Toward Implicit Learning for the Compositional Verification of Markov Decision Processes

R. Bouchekir and M.C. Boukala
LSI, Computer Science department
USTHB
BP 32 El-Alia Algiers, ALGERIA
{rbouchekir,mboukala}@usthb.dz

In this paper, we propose an automated compositional verification using implicit learning to verify Markov Decision Process (MDP) against probabilistic safety properties. Our approach, denoted *ACVuIL*, starts by encoding implicitly the MDP components by using compact data structures. Then, we use a sound and complete symbolic assume-guarantee reasoning rule to establish the compositional verification process. This rule uses the CDNF learning algorithm to generate automatically the symbolic probabilistic assumptions. Experimental results suggest promising outlooks for our approach.

keywords: Probabilistic model checking · Compositional verification · Symbolic model checking· Assume-guarantee paradigm · Machine learning · CDNF Learning algorithm.

1 Introduction

1.1 Context and Motivation

An important feature of modern systems is their complexity. This characteristic makes the design, implementation and verification of complex systems extremely difficult. This difficulty is enhanced by the often critical role of these systems (avionics control process, nuclear power plants, etc.). *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 [25],[18]. This is then checked against properties specified in probabilistic extensions of temporal logic, such as Probabilistic Computation Tree Logic (PCTL) [17].

Formal methods, including the Probabilistic Model Checking [1],[2],[3] suffer from the problem of state space explosion. This problem constitutes, even after several years of research, the main obstacle of probabilistic model checking. *Compositional verification* [26],[15],[16],[19] and *Symbolic model checking* [7],[27] are two promising approaches to cope with this problem. 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],[30],[9], 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 assumptions can be solved by using machine learning [9, 16], such as *CDNF* learning algorithm [5]. 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 verify 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) [14], 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 functions and Multi Terminal BDD. This encoding allows to store and explore a large number of states efficiently [9]. We use the Boolean functions as input of the CDNF learning algorithm. This algorithm generates an assumption which simulates a set of MDP component. The idea is to use this assumption for the verification instead of the real system components. Thus, if the size of this assumption is much smaller than the size of the corresponding MDP component, then we can expect significant gain of the verification performance. In our work, Interval Markov Decision Processes (IMDP) are used to represent assumptions. 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. This rule defines and establish the compositional verification process. We have illustrated our approach using a simple example, and we have applied our approach in a several case studies derived from PRISM benchmarks. Experimental results suggest promising outlooks for the implicit learning of the compositional verification.

The remainder of this paper is organized as follows: In section 2 we provide the most relevant works to our work. Section 3 provides some background knowledge about MDP, Interval MDP and the parallel composition MDP || IMDP. In Section 4, 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 6 concludes the paper and talks about future works.

2 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 [32], [33],[34], and then by Pnueli and Zuck [31], and by Baier and Kwiatkowska [3]. The symbolic probabilistic model checking algorithms have been proposed by [29], [10]. These algorithms have been implemented in a symbolic probabilistic model checker PRISM [22]. The main techniques used to generate counterexamples was detailed in [21]. A recent work [13] 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 [11], [30], [9]. The compositional verification of probabilistic systems has been a significant progress in these last years [23],[15],[16], [19]. Our approach is inspired by the work of [15],[16]. 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 [19]. 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.

3 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 $M = (States_M, s_0^M, \Sigma_M, \delta_M)$ where $States_M$ is a finite set of states, $s_0^M \in States_M$ is an initial state, Σ_M is a finite set of actions, $\delta_M \subseteq States_M \times (\Sigma_M \cup \{\tau\}) \times Dist(States_M)$ is a probabilistic transition relation.*

In a state s of MDP M, one or more transitions, denoted $(s,a) \to \mu$, are available, where $a \in \Sigma_M$ is an action label, μ is a probability distribution over states, where $\mu \neq 0$, and $(s,a,\mu) \in \delta_M$. A path through MDP is a (finite or infinite) sequence $(s_0,a_0,\mu_0) \to (s_1,a_1,\mu_1) \to \ldots$. An example of two MDP M_0 and M_1 is shown in Fig. 1.

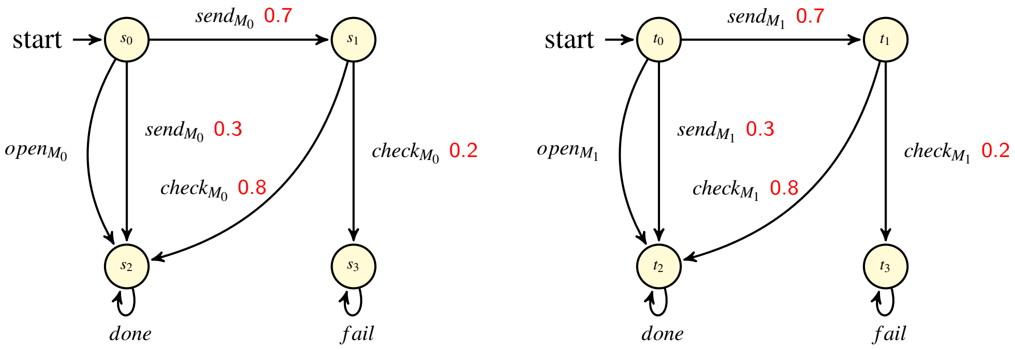

Figure 1: Example of two MDP, M_0 (left) and M_1 (right).

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 $I = (States_I, i_0^I, \Sigma_I, P^l, P^u)$ where $States_I, i_0^I$ and Σ_I are respectively the set of states, initial state and the set of actions. $P^l, P^u : States_I \times \Sigma_I \times States_I \mapsto [0,1]$ are matrices representing the lower/upper bounds of transition probabilities such that: $P^l(i,i') \leq P^u(i,i')$ for all states $i, i' \in States_I$.*

An example of IMDP I is shown in Fig. 2, where *all* represents the set of actions : $\{open_{M_0}, send_{M_0}, check_{M_0}, done, fail\}$ with the probability interval value equal to $[0,1]$.

In Definition 3, we describe how MDP and IMDP are composed together. This is done by using the asynchronous parallel operator ($\|$) defined by [28], where MDP and IMDP synchronise 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. $MI = \{States_M \times States_I, (s_0^M, s_0^I), \Sigma_M \cup \Sigma_I, P^l, P^u\}$, 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, $s'_i \in States_M$ and s_j, $s'_j \in States_I$: (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 $a \in \Sigma_M \cap \Sigma_I$, (ii) $s_i \xrightarrow{a,\mu_i} s'_i$, where $a \in \Sigma_M \setminus \Sigma_I$, and (iii) $s_j \xrightarrow{P^l(s_j,a)(s'_j),P^u(s_j,a)(s'_j)} s'_j$, where $a \in \Sigma_M \setminus \Sigma_I$.

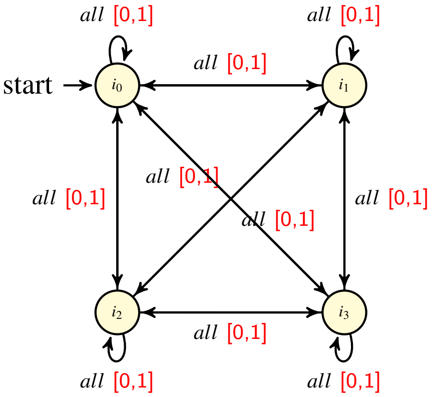

Figure 2: Example of IMDP I.

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 $var(d) \in X$, where X is a finite ordered set of Boolean variables. 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.*

4 ACVuIL approach

Our approach, probabilistic symbolic compositional verification using implicit learning (ACVuIL), aims to mitigate the state explosion problem. Fig. 3 illustrates an overview of ACVuIL. The first step consists to encode the system component M_0 using Boolean functions $\beta(M_0)$. Different from the explicit representing of the state space, the implicit representation using Boolean functions allows to store and explore a large number of states efficiently. $\beta(M_0)$ will be used as input of the CDNF learning algorithm as target language. The second step aims to generate an appropriate assumption S_{I_i}, which needs to abstract the behaviour of the original competent M_0. In our approach, we use the *CDNF* learning algorithm to generate automatically the assumptions. The second step starts by calling the *CDNF* learning algorithm, with $\beta(M_0)$ as input. At each iteration, the CDNF learns a new assumption $\beta(I_i)$ represented as Boolean functions. For the first iteration ($i = 0$), the CDNF generates *true* as assumption, for that, we generate a special initial assumption S_{I_0}. For ($i \geq 1$) iterations, we convert the generated assumption $\beta(I_i)$ to MTBDD S_{I_i}, then we refine the initial assumption S_{I_0} using S_{I_i}. We use the symbolic probabilistic model

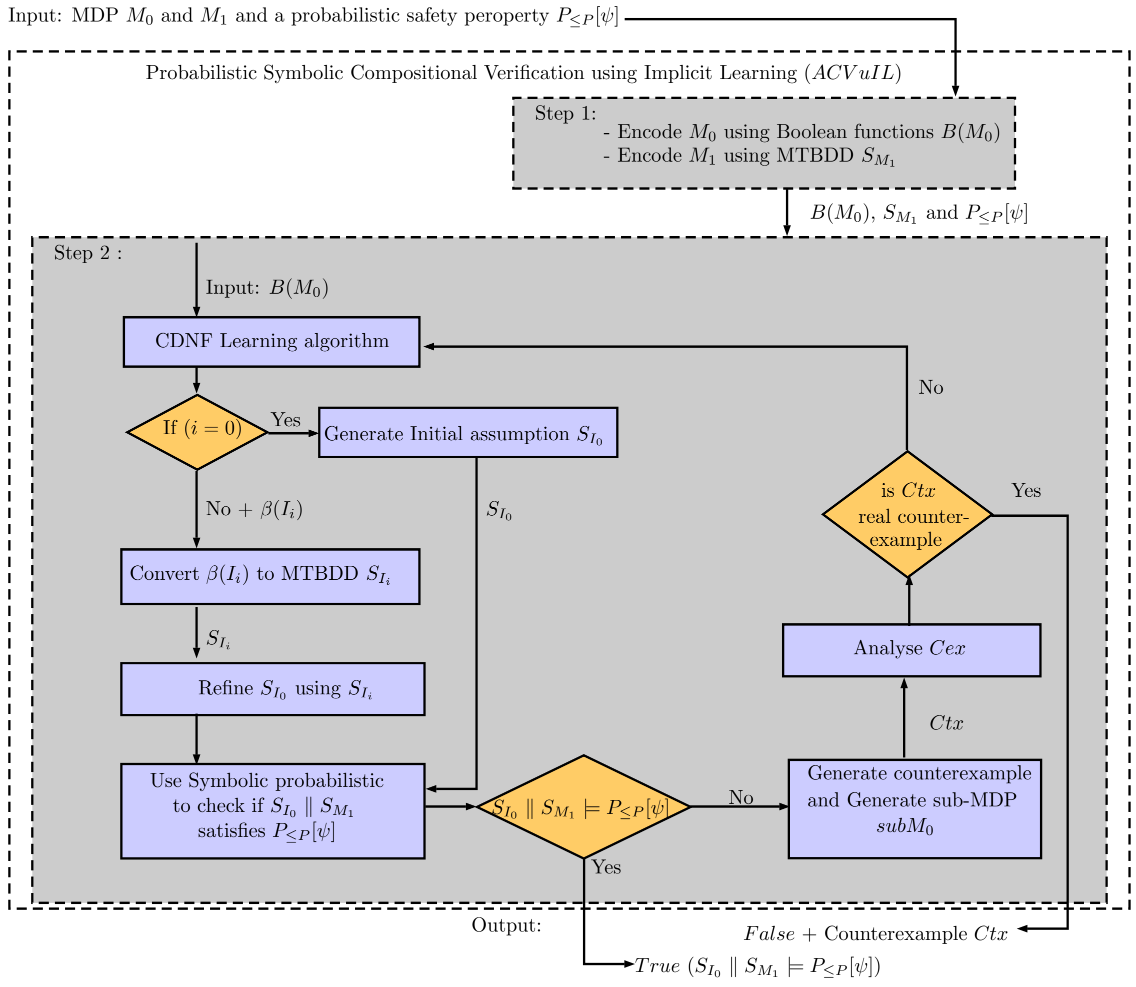
Input: MDP M_0 and M_1 and a probabilistic safety peroperty $P_{\leq P}[\psi]$
Probabilistic Symbolic Compositional Verification using Implicit Learning ($ACVuIL$)
Step 1:
- Encode M_0 using Boolean functions $B(M_0)$
- Encode M_1 using MTBDD S_{M_1}
$B(M_0)$, S_{M_1} and $P_{\leq P}[\psi]$
Step 2 :
Input: $B(M_0)$
CDNF Learning algorithm
If $(i = 0)$
Yes
Generate Initial assumption S_{I_0}
No + $\beta(I_i)$
S_{I_0}
Convert $\beta(I_i)$ to MTBDD S_{I_i}
S_{I_i}
Refine S_{I_0} using S_{I_i}
Use Symbolic probabilistic to check if $S_{I_0} \parallel S_{M_1}$ satisfies $P_{\leq P}[\psi]$
$S_{I_0} \parallel S_{M_1} \models P_{\leq P}[\psi]$
No
Yes
Generate counterexample and Generate sub-MDP $subM_0$
Ctx
Analyse Cex
is Ctx real counter-example
No
Yes
Output:
$False$ + Counterexample Ctx
$True$ ($S_{I_0} \parallel S_{M_1} \models P_{\leq P}[\psi]$)

Figure 3: An overview of our approach (ACVuIL).

checking algorithm (*SPMC*) to verify if $S_{I_0} \parallel S_{M_1}$ satisfies the probabilistic safety property $P_{\leq P}[\psi]$. If $SPMC(S_{I_0}, S_{M_1})$ returns true, then we can conclude that $M_0 \parallel M_1 \models P_{\leq P}[\psi]$ is true i.e. $P_{\leq P}[\psi]$ satisfies $M_0 \parallel M_1$, otherwise, we generate a counterexample *Ctx* illustrated why $P_{\leq P}[\psi]$ is violated. *Ctx* can be a real counterexample of the system $M_0 \parallel M_1$ or a spurious counterexample due to the generated assumption. Thus, at each iteration, we analyse if *Ctx* is real or not. If *Ctx* is real, then we can conclude that $M_0 \parallel M_1 \not\models P_{\leq P}[\psi]$ i.e. $P_{\leq P}[\psi]$ does not satisfy the system $M_0 \parallel M_1$, otherwise, we return *Ctx* to CDNF to generate a new assumption. Our compositional verification process is sound and complete. The soundness and completeness is guaranteed by the use of an assume-guarantee reasoning rule. All steps of our approach are described in details in the next sections.

4.1 Encoding MDP using Boolean functions

MDP can be encoded implicitly as Boolean functions, we denote by $\beta(M_0)$ the Boolean functions encoded MDP M_0. The encoding process of MDP using Boolean functions aims to reduce the size of the state space. Indeed, many works such as [7], [14], [29], [27] show that the implicit representation is often more efficient than the explicit representation. In addition, this Boolean functions will be used as input of the CDNF learning algorithm. In this section we describe the process of encoding MDP using Boolean functions.

Definition 6 *$\beta(M_0) = (Init_{M_0}, f_{M_0}\ (yxx'e^le^u))$ is a pair of Boolean functions encoded the MDP M_0, where $Init_{M_0}$ is predicate encoding the initial state $s_0^{M_0}$ over the set X and $f_M(yxx'e^le^u)$ 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. Y, X, X' and E are finite ordered set of Boolean variables with $Y \cap X \cap X' \cap E = \emptyset$. The set X encodes the states of M_0, X' next states, Y encodes actions and E encodes the probability values.*

More concretely, let $M_0 = (States_{M_0}, s_0^M, \Sigma_{M_0}, \delta_{M_0})$ be MDP, $n = |States_{M_0}|$, $m = |\Sigma_{M_0}|$ and $k = \lceil log_2(n) \rceil$. We can see δ_{M_0} as a function of the form $States_{M_0} \times \Sigma_{M_0} \times \{1, 2, ..., r\} \times States_{M_0} \to [0, 1]$, where r is the number of non-deterministic choice of a transition. We use a function $enc : States_{M_0} \to \{0, 1\}^k$ over $X = \langle x_1,\ x_2,\ ..., x_k \rangle$ to encode states in $States_{M_0}$ 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 actions 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_0}. $f_{M_0}\ (yxx'e^le^u)$ encodes the probabilistic transition relation δ_{M_0} 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 and 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 1 *To illustrate how we encode MDP as Boolean functions, we consider the MDP M_0 (Fig. 1). M_0 contains the set of states $States_{M_0} = \{s_0,\ s_1,\ s_2,\ s_3\}$ and the set of actions $\Sigma_{M_0} = \{open_{M_0},\ send_{M_0},\ check_{M_0},\ done,\ fail\}$. We use $X = \langle x_0, x_1 \rangle$ to encode the set of states in $States_{M_0}$ 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 $Y = \langle o, s, c, d, f \rangle$ to encode the actions $\{open_{M_0},\ send_{M_0},\ check_{M_0},\ done,\ fail\}$, respectively. Table I summarizes the process of encoding the transition function δ_{M_0}. $\beta(M_0) = (Init_{M_0}, f_{M_0}\ (yxx'e^le^u))$ encoded M_0 is $Init_{M_0} = \neg x_0 \wedge \neg x_1$ and*

$$f_{M_0}\ (yxx'e^le^u) = \begin{array}{l} ((s \wedge \neg x_0 \wedge \neg x_1 \wedge \neg x'_0 \wedge x'_1 \wedge e_3^l \wedge e_3^u) \vee (s \wedge \neg x_0 \wedge \neg x_1 \wedge x'_0 \wedge \neg x'_1 \wedge e_2^l \wedge e_2^u) \\ \vee (o \wedge \neg x_0 \wedge \neg x_1 \wedge x'_0 \wedge \neg x'_1 \wedge e_5^l \wedge e_5^u) \vee (c \wedge \neg x_0 \wedge x_1 \wedge x'_0 \wedge \neg x'_1 \wedge e_4^l \wedge e_4^u) \\ \vee (c \wedge \neg x_0 \wedge x_1 \wedge x'_0 \wedge x'_1 \wedge e_1^l \wedge e_1^u) \vee (d \wedge x_0 \wedge \neg x_1 \wedge x'_0 \wedge \neg x'_1 \wedge e_5^l \wedge e_5^u) \\ \vee (f \wedge x_0 \wedge x_1 \wedge x'_0 \wedge x'_1 \wedge e_5^l \wedge e_5^u)) \end{array}$$

$e_i \in E$	Predicate	$e_i \in E$	Predicate	$s_i \in \Sigma_{M_0}$	$enc(s_i)$
e_0^l	≥ 0	e_2^l	≥ 0.3	s_0	$\neg x_0 \wedge \neg x_1$
e_0^u	≤ 0	e_2^u	≤ 0.3	s_1	$\neg x_0 \wedge x_1$
e_1^l	≥ 0.2	e_3^l	≥ 0.7	s_2	$x_0 \wedge \neg x_1$
e_1^u	≤ 0.2	e_3^u	≤ 0.7	s_3	$x_0 \wedge x_1$
e_4^l	≥ 0.8	e_5^l	≥ 1		
e_4^u	≤ 0.8	e_5^u	≤ 1		

Table 1: Encoding the set of states and the probability values of MDP M_0 (Fig. 1).

4.2 Encoding MDP using MTBDD

In this paper, we also consider the implicit representation using MTBDD. MTBDD are used to encode components M_1 and to perform the probabilistic model checking. In Def. 7, we introduce Symbolic MDP (SMDP) and we provide the different data structures used to encode MDP. We denoted by S_{M_1} the SMDP encoded the MDP M_1.

Definition 7 *Symbolic MDP (SMDP) is a tuple $S_M = (X,\ Init_M,\ Y, f_{S_M}\ (yxx'))$ where X, X' and Y are finite ordered set of Boolean variables with $X \cap X' \cap Y = \emptyset$. $Init(X)$ is a BDD encoded the initial state and $f_{S_M}(yxx')$ is an MTBDD encoded the transition relation. The sets X, X' and Y are used to encode respectively the set of states, next states and the set of actions of M, and y,x,x' are valuations of receptively, Y,X,X'.*

The encoding of MDP as SMDP follows the same process as the encoding using Boolean functions.

Example 2 *We consider the MDP M_1 (Fig. 1) to illustrate the encoding of MDP using SMDP. M_1 contains the set of states $States_{M_1} = \{t_0,t_1,t_2,t_3\}$ and the set of actions $\Sigma_{M_1} = \{open_{M_1},\ send_{M_1},\ check_{M_1},$ $done,\ fail\}$. We use the set $X = \langle x_0, x_1 \rangle$ to encode the set of states $States_{M_0}$ as: $enc(t_0) = (00)$, $enc(t_1) = (01)$, $enc(t_2) = (10)$, $enc(t_3) = (11)$; and we use the set $Y = \langle s,o,c,d,f \rangle$ to encode the actions $\{open_{M_1},$ $send_{M_1},\ check_{M_1},\ done,\ fail\}$, respectively.*

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 $S_I = (X,\ Init_I,\ Y, f^l_{S_I}\ (yxx'),\ f^u_{S_I}\ (yxx'))$ where X, X' and Y are finite ordered set of Boolean variables with $X \cap X' \cap Y = \emptyset$. $Init_I$ is a BDD encodes the initial state and $f^l_{S_I}(yxx')$ and $f^u_{S_I}(yxx')$ are MTBDD encode the transition relation over $Y \cup X \cup X'$. The MTBDD $f^l_{S_I}(yxx')$ encodes the lower probability bound and $f^u_{S_I}(yxx')$ encodes the lower. The sets X, X' and Y encode respectively, the set of states, next states and the set of actions, and y,x,x' are valuations of receptively, Y,X,X'.*

4.3 Symbolic Assume-Guarantee Reasoning Rule

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

Definition 9 *Let $M_0 = (States_{M_0}, s_0^{M_0}, \Sigma_{M_0}, \delta_{M_0})$ and $I_i = (States_{I_i}, s_0^{I_i}, \Sigma_{I_i}, P^l, P^u)$ be MDP and IMDP, respectively. We say M_0 is embedded in I_i, written $M_0 \preceq I_i$, if and only if: (1) $States_{M_0} = States_{I_i}$, (2) $s_0^{M_0} = s_0^{I_i}$, (3) $\Sigma_{M_i} = \Sigma_{I_i}$, and (4) $P^l(s,a)(s') \leq \mu(s,a)(s') \leq P^u(s,a)(s')$ for every $s,s' \in States_M$ and $a \in \Sigma_M$.*

Example 3 *Consider the MDP M_0 shown in Fig. 1 and IMDP I shown in Fig. 2. They have the same number of states, identical initial state (s_0, i_0) and the same set of actions $\Sigma_{M_1} = \{open_{M_1},\ send_{M_1},$ $check_{M_1},\ done,\ fail\}$. 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 in Σ_{M_1}. For example, the transition probability between s_0 and s_1 is $s_0 \xrightarrow{send_{M_0},0.7} s_1$, which falls into the interval $[0,1]$ labelled the transition $i_0 \xrightarrow{send_{M_0},[0,1]} i_1$ in I. 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: ***if*** *$M_0 \preceq I$* ***and*** *$I \parallel M_1 \models P_{\leq P}[\psi]$* ***then*** *$M_0 \parallel M_1 \models P_{\leq P}[\psi]$. 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: (i) Check if M_0 is embedded in I, $M_0 \preceq I$, and (ii) 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 (Soundness) *Consider M_0 and M_1 be MDP, where $M_0 = (States_{M_0},\ s_0^{M_0},\ \Sigma_{M_0},\ \delta_{M_0})$, $M_1 = (States_{M_1}, s_0^{M_1}, \Sigma_{M_1}, \delta_{M_1})$, and IMDP I, $I = (States_I, s_0^I, \Sigma_I, P^l, P^u)$. If $M_0 \preceq I$ and based on Def. 9 we have $States_M = States_I$, $s_0^M = s_0^I$, $\Sigma_M = \Sigma_I$, and $P^l(s,a)(s') \leq \mu(s,a)(s') \leq P^u(s,a)(s')$ for every $s, s' \in States_{M_0}$ and $a \in \Sigma_{M_0}$. Based on Def. 3 and 9, $M_0 \parallel M_1$ 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 $s_i, s'_i \in States_{M_0}$ and $s_j, s'_j \in States_{M_1}$. 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.*

Proof (Completeness) *The completeness of our approach is guarantee since we always generate a new assumption to refine the initial one. In the worst case, the CDNF will learn a final assumption equivalent to the original component.*

4.4 CDNF Learning Algorithm

The CDNF learning algorithm [5] 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 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.

4.5 ACVuIL: Automatic Compositional Verification using Implicit Learning algorithm

Algorithm ACVuIL highlighted the main steps of our approach. ACVuIL accepts the system components MDP M_0, M_1 and the probabilistic safety property $\varphi = P_{\leq P}[\psi]$ as input. ACVuIL starts by encoding M_0 using Boolean functions and M_1 using SMDP. Then, it calls the CDNF learning algorithm to learn the initial assumption I_0. For the first iteration, CDNF learns $true$ as initial assumption. For that, ACVuIL calls the function $GenerateInitialAssumption$ to generate S_{I_0}. The process of generating the SIMDP S_{I_0} is described in the next section.

Algorithm 1 ACVuIL

```
Input: $M_0, M_1$ and $\varphi = P_{\leq P}[\psi]$
output: $SIMDP\ I_i$, set of counterexamples and a Boolean value
Begin
$\beta(M_0) \leftarrow$ Encode $M_0$ as a Boolean functions;
$S_{M_1} \leftarrow$ Encode $M_1$ as SMDP;
$I_0 \leftarrow$ CDNF ($\beta(M_0)$);
$S_{I_0} \leftarrow$ GenerateInitialAssumption($M_0, I_0$);
$result \leftarrow$ SPMC($S_{I_0}$, $S_{M_1}$, $\varphi$);
while ($result == false$) do
    $i \leftarrow i + 1$;
    $Ctx \leftarrow$ GenerateCounterexample ($S_{I_{i-1}}$, $S_{M_1}$, $\varphi$);
    $subM_0 \leftarrow$ GenerateSub-MDP ($M_0$,$Ctx$);
    real $\leftarrow$ AnalyseCounterexample ($subM_0$,$S_{M_1}$,$\varphi$);
    if ($real == true$) then
        return ($S_{I_{i-1}}$,$Ctx$,$false$);
    else
        $\beta(I_i) \leftarrow$ return $false$ to CDNF to generate new assumption;
        $S_{I_i} \leftarrow Refine\_S_{I_i}(S_{I_0}, \beta(I_i), \beta(M_0))$.
        $result \leftarrow$ SPMC($S_{I_i}$, $S_{M_1}$, $\varphi$);
    end if
end while;
return ($S_{I_i}$,$NULL$,$true$);
End
```

4.6 Generate initial assumption

The ACVuIL calls the function $GenerateInitialAssumption$ to generate the initial assumption S_{I_0}. This function accepts MDP M_0 and the Boolean functions I_0 as inputs, and returns SIMDP S_{I_0}. The process of generating S_{I_0} is described in Algo. 2.

Algorithm 2 $GenerateInitialAssumption$

```
Input: $MDP\ M_0$, Boolean functions $I_0$
output: $SIMDP\ S_{I_0}$
BEGIN
Create a new IMDP $Initial\_I_0$ equivalent to $M_0$, with transitions equal to $[0,1]$ between all states.
  The set of actions in $M_0$ are hold in each transition of $I_0$.
$S_{I_0} \leftarrow$ Encode $Initial\_I_0$ as SIMDP;
return $S_{I_0}$;
End
```

$GenerateInitialAssumption$ creates a new IMDP $Initial\_I_0$ equivalent to M_0, with transitions equal to $[0,1]$ between all states, and the set of actions are hold in each transition. Then it encodes the IMDP of $Initial\_I_0$ as SIMDP. 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 $Initial\_I_0$, in addition, this process will help to reduce the size of the implicit representation of $Initial\_I_0$ and this by combining any isomorphic sub-tree into a single tree, and eliminating any nodes whose left and right children are isomorphic.

Example 4 *To illustrate our approach, we consider the verification of $M_0 \parallel M_1$ (Fig. 1) against the probabilistic safety property $P_{\leq 0.0195}[\Diamond\text{"}err\text{"}]$, where "err" 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* 0.0195*. ACVuIL starts by encoding M_0 using Boolean functions $\beta(M_0)$. $\beta(M_0)$ encoded M_0 is illustrated in Sec 4.1. In addition, The encoding process of M_1 as SMDP is illustrated in Sec. 4.2. After encoding the system components using implicit representation, ACVuIL calls the function GenerateInitialAssumption to generate the initial assumption. The explicit representation of the initial assumption $Initial\_I_0$ is illustrated in Fig. 2.*

4.6.1 Symbolic probabilistic model checking (SPCV)

In line 8 and 19, ACVuIL calls the function Symbolic probabilistic model checking (SPCV). To model checking $S_{I_i} \parallel S_{M_1} \models P_{\leq P}[\psi]$, SPCV computes the parallel composition $S_{I_i} \parallel S_{M_1}$, where the result is SIMDP, because S_{I_i} is SIMDP. Indeed, model checking algorithm for IMDP was considered in [8],[4], where it was demonstrated that the verification of IMDP 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 IMDP needs to be avoided. Thus, we propose rather than verifying SIMDP $S_{I_i} \parallel S_{M_1}$, we verify only a restricted SMDP *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 [19] . The symbolic probabilistic model checking used in this work was proposed in [29].

Example 5 *To analyse if S_{I_0} could be used to establish the compositional verification, ACVuIL calls the symbolic model cheeking (SPCV) to check if $S_{I_0} \parallel S_{M_1} \models P_{\leq 0.0195}[\Diamond\text{"}err\text{"}]$. This latter returns $false$. In practice, to verify $S_{I_0} \parallel S_{M_1} \models P_{\leq 0.0195}[\Diamond fail]$ we used the model PRISM with the engine "MTBDD" [22].*

4.6.2 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 10 *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 Def. 10, 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 [21]. 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 [13] proposed to use causality in order to generate small counterexamples.

Example 6 *Since* $PSCV(S_{I_0} \parallel S_{M_1} \models P_{\leq 0.0195}[\Diamond\text{"}err\text{"}])$ *returns false, the ACVuIL calls the function* $GenerateCounterexample$ *to generate Ctx, which shows the reason why* $P_{\leq 0.0195}[\Diamond\text{"}err\text{"}])$ *is violated. In addition Ctx will be used to check if it is a real counterexample or not. In practice, we used the tool DiPro to generate counterexamples. This returns* $Ctx = \{(s_0,t_0) \xrightarrow{open_{M_1},1} (s_3,t_0) \xrightarrow{send_{M_2},0.7} (s_3,t_1) \xrightarrow{check_{M_2},0.2} (s_3,t_3)\}$.

4.6.3 Generate sub-MDP and analyse the probabilistic counterexamples

To analyse if the counterexample *Ctx* is *real* or not, ACVuIL generates a sub-MDP, where this latter represents a fragment of the MDP M_0 based on the probabilistic counterexample *Ctx*, where the MDP fragment $SubM_0$ contains only transitions present in *Ctx*. Thus, the fragment $SubM_0$ is obtained by removing from M_0 all states and transitions not appearing in any path of the set *Ctx*. Since we use symbolic data structures to encode the state space, we encode the MDP fragment $SubM_0$ using SMDP (following the same process to encode MDP). The function *GenerateSubMDP* is described in Algo. 3.

Algorithm 3 $GenerateSub-MDP$

```
Input: MDP $M_0$ and a set of counterexample $Ctx$
output: SMDP $subM_0$
Begin
Sub-MDP $M_0^{Ctx}$ = remove from $M_0$ all states and transitions not appearing in any path of the set $Ctx$;
SMDP $SubM_0$ = Encode $M_0^{Ctx}$ as SMDP;
return $SubM_0$;
End
```

Then ACVuIL calls the function *AnalyseCtx* aims to check whether the probabilistic counterexample *Ctx* is *real* or not. *Ctx* is a *real* counterexample of the system $M_0 \parallel M_1 \models P_{\leq P}[\psi]$ if and only if $SubM_0 \parallel S_{M_1} \models P_{\leq \rho}[\psi]$ does not hold i.e. *AnalyseCtx* returns $true$ if and only if the symbolic probabilistic model cheeking of $SubM_0 \parallel S_{M_1} \models P_{\leq \rho}[\psi]$ returns $false$, or $false$ otherwise. The function *AnalyseCtx* is described in Algo. 4.

Algorithm 4 $AnalyseCounterexample$

```
Input: SMDP $SubM_0$, $S_{M_1}$ and $\varphi = P_{\leq \rho}[\psi]$
output: Boolean value
Begin
Boolean sat = $SPMC(SubM_0, S_{M_1}, \varphi)$;
return ¬ sat;
End
```

Example 7 *To analyse the counterexamples, ACVuIL generates a* $sub-MDP$ *containing only states and transitions exist in Ctx. For our example, the set Ctx contains transition* $s_0 \xrightarrow{open_{M_1},1} s_3$*, where this transition is not present in* M_0*. Thus, AnalyseCtx returns false, since no sub-MDP was generated for this counterexample. After a few iterations, ACVuIL returns the final assumption* S_{I_f} *equivalent to the original component* M_0*. In this example, ACVuIL was not able to generate a final assumption more compact than the original component. Indeed, in the worst case, ACVuIL returns the original component as a final assumption.*

If the probabilistic counterexample *Ctx* is not real, then ACVuIL returns $false$ to the CDNF learning algorithm. When ACVuIL returns $false$ to CDNF, this means that the generated assumption is not equivalent to the target Boolean functions. Thus, CDNF generates a new assumption $\beta(I_i)$ $(i \geq 1)$. In line

18, the ACVuIL calls the functions $Refine\_S_{I_i}$ to refine the initial assumption. The function $Refine\_S_{I_i}$ is described in the next section (Sec. 4.6.4).

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

At each iteration of the ACVuIL, the generated assumption S_{I_i} converges to the target Boolean functions ($\beta(M_0)$). The function $Refine\_S_{I_i}$ aims to refine the initial assumption S_{I_0} using the new generated assumption. This is done by removing from the initial assumption all transitions between two states, if these states are present in the new generated assumption, and add transitions from the original component between these states.

Algorithm 5 $Refine\_S_{I_i}$

```
Input: $S_{I_0}$, $\beta(I_i)$, $\beta(M_0)$
output: $S_{I_i}$
Begin
SIMDP $S_{I_{tmp}} \leftarrow$ convert $\beta(I_i)$ to SIMDP.
We consider $f^u_{S_{I_{tmp}}}(yxx')$ the MTBDD encoding the lower probability values of $S_{I_{tmp}}$ and $f^u_{S_{I_0}}$ the MTBDD encoding the lower probability values of $S_{I_0}$;
for each valuation $v_{I_{tmp}} = (y_{I_{tmp}}, x_{I_{tmp}}, x'_{I_{tmp}}) \in f^u_{S_{I_{tmp}}}$ do
    remove from $S_{I_0}$ all valuations $v_{I_0} = (y_{I_0}, x_{I_0}, x'_{I_0})$ if $x_{I_{tmp}} = x_{I_0}$ and $x'_{I_{tmp}} = x'_{I_0}$;
    add all valuations $v_{M_0} = (y_{M_0}, x_{M_0}, x'_{M_0}, e^u_{M_0}) \in \beta(M_0)$ to $f^u_{S_{I_i}}$ if $x_{I_{tmp}} = x_{M_0}$ and $x'_{I_{tmp}} = x'_{M_0}$;
end for
optimise $f^u_{S_{I_0}}$;
return $S_{I_0}$;
End
```

5 Implementation and experimental results

We have implemented a prototype tool to evaluate our approach. 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 a counterexample otherwise. to implement our tool, we have used the library BULL [1], which impelements the CDNF learning algorithm and the tool Dipro [2] to generate counterexamples. In this section, we give the results obtained for the application of our approach in a several case studies derived from the PRISM benchmark [3]. For each case study, we check the model against a probabilistic safety property using: (i) symbolic monolithic probabilistic model checking and (ii) compositional verification (our approach). The tests were carried on a personal computer with Linux as operating system, 2.30 GHz I5 CPU and 4GB RAM.

For each case study, we compare the size of the original component M_0 and the final assumption I_f and this by considering the number of clauses ($\#Clauses$) and the number of nodes (MTBDD nodes). In addition, we compare the symbolic non-compositional verification (SMV) with our approach ACVuIL. For SMV, we report the size (number of MTBDD nodes) and the time for model construction (T4MC) for the model $S_{M_0} \parallel S_{M_1}$. For ACVuIL, we report the number of iterations for ACVuIL algorithm to learn

[1] https://sourceforge.net/projects/bull/

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

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

Table 2: Experimental results for the case studies Randomized dining philosophers, Randomized Self stabilising algorithm and Client-server

Case study	N.	P.	# Clauses		MTBDD nodes		SMV		ACVuIL			
							$S_{M_0} \parallel S_{M_1}$		#Ite.	T. Gen	$S_{I_f} \parallel S_{M_1}$	
			$\beta(M_0)$	$\beta(I_f)$	S_{M_0}	S_{I_f}	T4MC	Size		S_{I_f}	T4MC	Size
R.D. Philos	6	φ_1	3696	467	910	340	0.883	5008	237	30.57	0.110	1816
	8		48656	3486	1958	670	2.573	9215	308	67.28	0.128	3711
	10		599600	24677	3335	1062	7.353	14570	381	103.62	0.259	6164
R.S. Stab.	6	φ_2	21	12	7	7	0.001	63	12	2.14	0.001	63
	10		140	80	31	31	0.007	1023	29	7.05	0.007	1023
	14		784	448	127	127	0.016	16383	59	17.25	0.016	16383
	18		4032	2304	511	511	0.034	262143	102	39.52	0.034	262143
Client Server	5	φ_3	3348	3122	917	911	0.114	5962	12	17.02	0.091	5855
	6		12069	11052	1282	1274	0.146	7439	19	25.08	0.139	7129
	7		42282	38947	1707	1697	0.130	10684	28	41.06	0.181	8433

the final assumption S_{I_f} (#*ite*.), total time to generate S_{I_f} (T. Gen. S_{I_f}), as well as the size and T4MC to model checking $S_{I_f} \parallel S_{M_1}$.

The results are reported in Tab. 2. The case studies considered in our experimental results are:
(i) Randomized dining philosophers[24],[12], for this case study we check the property $\varphi_1 =$ *the probability that philosophers do not obtain their shared resource simultaneously is at most* 0.1, formally: $P_{\leq 0.1}[\Diamond"err"]$, where label *"err"* sands for every states satisfy: $[(s_N \geq 8)\&(s_N \leq 9)]$, and N is the component number, (ii) The second case study is Israeli and Jalfon [20] solution for the randomized Self stabilising algorithm, we check the system against property: $\varphi_2 =$ *the probability to reach a stable configuration for all algorithms is at most 0.999*, (iii) The third case study is a variant of the client-server model from [30]. 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: $\varphi_3 =$ *the probability a failure state is reached is at most 0.98.*

The overall results show that ACVuIL successfully generates assumptions for all case studies. As shown in Tab. 2, CDNF learns assumption $\beta(I_f)$ smaller than the original component $\beta(M_0)$. For the case studies R.D. Philos and Client-Server, the implicit representation of the final assumption using MTBDD is more compact than the implicit representation of the original components. However, for R.S.Stab. is the same size, this is due to the fact that ACVuIL had refined all transitions of the initial assumption, therefore, the final assumption is equal to the original component. For the verification time, the symbolic monolithic verification (non-compositional) verifies the system faster than our approach ACVuIL. Indeed, our approach takes more time to generate and refine the assumptions, as well as, the time necessary to generate counterexamples at each iteration.

6 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 ACVuIL is based on complete and sound symbolic assume-guarantee reasoning rule. The first step aims to encode the system components using compact data structures such as Boolean functions and MTBDD, then we use the compositional verification to model checking the system against the probabilistic safety property. In addition, we proposed to use the CDNF to learn automatically assumptions used in the verification process. We evaluated our approach using there case studies derived from PRISM benchmark. 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. Furthermore, we plan to evaluate our approach using real-life complex systems such as the verification of the composition of inter-organisational Workflows [6].

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] 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 (pp. 32-46). Springer Berlin Heidelberg.

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

[6] 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.

[7] 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.

[8] 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 (pp. 302-317). Springer Berlin Heidelberg.

[9] 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.

[10] 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.

[11] 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.

[12] Duflot M, Fribourg L, Picaronny C (2004) Randomized dining philosophers without fairness assumption. Distributed Computing, 17(1), 65-76.

[13] 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.

[14] 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.

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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 (pp. 119-131). ACM.

[21] 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.

[22] 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.

[23] 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.

[24] 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 (pp. 133-138). ACM.

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

[26] 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.

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

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

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

[30] 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.

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

[32] 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.

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

[34] 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.