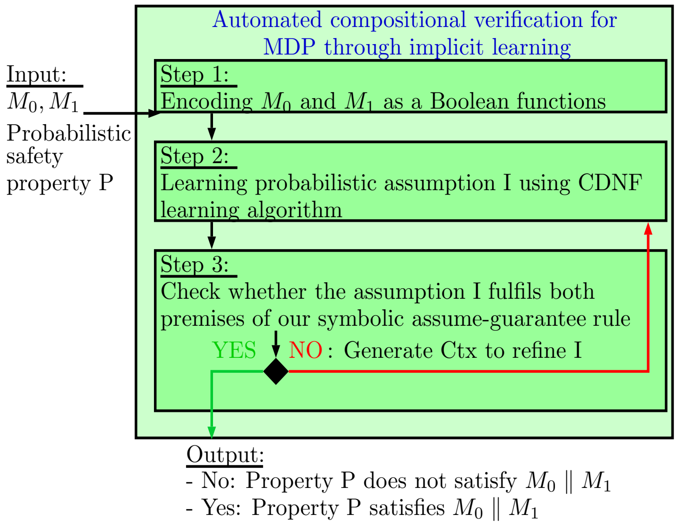
Automated compositional verification for MDP through implicit learning
Input:
M_0, M_1
Probabilistic safety property P
Step 1:
Encoding M_0 and M_1 as a Boolean functions
Step 2:
Learning probabilistic assumption I using CDNF learning algorithm
Step 3:
Check whether the assumption I fulfils both premises of our symbolic assume-guarantee rule
YES
NO: Generate Ctx to refine I
Output:
- No: Property P does not satisfy $M_0 \parallel M_1$
- Yes: Property P satisfies $M_0 \parallel M_1$

Fig. 4: An overview of our approach, in step 1 we encode MDP as Boolean functions, step 2 aims to learn a probabilistic assumption I and step 3 checks whether the assumption I fulfils both premises of our symbolic assume-guarantee rule.

A. Encoding MDP as Boolean Functions

MDP can be encoded implicitly as Boolean functions, we denote Symbolic MDP (SMDP) the Boolean functions encoding an MDP.

Definition 6: Symbolic MDP (SMDP) is a tuple $S_M = (X,\ Init_M,\ Y, f_{S_M}\ (yxx'e^le^u))$ where X and Y are finite ordered set of Boolean variables with $X \cap Y = \emptyset$. $Init(X)$ is an initial predicate over X and $f_{S_M}(yxx'e^le^u)$ is a transition predicate over $Y \cup X \cup X' \cup E$ where y, x, x', e^l, e^u are predicates of receptively, Y, X, X' and E. The set X encodes the states of S, X' next states, Y encodes alphabets and E encodes the probability values.

More concretely, let $M = (S_M, s_0, \Sigma_M, \delta_M, L)$ 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 $S_M \times \Sigma_M \times \{1, 2, ..., r\} \times S_M \to [0, 1]$. We use a function $enc : S_M \to \{0, 1\}^k$ over $X = \langle x_1,\ x_2,\ ..., x_k \rangle$ to encode states in S_M and $X' = \langle x_1', x_2', ..., x_k' \rangle$ to encode next states. We use also $Y = \langle\ y_1,\ y_2,\ ..., y_m\ \rangle$ to encode atomic propositions and we represent the probability values using $E = \langle e_1^l, e_1^u, e_2^l, e_2^u, ..., e_t^l, e^t \rangle$, where t is the number of distinct probability value in δ_M. $f_{S_M}\ (yxx'e^le^u)$ encodes the probabilistic transition relation Σ_M as a disjunction over a set of transition formulae, where each formula encodes a transition between two states. Suppose a transition $s \xrightarrow{a,p} s'$, we encode the state s, the next state s' and the action a using respectively $enc(s)$, $enc(s')$ and $enc(a)$, where enc is a function encodes: (i) states s over Boolean variable set X, (ii) next states s' over Boolean variable set X', and (iii) actions over Boolean variable Y. In addition, to encode the probability value p, we use the Boolean variables e^l and e^u, where e^l e^u encode predicates of the form $p \geq \mu(s, s')$ and $p \leq \mu(s, s')$ respectively. Thus, a transition of the from $s \xrightarrow{a,p} s'$ can be encoded as: $enc(y) \wedge enc(s) \wedge enc(s') \wedge e^l \wedge e^u$.

Example 2: To illustrate how we encode MDP as SMDP, we consider the MDP M_0 (Fig. 5). M_0 contains the set of states $S_{M_0} = \{s_0,\ s_1,\ s_2,\ s_3\}$ and the set of actions $\Sigma_{M_0} = \{open,\ send,\ restart,\ fail\}$. We use $X = \langle x_0, x_1 \rangle$ to encode the set of states in S_{M_0} as: $enc(s_0) = \neg x_0 \wedge \neg x_1$, $enc(s_1) = \neg x_0 \wedge x_1$, $enc(s_2) = x_0 \wedge \neg x_1$, $enc(s_3) = x_0 \wedge x_1$; and we use the set $Y = \langle o, s, r, f \rangle$ to encode the actions $\{open,\ send,\ restart,\ fail\}$, respectively. Table I summarizes the process of encoding the transition function δ_{M_0}. The transition relation of M_0 is encoded as:

$$f_{S_M}\ (yxx'e^le^u) = \begin{array}{l} ((o \wedge \neg x_0 \wedge \neg x_1 \wedge \neg x_0' \wedge \neg x_1' \wedge e_2^l \wedge e_2^u) \\ \vee(o \wedge \neg x_0 \wedge \neg x_1 \wedge \neg x_0' \wedge x_1' \wedge e_2^l \wedge e_2^u) \\ \vee(o \wedge \neg x_0 \wedge \neg x_1 \wedge x_0' \wedge x_1' \wedge e_2^l \wedge e_2^u) \\ \vee(r \wedge \neg x_0 \wedge \neg x_1 \wedge x_0' \wedge \neg x_1' \wedge e_4^l \wedge e_4^u) \\ \vee(s \wedge \neg x_0 \wedge x_1 \wedge \neg x_0' \wedge x_1' \wedge e_3^l \wedge e_3^u) \\ \vee(s \wedge \neg x_0 \wedge x_1 \wedge x_0' \wedge \neg x_1' \wedge e_3^l \wedge e_3^u) \\ \vee(r \wedge x_0 \wedge \neg x_1 \wedge \neg x_0' \wedge x_1' \wedge e_3^l \wedge e_3^u) \\ \vee(r \wedge x_0 \wedge \neg x_1 \wedge x_0' \wedge \neg x_1' \wedge e_3^l \wedge e_3^u) \\ \vee(f \wedge x_0 \wedge x_1 \wedge x_0' \wedge x_1' \wedge e_4^l \wedge e_4^u)) \\ \wedge(e_1^l \vee e_1^u) \wedge (e_2^l \vee e_2^u) \wedge (e_3^l \vee e_3^u) \wedge (e_4^l \vee e_4^u) \\ \wedge\neg(e_3^l \wedge e_2^u)\ \wedge\neg(e_4^l \wedge e_2^u)\ \wedge\neg(e_4^l \wedge e_3^u) \end{array}$$

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

Definition 7: Symbolic IMDP (SIMDP) is a tuple $S_I = (X,\ Init_I,\ Y, f_{S_I}^l\ (yxx'z),\ f_{S_I}^u\ (yxx'z), Z)$ where X, Y and Z, $Init_I$ are defined as for IMDP. $f_{S_I}^l(yxx'z)$ and $f_{S_I}^u(yxx'z)$ 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, $f_{S_I}^l(yxx'z)$ and $f_{S_I}^u(yxx'z)$ are MTBDD encoding the IMDP, respectively, with lower and upper probability values.

B. CDNF Learning Algorithm

The CDNF learning algorithm [4] is an exact learning algorithm for Boolean functions. It learns a Boolean formula in conjunctive disjunctive normal form (CDNF) for a target Boolean function over a fixed set of Boolean variables x. In this paper, we use this algorithm to learn the symbolic assumptions I for MDP represented by Boolean functions.

During the learning process, the CDNF learning algorithm interacts with a Teacher (Model checker) to make two types of queries: (i) *membership queries* and (ii) *equivalence queries*. A membership queries are used to check whether a valuation v over Boolean variables x satisfies the target function. Equivalence queries are used to check whether a conjectured Boolean function is equivalent to the target function.

In our approach, we generate an IMDP for M_0. This IMDP have the same states, actions and transactions. The only difference is the probability value, which is set to the interval $[0, 1]$. This IMDP will be used for the CDNF as target.

δ_{M_0}	$f_{S_M}\ (yxx'e^le^u)$
$s_0 \xrightarrow{open,1/3} s_0$	$o \wedge \neg x_0 \wedge \neg x_1 \wedge \neg x_0' \wedge \neg x_1' \wedge e_2^l \wedge e_2^u$
$s_0 \xrightarrow{open,1/3} s_1$	$o \wedge \neg x_0 \wedge \neg x_1 \wedge \neg x_0' \wedge x_1' \wedge e_2^l \wedge e_2^u$
$s_0 \xrightarrow{open,1/3} s_3$	$o \wedge \neg x_0 \wedge \neg x_1 \wedge x_0' \wedge x_1' \wedge e_2^l \wedge e_2^u$
$s_0 \xrightarrow{restart,1} s_2$	$r \wedge \neg x_0 \wedge \neg x_1 \wedge x_0' \wedge \neg x_1' \wedge e_4^l \wedge e_4^u$
$s_1 \xrightarrow{send,1/2} s_1$	$s \wedge \neg x_0 \wedge x_1 \wedge \neg x_0' \wedge x_1' \wedge e_3^l \wedge e_3^u$
$s_1 \xrightarrow{send,1/2} s_2$	$s \wedge \neg x_0 \wedge x_1 \wedge x_0' \wedge \neg x_1' \wedge e_3^l \wedge e_3^u$
$s_2 \xrightarrow{restart,1/2} s_1$	$r \wedge x_0 \wedge \neg x_1 \wedge \neg x_0' \wedge x_1' \wedge e_3^l \wedge e_3^u$
$s_2 \xrightarrow{restart,1/2} s_2$	$r \wedge x_0 \wedge \neg x_1 \wedge x_0' \wedge \neg x_1' \wedge e_3^l \wedge e_3^u$
$s_3 \xrightarrow{fail,1} s_3$	$f \wedge x_0 \wedge x_1 \wedge x_0' \wedge x_1' \wedge e_4^l \wedge e_4^u$

$e_i \in E$	Predicate	$s_i \in \Sigma_{M_0}$	$enc(s_i)$
e_1^l	≥ 0	s_0	$\neg x_0 \wedge \neg x_1$
e_1^u	≤ 0	s_1	$\neg x_0 \wedge x_1$
e_2^l	$\geq 1/3$	s_2	$x_0 \wedge \neg x_1$
e_2^u	$\leq 1/3$	s_3	$x_0 \wedge x_1$
e_3^l	$\geq 1/2$		
e_3^u	$\leq 1/2$		
e_4^l	≥ 1		
e_4^u	≤ 1		

TABLE I: Encoding the set of states and the transition function of MDP M_0 (Fig. 5).

C. Symbolic Assume-Guarantee Reasoning Rule

To establish the compositional verification process we propose a symbolic assume-guarantee reasoning proof rule, where assumptions are represented using IMDP. As described before, the compositional verification aims to generate a symbolic assumption I, where M_0 is embedded in I ($M_0 \preceq I$).

Definition 8: 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 IMDP, 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 3: Consider the MDP M_0 shown in Fig. 5 and IMDP I shown in Fig. 2. They have the same state space, identical initial state (s_0, i_0) and the same set of actions $\{open,$ $send,\ restart,\ fail\}$. 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 $s_0 \xrightarrow{open,1/3} s_1$, which falls into the interval $[0, 1]$ labelled the transition $i_0 \xrightarrow{open,[0,1]} i_1$ in I, formally $P^l(i_0, open)(i_1) \leq \mu(s_0, open)(s_1) \leq P^u(i_0, open)(i_0')$. Thus, we have $M_0 \preceq I$ (M_0 is embedded in I).

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

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

This proof rule means, if we have a system composed of two components M_0 and M_1, then we can check the

correctness of a probabilistic safety property $P_{\leq P}[\psi]$ over $M_0 \parallel M_1$ without constructing and verifying the full state space. Instead, we first generate an appropriate assumption I, where I is an IMDP, then we check if this assumption could be used to verify $M_0 \parallel M_1$ 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 $P_{\leq P}[\psi]$, $I \parallel M_1 \models P_{\leq P}[\psi]$.
If the two promises are satisfied then we can conclude that $M_0 \parallel M_1$ satisfies $P_{\leq P}[\psi]$.

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 IMDP I, $I = (S_I, s_0^I, \Sigma_I, P^l, P^u)$. If $M_0 \preceq I$ and based on Def. 8 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 Def. 3 and 8, $M_0 \parallel M_1$ 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: $\mu_{M_0\parallel M_1}((s_i,s_j),a)(s'_i,s'_j) = \mu_{M_0}((s_i),a)(s'_i) \times \mu_{M_1}((s_j),a)(s'_j)$ for any state $s_i, s'_i \in S_{M_0}$ and $s_j, s'_j \in S_{M_1}$. Thus, $P^l((s_i,s_j),a)(s'_i,s'_j) \leq \mu_{M_0\parallel M_1}((s_i,s_j),a)\ (s'_i,s'_j) \leq P^u((s_i,s_j),a)(s'_i,s'_j)$ 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 $I \parallel M_1 \models P_{\leq P}[\psi]$ then $M_0 \parallel M_1 \models P_{\leq P}[\psi]$ is guaranteed. ■

D. ACVuIL: Automatic Compositional Verification using Implicit Learning algorithm

Algorithm ACVuIL highlighted the main steps of our approach. ACVuIL starts by encoding M_0 as a Boolean functions, then it calls the CDNF learning algorithm to generate the first conjecture assumption. Instead of using M_0 as target, we generate an IMDP, which has the same states, actions and transactions as M_0 except the probability value, which is set to $[0,1]$. Indeed, we use IMDP instead of MDP as target in the aim of generating more compact and small assumptions. For the first assumption, CDNF generates $I_0 = true$. We handle the first conjecture assumption separately, since the $true$ value means that all the interval value between any two states is equal to $[0,1]$. Therefore, we are sure that M_0 is embedded in I_0 and ACVuIL checks directly the second promise of the symbolic assume-guarantee. This is done by using the symbolic model checking algorithm. Indeed, in practice we have used PRISM as Model checker [1]. Due to the absence of SAT-based model checking for MDP, we convert the Boolean functions encoded I_0 to MTBDD. The symbolic model cheeking of $I_0 \parallel M_1$ against the probabilistic safety property P returns $true$ if and only if $I_0 \parallel M_1$ satisfies P, in this case the ACVuIL returns $true$ which means that the system $M_0 \parallel M_1$ satisfies P. Otherwise, if ACVuIL returns $false$ then ACVuIL checks if really $M_0 \parallel M_1$ does not satisfy P or this is due to the conjecture assumption I_0. For that, we generate a counterexample Ctx illustrating why $I_0 \parallel M_1$ does not satisfy P. In practice we have used Dipro to generate counterexamples [2]. In line 11, the ACVuIL analyses the Ctx if is a real counterexample or not, this is done by generating a sub-MDP $SubM_0$ of M_0 containing only states and transitions exist in Ctx. Then it applies the symbolic model cheeking to check if $SubM_0 \parallel M_1$ satisfies P. If this returns $true$, this means that Ctx is real counterexample and the model $M_0 \parallel M_1$ does not satisfy P, theretofore, ACVuIL returns I, Ctx and $false$. Otherwise, if $SubM_0 \parallel M_1$ returns $true$, this means that the conjecture assumption can not be used to establish the compositional verification and ACVuIL should return the $SubM_0$ to CDNF to refine I. In line 17-21, the ACVuIL checks the two promises of the assume-guarantee reasoning rule for the refined assumption I_{i-1}.

Algorithm 1 ACVuIL

```
1: Input: $S_{M_0}, S_{M_1}$ and $\varphi = P_{\leq P}[\psi]$
2: output: $SIMDP\ I$, set of counterexamples and a Boolean value
3: Begin
4: SMDP $SM_0 \leftarrow$ Encode $M_0$ as a Boolean functions;
5: SIMDP $SIM_0 \leftarrow$ generate and encode IMDP for $M_0$;
6: $I_0$ = CDNF ($SIM_0$);
7: result $\leftarrow$ Symbolic model checking ($I_0$,$M_1$,$\varphi$);
8: i $\leftarrow$ 1;
9: while result == false do
10:    $Ctx \leftarrow$ Generate counterexample ($I_{i-1}$,$M_1$,$\varphi$);
11:    $subM_0 \leftarrow$ Generate sub-MDP ($M_0$,$Ctx$);
12:    real $\leftarrow$ Analyse counterexample ($I_{i-1}$,$M_1$,$\varphi$);
13:    if real == true then
14:       return ($I_{i-1}$,$Ctx$,false);
15:    else
16:       BCtx = Encode $subM_0$ using Boolean functions;
17:       $I_i \leftarrow$ return BCtx to CNDF to refine $I_{i-1}$
18:       isEmbedded $\leftarrow$ Check if $M_0 \preceq I$;
19:       while isEmbedded == false do
20:          return divergence to CDNF to refine $I_{i-1}$;
21:       end while
22:       result $\leftarrow$ Symbolic model checking ($I_i$,$M_1$,$\varphi$);
23:       $i \leftarrow i + 1$;
24:    end if
25: end while;
26: return ($I_{i-1}$,NULL,true);
27: End
```

E. Example

In this section, we present our approach using the following example: we consider the verification of $M_0 \parallel M_1$ (Fig. 5) against the probabilistic safety property $P_{\leq 0.375}[\Diamond fail]$, where "fail" stands for the state (s_3, t_3). This property means that the maximum probability that the system $M_0 \parallel M_1$ should never fails, over all possible adversaries, is less than 0.375.

ACVuIL starts by encoding M_0 using Boolean functions (Sec. IV-A). Then it calls the CDNF to generate the first conjecture assumption I_0. To analyse if I_0 could be used to establish the compositional verification, the ACVuIL calls the symbolic model cheeking to check if $I_0 \parallel M_1 \models P_{\leq 0.375}[\Diamond fail]$. This latter returns $false$. The ACVuIL calls Dipro to generate counterexample Ctx. Dipro returns the following set of paths as counterexample: $\{s_0 i_0 \xrightarrow{open,1/3} s_3 i_0, s_0 i_0 \xrightarrow{open,1/3} s_0 i_0 \xrightarrow{open,1/3} s_3 i_0\}$.

To analyse this counterexample, ACVuIL generates a sub-MDP containing only states and transitions exist in Ctx. This sub-MDP $subM_0$ is illustrated in Fig.

[1]http://www.prismmodelchecker.org
[2]https://se.uni-konstanz.de/research1/tools/dipro/

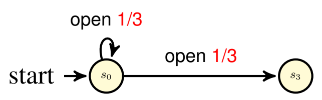

Fig. 5: sub-MDP $SubM_0$ generated to analyse Ctx.

The ACVuIL calls the symbolic model checker to analyse if $SubM_0 \parallel M_1$ satisfies $P_{\leq 0.375}[\Diamond fail]$. This latter returns $true$. Therefore, the Ctx is not a real counterexample for $M_0 \parallel M_1$. The ACVuIL encodes the $SubM_0$ using the Boolean functions, based on the same process of encoding an MDP, and return it CDNF to refine I_0. After a few iterations, following the same process as before, ACVuIL returns $true$, therefore, we can conclude that the system $M_0 \parallel M_1$ satisfies the safety property $P_{\leq 0.375}[\Diamond fail]$.

V. Conclusion and Future Works

In this paper, we proposed a fully-automated probabilistic symbolic compositional verification to verify probabilistic systems, where each component is an MDP. Our approach is based on our proposed symbolic assume-guarantee reasoning rule, which describes the compositional verification process. To generate assumptions, we proposed to use the CDNF learning algorithm which accept a Boolean function as input. For that, we converted the MDP and IMDP to Boolean functions. For the future works, we plan to proposed other assume-guarantee reasoning rule such as asymmetric rule or circular rule to handle more large and complex systems. in addition, we plan to evaluate our approach using large systems. A good example, real life example, is the verification of inter-organisational workflows [5].

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] Baier C, Katoen JP, Larsen KG (2008) Principles of model checking. MIT press.

[3] Baier C, Kwiatkowska M (1998) Model checking for a probabilistic branching time logic with fairness. Distributed Computing, 11(3), 125-155.

[4] Bshouty N.H (1995). Exact learning boolean functions via the monotone theory. Information and Computation, 123(1), 146-153.

[5] Bouchekir R, Boukhedouma S, Boukala M (2016). Automatic Compositional Verification of Probabilistic Safety Properties for Inter-organisationalWorkflow Processes.In Proceedings of the 6th International Conference on Simulation and Modeling Methodologies, Technologies and Applications - Volume 1: SIMULTECH, ISBN 978-989-758-199-1, pages 244-253.

[6] Burch JR, Clarke EM, McMillan KL, Dill DL, Hwang LJ (1992) Symbolic model checking: 10^{20} states and beyond. Information and computation, 98(2), 142-170.

[7] 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 (pp. 511-526). Springer Berlin Heidelberg.

[8] 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 (pp. 60-76). Springer Berlin Heidelberg.

[9] 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 (pp. 331-346). Springer Berlin Heidelberg.

[10] Debbi H, Debbi A, Bourahla M (2016) Debugging of probabilistic systems using structural equation modelling. International Journal of Critical Computer-Based Systems, 6(4), 250-274.

[11] Fujita M, McGeer PC, Yang JY (1997) Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. Formal methods in system design, 10(2-3), 149-169.

[12] Feng L, Kwiatkowska M, Parker D (2010) Compositional verification of probabilistic systems using learning. In Quantitative Evaluation of Systems (QEST), 2010 Seventh International Conference on the (pp. 133-142). IEEE.

[13] Feng L (2013) On learning assumptions for compositional verification of probabilistic systems (Doctoral dissertation, University of Oxford).

[14] He F, Gao X, Wang M, Wang BY, Zhang L (2016) Learning Weighted Assumptions for Compositional Verification of Markov Decision Processes. ACM Transactions on Software Engineering and Methodology (TOSEM), 25(3), 21.

[15] Hasson H, Jonsson B (1994) A logic for reasoning about time and probability. Formal Aspects of Computing, 6, 512-535.

[16] Hart S (1984) Probabilistic temporal logics for finite and bounded models. In Proceedings of the sixteenth annual ACM symposium on Theory of computing (pp. 1-13). ACM.

[17] Jansen N, Wimmer R, Abraham E, Zajzon B, Katoen JP, Becker B, Schuster J (2014) Symbolic counterexample generation for large discrete-time Markov chains. Science of Computer Programming, 91, 90-114.

[18] Kwiatkowska M, Norman G, Parker D (2011) PRISM 4.0: Verification of probabilistic real-time systems. In International Conference on Computer Aided Verification (pp. 585-591). Springer Berlin Heidelberg.

[19] 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 (pp. 23-37). Springer Berlin Heidelberg.

[20] Lehmann D, Shelah S (1982) Reasoning with time and chance. Information and Control, 53(3), 165-198.

[21] Larsen KG, Pettersson P, Yi W (1995) Compositional and symbolic model-checking of real-time systems. In Real-Time Systems Symposium, 1995. Proceedings., 16th IEEE (pp. 76-87). IEEE.

[22] McMillan KL (1993) Symbolic model checking. In Symbolic Model Checking (pp. 25-60). Springer US.

[23] Segala R (1995) Modelling and Verification of Randomized Distributed Real Time Systems. PhD thesis, Massachusetts Institute of Technology.

[24] Parker DA (2002) Implementation of symbolic model checking for probabilistic systems (Doctoral dissertation, University of Birmingham).

[25] 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. Formal Methods in System Design, 32(3), 175-205.

[26] Pnueli A, Zuck L (1986) Verification of multiprocess probabilistic protocols. Distributed Computing, 1(1), 53-72.

[27] Vardi MY (1985) Automatic verification of probabilistic concurrent finite state programs. In Foundations of Computer Science, 1985., 26th Annual Symposium on (pp. 327-338). IEEE.

[28] Vardi, MY, Wolper, P (1994). Reasoning about infinite computations. Information and computation, 115(1), 1-37.

[29] 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 (pp. 265-276). Springer Berlin Heidelberg.