

Probabilistic Black-Box Checking via Active MDP Learning

JUNYA SHIJUBO, MASAKI WAGA, and KOHEI SUENAGA, Kyoto University, Japan

We introduce a novel methodology for testing stochastic black-box systems, frequently encountered in embedded systems. Our approach enhances the established *black-box checking (BBC)* technique to address stochastic behavior. Traditional BBC primarily involves iteratively identifying an input that breaches the system's specifications by executing the following three phases: the *learning* phase to construct an automaton approximating the black box's behavior, the *synthesis* phase to identify a candidate counterexample from the learned automaton, and the *validation* phase to validate the obtained candidate counterexample and the learned automaton against the original black-box system. Our method, ProbBBC, refines the conventional BBC approach by (1) employing an active Markov Decision Process (MDP) learning method during the learning phase, (2) incorporating probabilistic model checking in the synthesis phase, and (3) applying statistical hypothesis testing in the validation phase. ProbBBC uniquely integrates these techniques rather than merely substituting each method in the traditional BBC; for instance, the statistical hypothesis testing and the MDP learning procedure exchange information regarding the black-box system's observation with one another. The experiment results suggest that ProbBBC outperforms an existing method, especially for systems with limited observation.

CCS Concepts: • **Theory of computation** → **Verification by model checking**; • **Software and its engineering** → *Formal software verification*; • **Computer systems organization** → *Embedded systems*.

Additional Key Words and Phrases: testing, stochastic systems, automata learning, probabilistic model checking

1 INTRODUCTION

Embedded systems (ESs), including cyber-physical systems (CPSs) and IoT systems, are often safety-critical systems, making their safety assurance crucial. One of the characteristics of such systems is that their behavior is governed by stochastic disturbances (e. g., due to communication error or uncertainty in physical environments), not only by the external inputs (e. g., from a controller or other agents). Another notable characteristic is that they are often black-box systems due to their proprietary or machine-learned components. These characteristics make testing and verification of such systems challenging.

Black-box checking (BBC) [29] is a technique to apply model checking to black-box systems via model learning. Given a black-box system \mathcal{M} under test (SUT) and a temporal logic formula φ , it tries to find an input sequence witnessing $\mathcal{M} \not\models \varphi$ by iterating the following three phases:

Learning phase learns an automaton $\hat{\mathcal{M}}$ that approximates the behavior of \mathcal{M} by using an active learning procedure such as L^* [5].

Synthesis phase tries to synthesize an execution trace σ witnessing $\hat{\mathcal{M}} \not\models \varphi$ by model checking.

Validation phase validates whether the found σ also witnesses $\mathcal{M} \not\models \varphi$. If σ does not witness $\mathcal{M} \not\models \varphi$, σ is an evidence of $\mathcal{M} \neq \hat{\mathcal{M}}$, and BBC goes back to the learning phase using σ to refine $\hat{\mathcal{M}}$. When no witness for $\hat{\mathcal{M}} \not\models \varphi$ is found in the synthesis phase, this phase tries to find an evidence of $\mathcal{M} \neq \hat{\mathcal{M}}$, typically through random testing.

There have been various extensions of BBC [13, 15, 25], including the ones focusing on CPSs [31, 35]. However, these techniques are limited to *deterministic* systems because the learned automaton is *deterministic*, e. g., a DFA or a Mealy machine.

This is the author version of the paper of the same name accepted to International Conference on Embedded Software (EMSOFT), 2023.

Authors' address: Junya Shijubo, shijubo@fos.kuis.kyoto-u.ac.jp; Masaki Waga, mwaga@fos.kuis.kyoto-u.ac.jp; Kohei Suenaga, ksuenaga@gmail.com, Kyoto University, Yoshida-honmachi, Sakyo-ku, Kyoto, 606-8501, Japan.

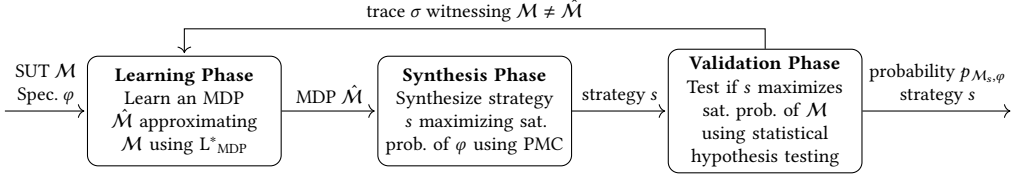


Fig. 1. Outline of probabilistic black-box checking. Given a black-box SUT \mathcal{M} and an LTL formula φ , it estimates the maximum satisfaction probability $p_{\mathcal{M},s,\varphi}$ of φ on \mathcal{M} as well as a strategy s to realize it.

Markov decision processes (MDP), an extension of Markov chains so that external inputs control the transition distribution, are widely used to formalize systems with external inputs and probabilistic branching. By fixing a *strategy* s , a function to decide the input to be fed to the system, a Markov chain $\hat{\mathcal{M}}_s$ can be constructed from an MDP $\hat{\mathcal{M}}$.

Given an MDP $\hat{\mathcal{M}}$ modeling a system and a temporal logic formula φ representing the specification, (*quantitative*) *probabilistic model checking (PMC)* [7] computes the maximum (or minimum) satisfaction probability $p_{\hat{\mathcal{M}},s,\varphi}$ of φ on $\hat{\mathcal{M}}$ as well as a strategy s to realize it. Such probability can be used as a safety measure of the system under verification. Although PMC returns the exact probability, it is challenging to apply it to real-world ESs because they are rarely white-box, and modeling the system under verification as an MDP is a non-trivial task.

In this paper, we propose *probabilistic black-box checking (ProbBBC)*, a quantitative extension of BBC for stochastic systems. Given a black-box SUT \mathcal{M} and a *linear temporal logic (LTL)* [34] formula φ , ProbBBC returns an estimation of the maximum satisfaction probability $p_{\mathcal{M},s,\varphi}$ of φ with a strategy s to realize it. ProbBBC can be used, for example, for the following purposes.

Example 1.1 (ProbBBC for debugging). Consider an IoT system \mathcal{M} that uses the MQTT protocol [1] for communication. Due to the wireless network’s instability, software bugs, or an unstable power supply, communication may not always be stable. For debugging such an issue, it is helpful to synthesize adversarial inputs leading to a communication error with a high probability. ProbBBC can synthesize such inputs by estimating the strategy that maximizes the probability of causing a communication error. If the probability returned by ProbBBC is small after a sufficiently long time, one can deem that the system \mathcal{M} is working well by the convergence of ProbBBC.

Example 1.2 (ProbBBC for controller synthesis). Consider a robot \mathcal{M} moving on a field with various conditions, e. g., concrete, grass, and mud. Our task is to create a controller for the robot to reach a specific goal. Due to sensing noise or slipping depending on the field’s condition, the robot’s movement may be probabilistic. Furthermore, it is highly challenging to formally model all the system and environmental details. ProbBBC can synthesize a near-optimal controller by estimating the strategy that maximizes the probability of reaching the goal within certain time steps. If it fails to synthesize a controller satisfying the specification after a sufficiently long time, it suggests that the task may be highly challenging by the convergence of ProbBBC.

Fig. 1 outlines ProbBBC, which, like conventional BBC, consists of the learning, synthesis, and validation phases. In the learning phase, ProbBBC learns a deterministic MDP $\hat{\mathcal{M}}$, i. e., an MDP such that the current state is uniquely identified from a sequence of inputs and outputs, from the SUT \mathcal{M} based on an active MDP learning algorithm L^*_{MDP} [33]. L^*_{MDP} systematically feeds input sequences to the SUT \mathcal{M} , estimates the probabilistic distribution of the output observations, and identifies the discrete structure and transition probabilities of \mathcal{M} . ProbBBC uses the resulting MDP $\hat{\mathcal{M}}$ as an approximation of \mathcal{M} .

In the synthesis phase, ProbBBC computes a strategy s of the approximate MDP $\hat{\mathcal{M}}$ that maximizes the satisfaction probability $p_{\hat{\mathcal{M}},s,\varphi}$ of φ on $\hat{\mathcal{M}}$ using PMC. We note that the strategy s may not

maximize the satisfaction probability $p_{\mathcal{M}_s, \varphi}$ of φ on the SUT \mathcal{M} due to the potential deviation of the approximate MDP $\hat{\mathcal{M}}$ from the SUT \mathcal{M} .

In the validation phase, ProbBBC conducts statistical hypothesis testing to check whether the obtained s also gives $p_{\mathcal{M}_s, \varphi}$ of φ for \mathcal{M} . ProbBBC obtains execution traces by executing the SUT \mathcal{M} with s multiple times. By calculating how many traces satisfy φ , ProbBBC obtains an estimation p of $p_{\mathcal{M}_s, \varphi}$.¹ Using this estimated value p of $p_{\mathcal{M}_s, \varphi}$, ProbBBC conducts statistical hypothesis testing with the null hypothesis $p_{\mathcal{M}_s, \varphi} = p_{\hat{\mathcal{M}}_s, \varphi}$. If $p_{\mathcal{M}_s, \varphi} \neq p_{\hat{\mathcal{M}}_s, \varphi}$ is established through the hypothesis test, ProbBBC tries to construct an execution trace σ witnessing the deviation of the approximate MDP $\hat{\mathcal{M}}$ from \mathcal{M} ; if such σ is found, ProbBBC feeds it back to the learning phase to refine the approximate MDP $\hat{\mathcal{M}}$.

If ProbBBC fails to find such σ , it resorts to random sampling in an attempt to detect a deviation between the approximate MDP $\hat{\mathcal{M}}$ and the SUT \mathcal{M} . ProbBBC randomly synthesizes input sequences, feeds them to both the approximate MDP $\hat{\mathcal{M}}$ and the SUT \mathcal{M} , and compares the probabilistic distribution of their outputs. If ProbBBC finds a witness of their deviation, it feeds the witness back to the learning phase. Otherwise, ProbBBC deems the approximate MDP $\hat{\mathcal{M}}$ converged to the SUT \mathcal{M} and returns the satisfaction probability $p_{\mathcal{M}_s, \varphi}$ of the specification φ on the SUT \mathcal{M} , which is already obtained in the previous phase.

Moreover, we optimized the validation phase using the information used for MDP learning. For example, we immediately terminate the validation phase and return to the learning phase when some assumptions regarding the estimation of transition probabilities prove to be incorrect.

We implemented a prototype tool of ProbBBC. We conducted experiments to evaluate the performance of ProbBBC using benchmarks related to CPS or IoT scenarios, which are also used in previous papers on MDP learning [33] or MDP testing via learning [4]. Results from our experiment indicate that the satisfaction probability $p_{\mathcal{M}_s, \varphi}$ estimated by ProbBBC is usually close to the true value. Moreover, ProbBBC tends to estimate a better probability than an existing approach [4], especially when the observability in the SUT \mathcal{M} is limited.

Summary of the contributions. Our contributions are summarized as follows.

- We propose probabilistic black-box checking (ProbBBC) by combining active MDP learning, probabilistic model checking, and statistical hypothesis testing.
- We optimized the validation phase using the information for MDP learning.
- Our experiment results suggest that our approach outperforms an existing approach.

Outline. We show some related approaches in Section 2. After recalling the preliminaries in Section 3, we introduce probabilistic black-box checking (ProbBBC) in Section 4. We show the experimental evaluation in Section 5, and conclude in Section 6.

2 RELATED WORK

Table 1 summarizes testing methods for black-box systems with external inputs and probabilistic branching. These methods are categorized into two approaches: PMC-based and SMC-based approaches. ProbBBC is categorized as the PMC-based approach. In the PMC-based approach, we learn an MDP $\hat{\mathcal{M}}$ that approximates the SUT \mathcal{M} , then compute an optimal strategy s for the learned MDP $\hat{\mathcal{M}}$ along with its corresponding probability using PMC. When the learned MDP $\hat{\mathcal{M}}$ closely approximates the SUT \mathcal{M} , we can expect the strategy s to be near-optimal for the SUT \mathcal{M} as well.

¹We remark that estimation of $p_{\mathcal{M}_s, \varphi}$ by executing \mathcal{M}_s is not trivial since \mathcal{M} is a black-box system, whereas s requires the information on the current state of the system in each step. To address this challenge, we execute \mathcal{M}_s using $\hat{\mathcal{M}}$ as a scaffold; we explain this technique in Section 4.2.1.

Table 1. Testing methods for black-box systems with external inputs and probabilistic branching. PMC stands for probabilistic model checking, and SMC stands for statistical model checking.

	(Near-) optimal strategy synthesis	Probability computation	MDP learning
Ours	PMC	PMC	Active [33]
[4]	PMC	PMC	Passive [24]
[10, 12]	random sampling	SMC	N/A
[9]	delayed Q-learning [32]	SMC	N/A

Aichernig and Tappler [4] propose another method of this approach, which we denote as ProbBlackReach. The main difference from ProbBBC is in the MDP learning algorithm: ProbBBC *actively* learns an MDP with L^*_{MDP} [33], whereas ProbBlackReach *passively* learns an MDP with AALERGIA [24]. An active MDP learning algorithm adaptively samples execution traces for learning within the learning algorithm. In contrast, a passive MDP learning algorithm requires externally constructed execution traces. In [4], they use an ε -greedy algorithm to sample execution traces. This algorithm primarily samples traces using the the best strategy at the time but also employs a random input with probability ε . Due to its greedy sampling approach, the method often gets stuck in a suboptimal strategy, as we observe experimentally in Section 5. In contrast, ProbBBC tends outperform in finding a near-optimal strategy because L^*_{MDP} samples execution traces so that the transition function from each state can be identified, and the obtained execution traces tend to cover broader behavior. Another less significant difference lies in the properties each method supports: ProbBBC supports safety LTL, whereas ProbBlackReach is limited to reachability properties.

Statistical model checking (SMC) [3] estimates the satisfaction probability of a specification through random sampling of execution traces. Because of the sampling-based approach of SMC, it can handle black-box systems. SMC is primarily for purely probabilistic systems with no external inputs (e. g., Markov chains rather than MDPs), and when we apply SMC to systems with external inputs, we need to synthesize a (near-) optimal strategy to determine inputs. In [12, 22], strategies are constructed by random sampling with a concise encoding of each strategy. In [9], strategies are constructed by delayed Q-learning [32]. These methods, which directly learn and apply a strategy on the SUT, require the current state of the SUT to be observable. In contrast, ProbBBC indirectly uses a strategy via the approximate MDP, and the current state can be unobservable.

Schematically, ProbBBC is based on *black-box checking (BBC)* [29]. ProbBBC is a quantitative extension of BBC in the following sense: i) It handles systems with *probabilistic* transitions, which is more general than *deterministic* transitions supported by BBC; ii) It tries to return a *strategy* to *maximize* the satisfaction probability of the given property φ , which is a quantitative generalization of returning an *input sequence violating* φ . Furthermore, BBC for a deterministic system \mathcal{M} against an LTL formula φ is reducible to ProbBBC for \mathcal{M} against $\neg\varphi$ by checking if the maximum satisfaction probability of $\neg\varphi$ is 1.

3 PRELIMINARIES

For a set S , we denote its power set by $\mathcal{P}(S)$, the set of finite sequences of S elements by S^* , and the set of the infinite sequence of S by S^ω . For an infinite sequence $s = s_0, s_1, \dots \in S^\omega$ and $i, j \in \mathbb{N}$ where $i \leq j$, we denote the subsequence $s_i, s_{i+1}, \dots, s_j \in S^*$ by $s[i, j]$ and $s_i, s_{i+1}, \dots \in S^\omega$ by $s[i, \infty]$. We write $s \cdot s'$ for the concatenation of a finite sequence $s \in S^*$ and an infinite sequence $s' \in S^\omega$ of S . A *trace* σ is an alternating sequence of inputs and outputs (i.e., $\sigma \in \Sigma^{\text{out}} \times (\Sigma^{\text{in}} \times \Sigma^{\text{out}})^*$). We denote the set of probability distributions over S by $\text{Dist}(S)$: for any $\mu : S \rightarrow [0, 1]$ in $\text{Dist}(S)$, $\sum_{s \in S} \mu(s) = 1$ holds. For sets X and Y , we denote $X \subseteq_{\text{fin}} Y$ if $X \subseteq Y$ holds and X is finite.

3.1 Model of systems and specification logic

Here, we briefly review some notions related to (probabilistic) model checking. See, e. g., [18] for a more formal reasoning of probabilities using measure theory.

3.1.1 Mealy machine.

Definition 3.1 (Mealy machine). A (deterministic) Mealy machine is a 5-tuple $\hat{\mathcal{M}} = (Q, \Sigma^{\text{in}}, \Sigma^{\text{out}}, q_0, \Delta)$, where Q is the finite set of states, Σ^{in} and Σ^{out} are the input and output alphabets, $q_0 \in Q$ is the initial state, and $\Delta : (Q \times \Sigma^{\text{in}}) \rightarrow (Q \times \Sigma^{\text{out}})$ is the transition function. We write $\mathcal{L}(\mathcal{M}) \subseteq (\Sigma^{\text{in}} \times \Sigma^{\text{out}})^\omega$ for the *language* of a Mealy machine \mathcal{M} defined as follows:

$$\mathcal{L}(\mathcal{M}) := \{(a_0, b_0), (a_1, b_1), \dots \mid \exists q_1, q_2, \dots \in Q^\omega, \forall i \in \mathbb{N}. \Delta(q_i, a_i) = (q_{i+1}, b_i)\}.$$

For $\sigma \in (\Sigma^{\text{in}})^\omega$ and a Mealy machine \mathcal{M} , we write $\mathcal{M}(\sigma)$ for the output obtained by feeding σ to \mathcal{M} . More precisely, $\mathcal{M}(\sigma)$ for $\sigma = a_0, a_1, \dots \in (\Sigma^{\text{in}})^\omega$ is defined as $b_0, b_1, \dots \in (\Sigma^{\text{out}})^\omega$ such that $(a_0, b_0), (a_1, b_1), \dots \in \mathcal{L}(\mathcal{M})$. Notice that this notation is well-defined since \mathcal{M} is deterministic. We write $\mathcal{L}^{\text{fin}}(\mathcal{M})$ for the *finite language* of \mathcal{M} defined as the set of finite prefixes of $\mathcal{L}(\mathcal{M})$: $\mathcal{L}^{\text{fin}}(\mathcal{M}) := \{\sigma \in (\Sigma^{\text{in}} \times \Sigma^{\text{out}})^* \mid \exists \sigma' \in (\Sigma^{\text{in}} \times \Sigma^{\text{out}})^\omega. \sigma \cdot \sigma' \in \mathcal{L}(\mathcal{M})\}$.

3.1.2 Markov decision process. We use *Markov decision processes (MDPs)* for modeling systems that exhibit stochastic behavior.

Definition 3.2 (Markov decision process). A Markov decision process (MDP) is a 6-tuple $\hat{\mathcal{M}} = (Q, \Sigma^{\text{in}}, \Sigma^{\text{out}}, q_0, \Delta, L)$, where Q is the finite set of states, Σ^{in} and Σ^{out} are input and output alphabet, $q_0 \in Q$ is the initial state, $\Delta : (Q \times \Sigma^{\text{in}}) \rightarrow \text{Dist}(Q)$ is the probabilistic transition function, and $L : Q \rightarrow \Sigma^{\text{out}}$ is the labeling function. A *path* ρ of $\hat{\mathcal{M}}$ is an element of $Q \times (\Sigma^{\text{in}} \times Q)^*$. We write $\text{Path}_{\hat{\mathcal{M}}}$ for the set of all paths of MDP $\hat{\mathcal{M}}$. We say MDP $\hat{\mathcal{M}}$ is *deterministic* iff for any $q \in Q$ and for any $a \in \Sigma^{\text{in}}, q', q'' \in Q$, $\Delta(q, a)(q') > 0$ and $\Delta(q, a)(q'') > 0$ implies $q' = q''$ or $L(q') \neq L(q'')$.

In this paper, we let $\Sigma^{\text{out}} = \mathcal{P}(\text{AP})$ where **AP** is a set of relevant atomic propositions, and the labeling function L returns the propositions that hold at the given state.

An execution of an MDP $\hat{\mathcal{M}} = (Q, \Sigma^{\text{in}}, \Sigma^{\text{out}}, q_0, \Delta, L)$ can be seen as an interaction with a system and a controller. An execution of $\hat{\mathcal{M}}$ starts at the initial state q_0 . Suppose the system is at state q . The controller chooses an input $a \in \Sigma^{\text{in}}$ to the system; then, the system's state changes based on the probability distribution $\Delta(q, a)$. This interaction between a system and its controller is expressed by a path ρ of states and inputs starting at the initial state q_0 , i. e., $\rho = q_0, a_1, q_1, a_2, q_2, \dots, a_n, q_n$; at each state q_k , the input a_{k+1} is chosen by the controller, and the next state is chosen probabilistically according to the distribution $\Delta(q_k, a_{k+1})$.

A path $\rho = q_0, a_1, q_1, a_2, q_2, \dots, a_n, q_n$ induces a trace σ of the MDP $\hat{\mathcal{M}}$; $\sigma = b_0, a_1, b_1, a_2, b_2, \dots, a_n, b_n$, where for each $k \in \{0, 1, \dots, n\}$, $b_k = L(q_k)$. If the MDP $\hat{\mathcal{M}}$ is deterministic, the path ρ corresponding to a trace σ is uniquely identified.

The intuition of the interaction between a system and its controller is formalized by modeling the controller as a *strategy*.

Definition 3.3 (Strategy). For an MDP $\hat{\mathcal{M}} = (Q, \Sigma^{\text{in}}, \Sigma^{\text{out}}, q_0, \Delta, L)$, a *strategy* for $\hat{\mathcal{M}}$ is a function $s : \text{Path}_{\hat{\mathcal{M}}} \rightarrow \text{Dist}(\Sigma^{\text{in}})$.

A strategy s can be seen as a controller that probabilistically chooses an input $a \in \Sigma^{\text{in}}$ to be fed to $\hat{\mathcal{M}}$ from a path $\rho \in \text{Path}_{\hat{\mathcal{M}}}$ that represents the execution of the system so far based on the probability distribution $s(\rho)$.

An interacting system composed of an MDP $\hat{\mathcal{M}}$ and a strategy s is modeled as a discrete-time Markov chain (DTMC) [14]. We write $\hat{\mathcal{M}}_s$ for this DTMC.

A strategy is *finite-memory* if its choice of inputs depends only on the current state and a finite mode updated by each input and state, not on the entire path. Any finite-memory deterministic strategy can be encoded by a Mealy machine $(M, Q \times \Sigma^{\text{in}}, \Sigma^{\text{in}}, m_0, E)$, where the choice of input $a \in \Sigma^{\text{in}}$ depends only on M and Q (i. e., for any $m \in M, q \in Q$, and $a, a' \in \Sigma^{\text{in}}$, the second elements of $E(m, (q, a))$ and $E(m, (q, a'))$ are the same). For an MDP $\hat{M} = (Q, \Sigma^{\text{in}}, \Sigma^{\text{out}}, q_0, \Delta, L)$ and a finite-memory deterministic strategy s of \hat{M} encoded by a Mealy machine $(M, Q \times \Sigma^{\text{in}}, \Sigma^{\text{in}}, m_0, E)$, \hat{M}_s is a *finite-state* DTMC with the state space $Q \times M \times \Sigma^{\text{in}}$, initial state (q_0, m_0, a_0) , where $a_0 \in \Sigma^{\text{in}}$ is such that $(m, a_0) = E(m_0, (q_0, a))$ for some $m \in M$ and $a \in \Sigma^{\text{in}}$, and the transition probability $P((q_{k+1}, m_{k+1}, a_{k+1}) \mid (q_k, m_k, a_k))$ from $(q_k, m_k, a_k) \in Q \times M \times \Sigma^{\text{in}}$ to $(q_{k+1}, m_{k+1}, a_{k+1}) \in Q \times M \times \Sigma^{\text{in}}$ is $\Delta(q_k, L(m_k, (q_k, a_k)))(q_{k+1})$ if $L(m_k, (q_k, a_k)) = (m_{k+1}, a_{k+1})$, and otherwise 0.

3.1.3 Linear temporal logic (LTL). We use a probabilistic extension of the *linear temporal logic (LTL)* [30] for specifying the temporal properties of the behavior of a system.

Definition 3.4 (Syntax of LTL). For a finite set \mathbf{AP} of atomic propositions, the syntax of *linear temporal logic* is defined as follows, where $p \in \mathbf{AP}$ and $i, j \in \mathbb{N} \cup \{\infty\}$ satisfying $i \leq j$

$$\varphi, \psi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \psi \mid \mathcal{X}\varphi \mid \varphi \mathcal{U}_{[i,j]} \psi.$$

An LTL formula φ asserts a property on an infinite sequence $\pi \in (\mathcal{P}(\mathbf{AP}))^\omega$ of subsets of \mathbf{AP} . Intuitively, \top holds for any π ; p holds if the first element of π includes p ; $\neg\varphi$ holds if φ does not hold for π ; $\varphi \vee \psi$ holds if either φ or ψ holds for π ; and $\mathcal{X}\varphi$ holds if φ holds for $\pi[1, \infty]$. The formula $\varphi \mathcal{U}_{[i,j]} \psi$ asserts that (1) ψ becomes true in the future along with π that arrives within $l \in [i, j)$ steps and (2) φ continues to hold until ψ becomes true. More precisely, $\varphi \mathcal{U}_{[i,j]} \psi$ holds if ψ holds for some $\pi' = \pi[l, \infty]$ such that $l \in [i, j)$ and φ holds for $\pi[m, \infty]$ for any $m \in [0, l)$.

The semantics of LTL formulas is defined by the following satisfaction relation $(\pi, k) \models \varphi$ that represents that φ holds for $\pi[0, k]$. For an infinite sequence π , an index k , and an LTL formula φ , $(\pi, k) \models \varphi$ intuitively stands for “ π satisfies φ at k ”.

Definition 3.5 (Semantics of LTL). For an LTL formula φ , an infinite sequence $\pi = \pi_0, \pi_1, \dots \in (\mathcal{P}(\mathbf{AP}))^\omega$ of subsets of atomic propositions, and $k \in \mathbb{N}$, we define the satisfaction relation $(\pi, k) \models \varphi$ as follows.

$$\begin{aligned} (\pi, k) \models \top & \quad (\pi, k) \models p \iff p \in \pi_k & \quad (\pi, k) \models \neg\varphi & \iff (\pi, k) \not\models \varphi \\ (\pi, k) \models \varphi \vee \psi & \iff (\pi, k) \models \varphi \vee (\pi, k) \models \psi & \quad (\pi, k) \models \mathcal{X}\varphi & \iff (\pi, k+1) \models \varphi \\ (\pi, k) \models \varphi \mathcal{U}_{[i,j]} \psi & \iff \exists l \in [k+i, k+j). (\pi, l) \models \psi \wedge \forall m \in \{k, k+1, \dots, l-1\}. (\pi, m) \models \varphi \end{aligned}$$

If we have $(\pi, 0) \models \varphi$, we denote $\pi \models \varphi$.

We use the following syntax sugars of LTL formulas defined as follows.

$$\begin{aligned} \perp & \equiv \neg\top, \quad \varphi \wedge \psi \equiv \neg((\neg\varphi) \vee (\neg\psi)), \quad \varphi \rightarrow \psi \equiv (\neg\varphi) \vee \psi, \quad \diamond_{[i,j]}\varphi \equiv \top \mathcal{U}_{[i,j]}\varphi, \\ \square_{[i,j]}\varphi & \equiv \neg(\diamond_{[i,j]}\neg\varphi), \quad \varphi \mathcal{U} \psi \equiv \varphi \mathcal{U}_{[0,\infty)} \psi, \quad \diamond\varphi \equiv \diamond_{[0,\infty)}\varphi, \quad \square\varphi \equiv \square_{[0,\infty)}\varphi \end{aligned}$$

The intuition of \perp , $\varphi \wedge \psi$, and $\varphi \rightarrow \psi$ should be clear. The formula $\diamond_{[i,j]}\varphi$ asserts that “ φ holds eventually between i -step and j -step future”; the formula $\square_{[i,j]}\varphi$ stands for “ φ holds globally between i -step and j -step future”. We also introduce \mathcal{U} , \diamond , and \square for the unbounded versions of $\mathcal{U}_{[i,j]}$, $\diamond_{[i,j]}$, and $\square_{[i,j]}$.

Given a Mealy machine \mathcal{M} and an LTL formula φ , we write $\mathcal{M} \models \varphi$ if $\mathcal{M}(\sigma) \models \varphi$ for any $\sigma \in (\Sigma^{\text{in}})^\omega$. We write $\mathcal{M} \not\models \varphi$ if $\mathcal{M} \models \varphi$ does not hold.

Safety LTL is a subclass of LTL that consists of the LTL formulas expressing safety properties, whose violation can be witnessed by a *finite* sequence.

Definition 3.6 (Safety). An LTL formula φ is *safety* if, for any infinite sequence $\pi \in (\mathcal{P}(\mathbf{AP}))^\omega$ satisfying $\pi \not\models \varphi$, there is $i \in \mathbb{N}$ such that for any infinite sequence $\pi' \in (\mathcal{P}(\mathbf{AP}))^\omega$, we have $\pi[0, i] \cdot \pi' \not\models \varphi$.

3.1.4 Quantitative probabilistic model checking. *Probabilistic model checking (PMC)* [7] is a method to verify MDPs against LTL formulas. We use *quantitative PMC*, which computes the maximum (or minimum) satisfaction probability of the given LTL formula φ on the MDP \hat{M} .

Quantitative probabilistic model checking problem:

INPUT: An MDP $\hat{M} = (Q, \Sigma^{\text{in}}, \Sigma^{\text{out}}, q_0, \Delta, L)$ and an LTL formula φ

PROBLEM: Computes the maximum satisfaction probability $\max_{s \in \text{Sched}_{\hat{M}}} \mathbb{P}_{\hat{M}, s}(\varphi)$ of φ on \hat{M} , where $\text{Sched}_{\hat{M}}$ is the set of strategies of \hat{M} and $\mathbb{P}_{\hat{M}, s}(\varphi)$ is the probability of an execution of the DTMC \hat{M}_s satisfying φ .

For any MDP \hat{M} and an LTL formula φ , there is a finite-memory deterministic strategy s maximizing the satisfaction probability of φ on \hat{M} [20]. A probabilistic model checker, e. g., PRISM [19], takes an MDP \hat{M} and an LTL formula φ , and computes a finite-memory deterministic strategy s that maximizes the probability of \hat{M}_s satisfying φ ; it returns s as well as the probability p .

3.2 Active learning of automata and MDPs

Active automata learning is a class of algorithms to construct an automaton through interactions between the *learner* and a *teacher*. In the L^* algorithm [5], the best known active automata learning algorithm, the learner constructs the minimum DFA \mathcal{A} over Σ recognizing the target language $\mathcal{L}_{\text{tgt}} \subseteq \Sigma^*$ using *membership* and *equivalence* questions to the teacher. In a membership question, the learner asks if a word $w \in \Sigma^*$ is a member of \mathcal{L}_{tgt} , i. e., $w \in \mathcal{L}_{\text{tgt}}$. Membership questions are used to obtain sufficient information to construct a candidate DFA \mathcal{A}_{cnd} . In an equivalence question, the learner asks if the candidate DFA \mathcal{A}_{cnd} recognizes the target language \mathcal{L}_{tgt} , i. e., $\mathcal{L}(\mathcal{A}_{\text{cnd}}) = \mathcal{L}_{\text{tgt}}$. If we have $\mathcal{L}(\mathcal{A}_{\text{cnd}}) \neq \mathcal{L}_{\text{tgt}}$, the teacher returns a word *cex* satisfying $\text{cex} \in \mathcal{L}(\mathcal{A}_{\text{cnd}}) \Delta \mathcal{L}_{\text{tgt}}$ as a witness of $\mathcal{L}(\mathcal{A}_{\text{cnd}}) \neq \mathcal{L}_{\text{tgt}}$, where $\mathcal{L}(\mathcal{A}_{\text{cnd}}) \Delta \mathcal{L}_{\text{tgt}}$ is the symmetric difference between $\mathcal{L}(\mathcal{A}_{\text{cnd}})$ and \mathcal{L}_{tgt} , i. e., $\mathcal{L}(\mathcal{A}_{\text{cnd}}) \Delta \mathcal{L}_{\text{tgt}} = (\mathcal{L}(\mathcal{A}_{\text{cnd}}) \setminus \mathcal{L}_{\text{tgt}}) \cup (\mathcal{L}_{\text{tgt}} \setminus \mathcal{L}(\mathcal{A}_{\text{cnd}}))$. Equivalence questions are used to decide if the learning process can be finished.

The L^* algorithm uses a 2-dimensional array called an *observation table* to maintain the information obtained from the teacher. Fig. 2 illustrates an observation table during learning $\mathcal{L}_{\text{tgt}} = \{w \in (\text{ab})^* \mid (\# \text{ of } a \text{ in } w) \bmod 2 = (\# \text{ of } b \text{ in } w) \bmod 2 = 0\}$. The rows and columns of an observation table are indexed by finite sets of words $P \cup (P \cdot \Sigma)$ and S . In Fig. 2, P is displayed above the horizontal line: we have $P = \{\varepsilon, a, b\}$, $(P \cdot \Sigma) \setminus P = \{aa, ab, ba, bb\}$, and $S = \{\varepsilon, b\}$. The cell indexed by $(p, s) \in (P \cup (P \cdot \Sigma)) \times S$ shows if the concatenation $p \cdot s \in \Sigma^*$ is a member of \mathcal{L}_{tgt} . For example, the cell indexed by (a, b) shows $ab \notin \mathcal{L}_{\text{tgt}}$.

	ε	b
ε	\top	\perp
a	\perp	\perp
b	\perp	\top
aa	\top	\perp
ab	\perp	\perp
ba	\perp	\perp
bb	\top	\perp

Fig. 2. An observation table in L^* .

A DFA can be constructed from an observation table satisfying certain conditions. Fig. 3 shows the DFA constructed from the observation table in Fig. 2. In the DFA construction, a state is constructed for each unique row. For example, from the observation table in Fig. 2, three states are constructed, corresponding to the rows (\top, \perp) , (\perp, \perp) , and (\perp, \top) . A state is accepting if the cell indexed by (p, ε) is \top , where p is the index of a row corresponding to the state. The successor of the state corresponding to the row indexed by p with $a \in \Sigma$ is the state corresponding to the row indexed by $p \cdot a$. For example, from the observation table in Fig. 2, the successor of the state corresponding to (\top, \perp) , which is indexed by ε , with a is (\perp, \perp) , which is indexed by a . For

Algorithm 1: Outline of the candidate-generation procedure.

```

1 while Observation table is not closed or consistent do           // Candidate generation phase
2   | update  $P$  or  $S$  and fill the observation table with membership questions
3 return constructCandidateAutomaton( $P, S$ )

```

Algorithm 2: Outline of the equivalence checking procedure.

```

1 if  $\mathcal{L}_{\text{tgt}} = \mathcal{L}(\mathcal{A}_{\text{cnd}})$  then           // Equivalence checking phase with an equivalence question
2   | return OK( $\mathcal{A}_{\text{cnd}}$ )
3 else
4   | let  $cex$  be an evidence of  $\mathcal{L}_{\text{tgt}} \neq \mathcal{L}(\mathcal{A}_{\text{cnd}})$ , i. e.,  $cex \in \mathcal{L}_{\text{tgt}} \Delta \mathcal{L}(\mathcal{A}_{\text{cnd}})$ 
5   | return CEx( $cex$ )

```

Algorithm 3: Outline of the L^* algorithm for active automata learning.

```

1  $P \leftarrow \{\varepsilon\}; S \leftarrow \{\varepsilon\}$ 
2 while  $\top$  do
3   | while Observation table is not closed or consistent do           // Candidate generation phase
4     | update  $P$  or  $S$  and fill the observation table with membership questions
5     |  $\mathcal{A}_{\text{cnd}} \leftarrow$  constructCandidateAutomaton( $P, S$ )
6     | if  $\mathcal{L}_{\text{tgt}} = \mathcal{L}(\mathcal{A}_{\text{cnd}})$  then           // Equivalence checking phase with an equivalence question
7       | return  $\mathcal{A}_{\text{cnd}}$ 
8     | else
9       | let  $cex$  be an evidence of  $\mathcal{L}_{\text{tgt}} \neq \mathcal{L}(\mathcal{A}_{\text{cnd}})$ , i. e.,  $cex \in \mathcal{L}_{\text{tgt}} \Delta \mathcal{L}(\mathcal{A}_{\text{cnd}})$ 
10      | add prefixes of  $cex$  to  $P$  and fill the observation table with membership questions

```

the above construction, the observation table must satisfy the following two conditions. Note that the observation table in Fig. 2 satisfies these conditions.

closedness For each row indexed by $p \in (P \cdot \Sigma) \setminus P$, there is a row indexed by some $p' \in P$ with the same contents.

consistency For each pair of rows indexed by $p, p' \in P$ if their contents are the same, for any $a \in \Sigma$, the rows indexed by $p \cdot a$ and $p' \cdot a$ also have the same contents.

Algorithm 3 outlines the L^* algorithm. The L^* algorithm consists of two phases: candidate generation and equivalence checking phases. In the candidate generation phase, which is outlined in Algorithm 1, the learner increases P and S and fills the observation table using membership questions until the observation table becomes closed and consistent (lines 3 to 4). Once the observation table becomes closed and consistent, a hypothesis DFA \mathcal{A}_{cnd} is constructed (line 5). Then, in the equivalence checking phase outlined in Algorithm 2, the learner checks if \mathcal{A}_{cnd} recognizes the target language using an equivalence question (line 6). If we have $\mathcal{L}_{\text{tgt}} = \mathcal{L}(\mathcal{A}_{\text{cnd}})$, \mathcal{A}_{cnd} is returned. Otherwise, the teacher returns an evidence cex of $\mathcal{L}_{\text{tgt}} \neq \mathcal{L}(\mathcal{A}_{\text{cnd}})$ (line 9), and the prefixes of cex are added to P (line 10).

The L^*_{MDP} algorithm [33] is an extension of the L^* algorithm for active MDP learning. It learns a deterministic MDP using similar questions and an observation table. In the L^*_{MDP} algorithm, the teacher maintains a prefix-closed multiset \mathcal{S} of traces, which contains all the information obtained

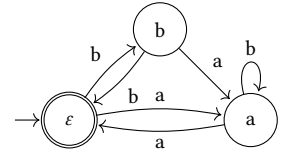


Fig. 3. Candidate DFA \mathcal{A}_{cnd} constructed from the observation table in Fig. 2.

	a	b
p	{p ↦ 15, q ↦ 15}	{q ↦ 25}
paq	{p ↦ 8, q ↦ 12}	{p ↦ 4, q ↦ 6}
pap	{p ↦ 5, q ↦ 5}	{q ↦ 12}
pbp	∅	∅
pbq	{p ↦ 4, q ↦ 6}	{p ↦ 2, q ↦ 3}
paqap	{p ↦ 3, q ↦ 3}	{q ↦ 4}
paqaq	{p ↦ 2, q ↦ 3}	{p ↦ 2, q ↦ 3}
paqbp	{p ↦ 3, q ↦ 3}	{q ↦ 4}
paqbq	{p ↦ 2, q ↦ 3}	{p ↦ 2, q ↦ 3}

Fig. 4. An observation table in L^*_{MDP} with $\Sigma^{in} = \{a, b\}$ and $\Sigma^{out} = \{p, q\}$.

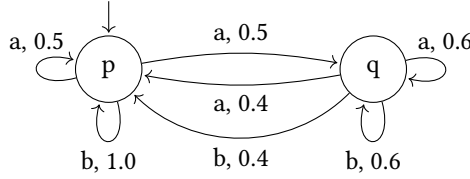


Fig. 5. Candidate MDP constructed from the observation table in Fig. 4. The label at each state represents the output.

by executing the system under learning and used to answer queries. Fig. 4 illustrates an observation table in the L^*_{MDP} algorithm. The following summarizes the major differences in the observation table.

- To learn an MDP with inputs Σ^{in} and outputs Σ^{out} , the indices are modified: the row indices are $P \cup (P \cdot \Sigma^{in} \cdot \Sigma^{out}) \subseteq_{fin} \Sigma^{out} \cdot (\Sigma^{in} \cdot \Sigma^{out})^*$ and the column indices are $S \subseteq_{fin} \Sigma^{in} \cdot (\Sigma^{out} \cdot \Sigma^{in})^*$.
- The cell indexed by (p, s) represents a function $T_{p,s}: \Sigma^{out} \rightarrow \mathbb{N}$ mapping each output to the frequency of its appearance in \mathcal{S} after $p \cdot s$. For instance, in Fig. 4, the cell indexed by (p, a) shows that we observed each of “pap” and “paq” for 15 times.

The notion of closedness and consistency are also updated to statistically compare the rows with a *Hoeffding bound* [16]. To maintain such an observation table, the membership question is replaced with the following questions: i) Given a trace $\sigma \in \Sigma^{out} \cdot (\Sigma^{in} \cdot \Sigma^{out})^*$, it returns the function mapping $b \in \Sigma^{out}$ to the frequency $\mathcal{S}(\sigma \cdot b)$ of $\sigma \cdot b$ in the multiset \mathcal{S} ; ii) Given a trace $\sigma \in \Sigma^{out} \cdot (\Sigma^{in} \cdot \Sigma^{out})^*$, it returns if \mathcal{S} contains sufficient information to estimate the output distribution after σ ; iii) It asks the teacher to refine \mathcal{S} by sampling traces rarely appearing in \mathcal{S} .

The MDP construction from an observation table is also similar to the DFA construction in the L^* algorithm. Fig. 5 shows the MDP constructed from the observation table in Fig. 4. State construction is based on the comparison of rows by Hoeffding bound. Transitions are constructed by estimating the probability distribution of the successors using the contents of the cells.

3.3 Black-box checking

Black-box checking (BBC) [29] is a method for testing a black-box system \mathcal{M} against its specification φ . BBC takes the following inputs: i) a deterministic black-box system \mathcal{M} that takes $\sigma \in (\Sigma^{in})^\omega$ as input and outputs $\mathcal{M}(\sigma) \in \mathcal{P}(\text{AP})^\omega$ and ii) a safety LTL formula φ . The input to and the output from \mathcal{M} are both streams. The aim of BBC is to find a counterexample $\sigma' \in (\Sigma^{in})^*$ that witnesses the violation of φ : For any $\sigma'' \in (\Sigma^{in})^\omega$, the counterexample σ' satisfies $\mathcal{M}(\sigma' \cdot \sigma'') \not\models \varphi$. If such a counterexample is not found, BBC reports so.

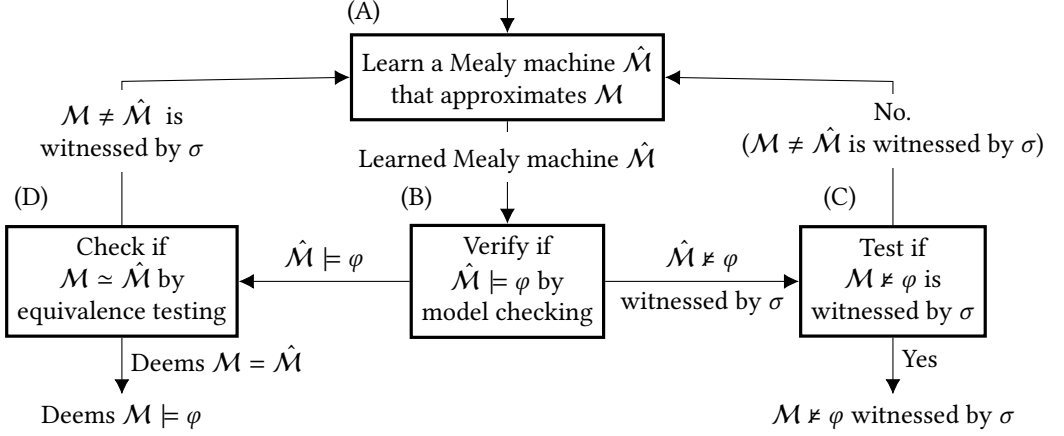


Fig. 6. The workflow of black-box checking.

BBC consists of learning, synthesis, and validation phases. BBC iterates the learning phase to obtain a Mealy machine \hat{M} that approximates the behavior of M , the synthesis phase to synthesize a witness σ of $\hat{M} \not\models \varphi$, and the validation phase to check whether σ is also the true counterexample to $M \models \varphi$. By combining automata learning and model checking, BBC guides the testing procedure efficiently instead of randomly generating various test inputs and expecting some of them to falsify the specification φ .

Fig. 6 illustrates the workflow of BBC. We explain each component in the figure below. (The heading of each item corresponds to a box in Fig. 6.)

- (A) Using the candidate generation phase of a variant of the L^* algorithm, BBC obtains a Mealy machine \hat{M} approximating the behavior of M .
- (B) Then, BBC verifies \hat{M} against the specification φ using model checking.
- (C) Suppose the model checker in (B) asserts $\hat{M} \not\models \varphi$. Let $\sigma \in (\Sigma^{\text{in}})^*$ be a counterexample that witnesses $\hat{M} \not\models \varphi$; notice that such σ exists in $(\Sigma^{\text{in}})^*$ because φ is a safety LTL formula. This σ may not be a valid counterexample for M because \hat{M} is merely an approximation of M . BBC checks whether σ is a valid counterexample also for M by feeding σ to M and checking whether $M(\sigma) \not\models \varphi$ holds. If $M(\sigma) \not\models \varphi$, then σ is a valid counterexample for M . Otherwise, although σ does not witness $M \not\models \varphi$, it does witness the behavioral difference between M and \hat{M} . Therefore, BBC adds σ to the data used by the learning phase and jumps to Step (A). Then, the learning procedure progresses to learn more precise \hat{M} .
- (D) Suppose the model checker used in Step (B) successfully verifies $\hat{M} \models \varphi$. Then, BBC tests the equivalence between M and \hat{M} , typically by randomly generating many elements of $(\Sigma^{\text{in}})^*$ and feeding them to M and \hat{M} . If this step finds $\sigma \in (\Sigma^{\text{in}})^*$ such that $M(\sigma) \neq \hat{M}(\sigma)$, then this σ witnesses the behavioral difference between M and \hat{M} ; therefore, BBC adds σ to the data used by the learning phase and jumps to Step (A). Otherwise, BBC reports that no counterexample is found deeming \hat{M} is equivalent to M .

In Fig. 6, (A) corresponds to the learning phase, (B) corresponds to the synthesis phase, and (C–D) correspond to the validation phase. Notice that the validation phase (C) and (D) as a whole checks the equivalence of M and \hat{M} . Instead of checking the equivalence only by the inefficient random test in (D), BBC first checks a necessary condition of the equivalence: $M(\sigma) \not\models \varphi$ for σ such that $\hat{M}(\sigma) \not\models \varphi$ obtained in (C). BBC then proceeds to (D) only if this check in (C) passes.

4 PROBABILISTIC BLACK-BOX CHECKING

We present *probabilistic black-box checking* (ProbBBC). ProbBBC is an extension of BBC in Section 3.3 for stochastic systems. We assume that the system under test (SUT) is a *black-box MDP* with finite states, i. e., there is an underlying MDP $(Q, \Sigma^{\text{in}}, \Sigma^{\text{out}}, q_0, \Delta, L)$ with $|Q| < \infty$ representing the SUT \mathcal{M} , but we only know Σ^{in} and Σ^{out} . We assume that the MDP is deterministic, which intuitively requires sufficient observability to distinguish the next states. We also assume that the SUT is *executable*, i. e., we can perform the following probabilistic operations on the *unobservable* current state $q \in Q$ of the SUT \mathcal{M} .

- We can reset the current state q of the SUT \mathcal{M} to the initial state q_0 and obtain the output $L(q_0)$.
- For an input $a \in \Sigma^{\text{in}}$, we can update the current state q of the SUT \mathcal{M} to $q' \in Q$ according to the distribution $\Delta(q, a)$ and obtain the output $L(q')$ after the transition.

We believe that these assumptions are acceptable in many usage scenarios. For example, in reinforcement learning, Q-learning [36] assumes that the environment is a black-box MDP that is finite, executable, and fully observable, which is more restrictive than ours.

Along with the stochastic extension of the SUT, we also change the problem to synthesize an optimal strategy rather than finding an input violating the given LTL formula. The problem we approximately solve with ProbBBC is summarized as follows.

Optimal strategy synthesis problem:

INPUT: A black-box MDP \mathcal{M} and an LTL formula φ

PROBLEM: Find a strategy s maximizing the satisfaction probability $p_{\mathcal{M}_s, \varphi}$ of φ on \mathcal{M}

4.1 Overview of ProbBBC

Fig. 7 outlines ProbBBC. As the conventional BBC does, ProbBBC consists of the learning phase ((A) in Fig. 7), the synthesis phase ((B) in Fig. 7), and the validation phase ((C–E) in Fig. 7). Given a black-box system \mathcal{M} and an LTL formula φ , ProbBBC applies (1) the candidate generation phase of L^*_{MDP} in Section 3.2 to construct an MDP $\hat{\mathcal{M}}$ that approximates \mathcal{M} in the learning phase ((A) in Fig. 7) and (2) quantitative probabilistic model checking [7] to synthesize s that maximizes the probability of $\hat{\mathcal{M}}_s$ satisfying φ in the synthesis phase ((B) in Fig. 7).

The goal of the validation phase is the same as that of the conventional BBC: to check that \mathcal{M} and $\hat{\mathcal{M}}$ are equivalent to each other. To this end, the validation phase first checks the necessary condition of the equivalence: $p_{\mathcal{M}_s, \varphi}$ is the same as $p_{\hat{\mathcal{M}}_s, \varphi}$, where $p_{\mathcal{M}_s, \varphi}$ and $p_{\hat{\mathcal{M}}_s, \varphi}$ are the satisfaction probabilities of φ on \mathcal{M}_s and $\hat{\mathcal{M}}_s$ ((C) and (D) in Fig. 7). We call this procedure *strategy-guided comparison* of \mathcal{M} and $\hat{\mathcal{M}}$. If this necessary condition seems to be satisfied, the validation phase applies the equivalence checking phase of L^*_{MDP} to check the equivalence of \mathcal{M} and $\hat{\mathcal{M}}$ *without* guided by the strategy s ((E) in Fig. 7).

The strategy-guided comparison is inspired by the witness checking of the conventional BBC ((C) in Fig. 6). However, unlike the witness checking of the conventional BBC, which can check whether the counterexample obtained by model checking is a true counterexample for \mathcal{M} by executing \mathcal{M} only once, there are following challenges in the strategy-guided comparison of ProbBBC.

- ProbBBC must estimate the *probability* $p_{\mathcal{M}_s, \varphi}$ of \mathcal{M}_s satisfying φ . A one-shot execution of \mathcal{M}_s is not enough to estimate this probability.
- ProbBBC needs to decide whether $p_{\mathcal{M}_s, \varphi}$ differs from $p_{\hat{\mathcal{M}}_s, \varphi}$ obtained in the synthesis phase. However, since an estimation of $p_{\mathcal{M}_s, \varphi}$ is a probabilistic variable that is not guaranteed to be the true probability, simply comparing an obtained estimation with $p_{\hat{\mathcal{M}}_s, \varphi}$ is not enough.

To address these challenges, ProbBBC first executes \mathcal{M}_s multiple times and estimates the probability $p_{\mathcal{M}_s, \varphi}$ from the samples ((1) of (C) in Fig. 7). Then, ProbBBC conducts statistical hypothesis testing with the null hypothesis $p_{\mathcal{M}_s, \varphi} = p_{\hat{\mathcal{M}}_s, \varphi}$ ((2) of (C) in Fig. 7). If $p_{\mathcal{M}_s, \varphi} \neq p_{\hat{\mathcal{M}}_s, \varphi}$ is established from the hypothesis testing, ProbBBC constructs a trace that witnesses $\mathcal{M} \neq \hat{\mathcal{M}}$; this trace is added to the data that the learning phase uses ((D) in Fig. 7).

If the strategy-guided comparison fails to find an evidence of $\mathcal{M} \neq \hat{\mathcal{M}}$, we compare \mathcal{M} and $\hat{\mathcal{M}}$ using the equivalence checking phase of the L^*_{MDP} algorithm in Section 3.2 ((E) in Fig. 7). In our implementation, among various choices of the input construction (e. g., the W-method [11]), we use uniform random sampling, i. e., each input $a \in \Sigma^{\text{in}}$ is sampled from the uniform distribution over Σ^{in} and the sampling stops with a certain stop probability. Moreover, the stop probability is reduced when we fail to find an evidence of $\mathcal{M} \neq \hat{\mathcal{M}}$ so that any length of inputs are eventually sampled. Such an adaptive change of stop probability makes the parameters robust to the complexity of \mathcal{M} .

To enhance ProbBBC, we utilize the information obtained during the strategy-guided comparison also in the learning phase. In the strategy-guided comparison, ProbBBC adds the traces obtained by sampling \mathcal{M}_s to the observation table used in the learning phase. The closedness and the consistency of the observation table are periodically checked; if the observation table turns out to not-closed or inconsistent, then the execution of the validation phase is stopped, and the learning phase starts to learn new $\hat{\mathcal{M}}$ with the new observation table. The interruption also happens when the validation phase discovers a trace that is impossible in $\hat{\mathcal{M}}$. These optimizations are achieved by (1) sharing the multiset \mathcal{S} of traces mentioned in Section 3.2 between the learning and validation phases and (2) adding the traces discovered during the validation phase to \mathcal{S} .

Notice that the combination of the synthesis and the validation phases ((B–E) in Fig. 7) can be considered as the equivalence checking phase of the L^*_{MDP} algorithm because, overall, it tries to find an evidence of $\mathcal{M} \neq \hat{\mathcal{M}}$. Therefore, properties of L^*_{MDP} (such as convergence) shown in [33] also holds for ProbBBC with additional discussion on the strategy-guided comparison. As a corollary of the convergence, we have the correctness of ProbBBC, which we show in Section 4.3.

4.2 Detail of the strategy-guided comparison in the validation phase

In the following, we explain the detail of the validation phase of ProbBBC, focusing on the strategy-guided comparison. This part is of the largest technical novelty among the three phases.

4.2.1 Comparison of \mathcal{M} and $\hat{\mathcal{M}}$ with a strategy. Algorithm 4 outlines our algorithm to compare \mathcal{M} and $\hat{\mathcal{M}}$ with strategy s , which corresponds to the box (C) in Fig. 7. In the loop starting from line 2, ProbBBC repeatedly samples traces of the SUT \mathcal{M} up to the required sample size N . For each iteration, we obtain a trace σ by executing the SUT \mathcal{M} (line 3), recording it to the multiset \mathcal{S} shared with the learning phase (line 4), and check if σ satisfies φ or not (line 5). If φ is *bounded*, i. e., its satisfaction can be decided by traces of length k for some $k \in \mathbb{N}$, one can sample traces of length k . Otherwise, the trace length must be randomly decided. After the sampling, we estimate the satisfaction probability $\bar{p}_{\mathcal{M}_s, \varphi}$ of φ by \mathcal{M}_s (line 7) and compare it with the satisfaction probability $p_{\hat{\mathcal{M}}_s, \varphi}$ by the learned MDP (line 8), which is computed by quantitative probabilistic model checking.

In executing the SUT \mathcal{M} in line 3 in `SampleSingleTrace`, ProbBBC needs to execute \mathcal{M}_s . However, s requires the path of \mathcal{M}_s to produce an input to be fed to \mathcal{M} , which is not possible in this case because the sequence of the states of \mathcal{M} is unknown. We use the path of $\hat{\mathcal{M}}$ instead; in executing \mathcal{M}_s , ProbBBC also runs $\hat{\mathcal{M}}_s$ and maintains the corresponding path ρ in the MDP $\hat{\mathcal{M}}$. Then, $s(\rho)$ is fed to \mathcal{M} as the next input. Such use of $\hat{\mathcal{M}}$ is justified by the convergence of $\hat{\mathcal{M}}$ to \mathcal{M} in the limit. Notice that such a path $\rho = q_0, a_1, q_1, \dots, a_n, q_n$ is uniquely determined because (1) we know the initial state q_0 and the inputs a_1, a_2, \dots, a_n , (2) we can observe the output b_0, b_1, \dots, b_n of the \mathcal{M} ,

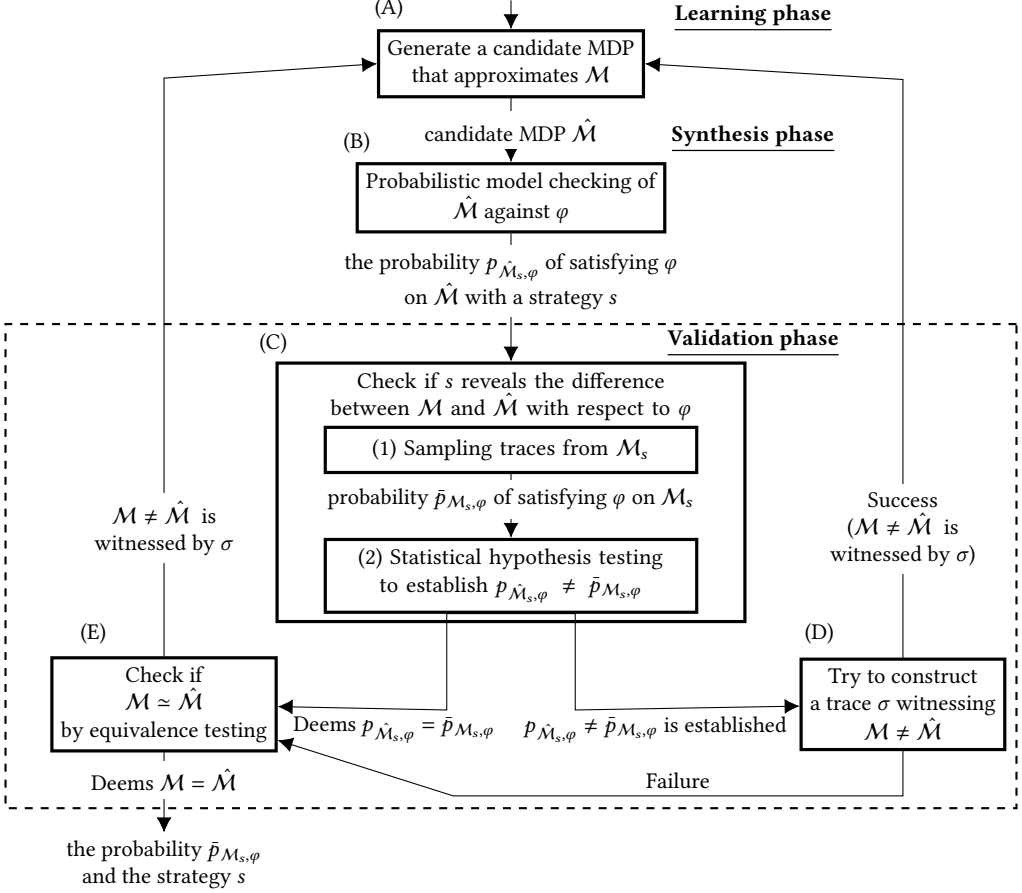


Fig. 7. The workflow of probabilistic black-box checking.

Algorithm 4: Comparison of \mathcal{M} and $\hat{\mathcal{M}}$ with a strategy s and an LTL formula φ .

1 Function CompareWithStrategy($\hat{\mathcal{M}}, \mathcal{M}, s, p_{\hat{\mathcal{M}}_s, \varphi}, \varphi$):

input : An MDP $\hat{\mathcal{M}}$, the SUT \mathcal{M} , a strategy s , probability $p_{\hat{\mathcal{M}}_s, \varphi}$, and a safety LTL formula φ
output: If $\hat{\mathcal{M}}$ and \mathcal{M} are deemed equivalent for the inputs determined by s with respect to φ
2 for $i \leftarrow 1$ **to** N **do**
3 $\sigma \leftarrow \text{SampleSingleTrace}(\mathcal{M}, s)$
4 **add** the prefixes of σ **to** \mathcal{S}
5 **if** $\sigma \models \varphi$ **then** $x_i \leftarrow 1$
6 **else** $x_i \leftarrow 0$
7 $\bar{p}_{\mathcal{M}_s, \varphi} \leftarrow \sum_{i=0}^k x_i / N$; $std_x \leftarrow \text{StandardDeviation}(x_1, x_2, \dots, x_N)$

 // Conduct Student's t-testing, where the null hypothesis is $p_{\hat{\mathcal{M}}_s, \varphi} = \bar{p}_{\mathcal{M}_s, \varphi}$
8 return StudentTesting($p_{\hat{\mathcal{M}}_s, \varphi}, \bar{p}_{\mathcal{M}_s, \varphi}, std_x, N$)

Algorithm 5: Construction of the trace differentiating the SUT and an approximate MDP.

```

1 Function ConstructWitnessTrace( $\hat{\mathcal{M}}, \mathcal{S}$ ):
   input : An MDP  $\hat{\mathcal{M}}$  and a multiset  $\mathcal{S}$  of traces of the SUT
   output: If a trace  $\sigma$  witnessing  $\mathcal{M} \neq \hat{\mathcal{M}}$  is found in  $\mathcal{S}$ , returns  $\sigma$ . Otherwise, returns  $\perp$ 
2   let  $\mathcal{S}_{\text{set}}$  be  $\mathcal{S}$  without multiplicity
3   while  $\mathcal{S}_{\text{set}} \neq \emptyset$  do
4      $\sigma \leftarrow$  pop one of the shortest traces from  $\mathcal{S}_{\text{set}}$ 
5     let  $\sigma^- \cdot o = \sigma$  such that  $\sigma^- \in (\Sigma^{\text{out}} \times \Sigma^{\text{in}})^*$  and  $o \in \Sigma^{\text{out}}$ 
6      $p_{\hat{\mathcal{M}}} \leftarrow$  the probability of observing  $o$  in  $\hat{\mathcal{M}}$  after  $\sigma^-$ 
7      $p_{\mathcal{S}} \leftarrow \mathcal{S}(\sigma)/\mathcal{S}(\sigma^-)$  //  $p_{\mathcal{S}}$  approximates the probability  $p_{\mathcal{M}}$  to observe  $o$ 
      after  $\sigma^-$  in  $\mathcal{M}$ .
8     if  $|p_{\hat{\mathcal{M}}} - p_{\mathcal{S}}|$  is greater than a bound then return  $\sigma$ 
9   return  $\perp$ 

```

and (3) the successor q_{k+1} in $\hat{\mathcal{M}}$ is uniquely determined from the previous state q_k , the input a_{k+1} , and the output b_{k+1} because $\hat{\mathcal{M}}$ is a *deterministic* MDP.

The sample size N needs to be sufficiently large so that the estimated probability $\bar{p}_{\mathcal{M}_s, \varphi}$ is close to the true probability of \mathcal{M}_s satisfies φ . ProbBBC decides N based on the parameters ϵ and δ specified by a user so that $\mathbb{P}(|\bar{p}_{\mathcal{M}_s, \varphi} - p_{\mathcal{M}_s, \varphi}| \geq \epsilon) \leq \delta$, where $p_{\mathcal{M}_s, \varphi}$ is the true satisfaction probability of φ by the black-box DTMC \mathcal{M}_s . If $0 < p_{\mathcal{M}_s, \varphi} < 1$ holds, using the property of Chernoff bound [28], we have the following property among the sample size N and the parameters ϵ and δ : $\delta = 2e^{-2N\epsilon^2}$; hence, ProbBBC uses $N = \left\lceil \frac{\ln(2) - \ln(\delta)}{2\epsilon^2} \right\rceil$, which is also used in the context of SMC [21]. In our experiments, we directly fix N instead of deriving it from δ and ϵ .

For the comparison of $p_{\hat{\mathcal{M}}_s, \varphi}$ and $\bar{p}_{\mathcal{M}_s, \varphi}$ in line 8, we perform a one-sample Student's t-test [17], where the null hypothesis is $p_{\hat{\mathcal{M}}_s, \varphi} = \bar{p}_{\mathcal{M}_s, \varphi}$. We remark that $p_{\hat{\mathcal{M}}_s, \varphi}$ is the exact maximum probability of $\hat{\mathcal{M}}_s$ satisfying φ , which is obtained by probabilistic model checking. Here, the one-sample Student's t-test can be used because N is large, and the binomial distribution $B(N, p_{\mathcal{M}_s, \varphi})$ is reasonably close to a normal distribution [26].

4.2.2 Witness trace construction. If Algorithm 4 deems $\mathcal{M} \neq \hat{\mathcal{M}}$, we try to construct a witnessing trace σ . The constructed witness σ is used to refine the observation table in the learning phase.

Algorithm 5 outlines the witness construction. From the multiset \mathcal{S} of the traces observed in \mathcal{M} , we construct the set \mathcal{S}_{set} of the traces in \mathcal{S} dropping the multiplicity from \mathcal{S} (line 2). Then, for each $\sigma \in \mathcal{S}_{\text{set}}$, we check if σ is an evidence of $\mathcal{M} \neq \hat{\mathcal{M}}$ (lines 3 to 8) in the increasing order of the length of the traces in \mathcal{S}_{set} . Notice that \mathcal{S} is prefix-closed, and hence \mathcal{S}_{set} is also prefix-closed. Therefore, Algorithm 5 tries to find one of the shortest traces σ witnessing $\mathcal{M} \neq \hat{\mathcal{M}}$.

At line 4, we pick one of the shortest traces σ from \mathcal{S}_{set} ; at lines 6 and 7, we compute the probability $p_{\hat{\mathcal{M}}}$ of observing σ in $\hat{\mathcal{M}}$ if the same input as σ is fed to $\hat{\mathcal{M}}$. This probability is computed by following the transitions of $\hat{\mathcal{M}}$ according to σ and multiplying the transition probabilities in $\hat{\mathcal{M}}$.

At line 7, we estimate the probability $p_{\mathcal{M}}$ to observe o by feeding the prefix σ^- of σ such that $\sigma = \sigma^- \cdot o$ to \mathcal{M} . Since \mathcal{M} is a black-box system, we can only estimate this probability. For the estimation, we use \mathcal{S} to approximate the trace distribution of \mathcal{M} and compute the probability to observe σ after σ^- . Concretely, we estimate $p_{\mathcal{M}}$ by $p_{\mathcal{S}} = \mathcal{S}(\sigma)/\mathcal{S}(\sigma^-)$, which is unbiased because the sampling of the outputs in \mathcal{S} follows the transition probabilities of \mathcal{M} .

At **line 8**, we compare $p_{\hat{\mathcal{M}}}$ and p_S . We decide whether they are different based on the criteria using a Chernoff bound mentioned in **Section 4.2.1**: we deem σ reveals the difference between $\hat{\mathcal{M}}$ and \mathcal{M} if $|p_{\hat{\mathcal{M}}} - p_S| > \sqrt{\frac{\ln(2) - \ln(\delta)}{2S(\sigma^+)}}$ holds for the parameter δ in **Section 4.2.1**. If σ differentiates $\hat{\mathcal{M}}$ and \mathcal{M} , it is returned (**line 8**). If we cannot find a trace differentiating $\hat{\mathcal{M}}$ and \mathcal{M} , we return \perp (**line 9**), which represents the failure of the trace construction.

4.2.3 Optimization using the observation table. To enhance the entire procedure of ProbBBC, our implementation applies the following optimizations to the validation phase.

- For each trace σ sampled from the SUT \mathcal{M}_s , we check if the candidate MDP $\hat{\mathcal{M}}$ has a path ρ corresponding to the trace σ . If there is no such path in $\hat{\mathcal{M}}$, the trace σ differentiates the SUT \mathcal{M} and the candidate MDP $\hat{\mathcal{M}}$. Then, the validation phase returns such σ as a witness of $\mathcal{M} \neq \hat{\mathcal{M}}$, and the learning phase starts.
- We also periodically update each cell of the observation table using the updated multiset \mathcal{S} of the traces obtained from \mathcal{M}_s , and check if the observation table is still closed and consistent. If the observation table is not closed or not consistent, we stop the validation phase and go back to the learning phase to refine $\hat{\mathcal{M}}$.

4.3 Convergence of ProbBBC

We prove the convergence of ProbBBC by showing that MDPs learned during an execution of ProbBBC converges to one equivalent to \mathcal{M} with probability 1 given that \mathcal{M} is a deterministic MDP $(Q, \Sigma^{\text{in}}, \Sigma^{\text{out}}, q_0, \Delta, L)$ with $|Q| < \infty$. Therefore, the strategy obtained by ProbBBC converges to the optimal one for \mathcal{M} . In this section, we assume that the equivalence checking procedure ((E) in **Fig. 6**) samples input, runs both \mathcal{M} and $\hat{\mathcal{M}}$ using the input multiple times, and subsequently compares the output distributions using hypothesis testing; the null hypothesis states that both distributions are identical.

4.3.1 Fair-sampling assumption. We first introduce the *fair sampling* assumption for the equivalence checking procedure, which postulates that the procedure explores each observable path of the provided \mathcal{M} infinitely many times with a positive probability. To formalize this assumption, we define the set $L_{\mathcal{M}}$ of *access sequences* of \mathcal{M} . Essentially, $L_{\mathcal{M}}$ is the set of the observable traces corresponding to a path \mathcal{M} where no state is repeated.

Definition 4.1 (Access sequence). A path $q_0, a_1, q_1, a_2, q_2, \dots, a_n, q_n$ is *cycle-free* if $i \neq j$ implies $q_i \neq q_j$. A cycle-free path $q_0, a_1, q_1, a_2, q_2, \dots, a_n, q_n$ is defined to be *maximal* if, for any $a \in \Sigma^{\text{in}}$ and for any $q' \in Q$, $q' \notin \{q_0, \dots, q_n\}$ implies $\Delta(q_n, a)(q') = 0$. A cycle-free path $q_0, a_1, q_1, a_2, q_2, \dots, a_n, q_n$ is *observable* if $\Delta(q_i, a_{i+1})(q_{i+1}) > 0$ for any $i \in [0, n-1]$. The set of *access sequences* $L_{\mathcal{M}}$ is the set of traces that correspond to maximal and observable cycle-free paths of \mathcal{M} ; concretely, $L_{\mathcal{M}}$ is defined by $\{L(q_0), a_1, L(q_1), a_2, \dots, a_n, L(q_n) \mid q_0, a_1, q_1, a_2, \dots, a_n, q_n \text{ is cycle-free, observable, and maximal}\}$; as such, $L_{\mathcal{M}}$ is a finite set.

Definition 4.2 (Fairness assumption). We say an equivalence testing procedure satisfies *fairness assumption* if it samples every element in $L_{\mathcal{M}}$ infinitely often with a positive probability.

For instance, the equivalence testing by uniform sampling (**Section 4.1**) used in our implementation satisfies the fairness assumption.

4.3.2 Outline of the convergence proof. We prove the following theorem on the convergence of ProbBBC.

THEOREM 4.3 (CONVERGENCE). *Under the fair-sampling assumption, the MDPs synthesized by (A) in an execution of ProbBBC converge to an MDP equivalent to \mathcal{M} with probability 1.*

Our proof of this theorem is built upon the convergence proof of the L^*_{MDP} [33], in which the authors have shown that, under the fair-sampling condition, L^*_{MDP} converges to a correct MDP with probability 1.

As discussed in Section 4.1, the synthesis and the validation phases of ProbBBC in combination can be viewed as an enhancement to the equivalence checking of L^*_{MDP} . Nonetheless, the convergence of ProbBBC does not directly follow from that of L^*_{MDP} due to our validation phase being based on the strategy-guided comparison, which may lead to a biased sampling of paths.

Crucially, we note that if we can prove each execution of the loop (A–D) terminates with probability 1 (i.e., the loop being *almost surely* terminating), we can then guarantee that the fair sampling procedure (E) is executed eventually. Consequently, no matter the strategies synthesized by (B), every element of $L_{\mathcal{M}}$ is fairly sampled, and therefore, the convergence of ProbBBC follows.

4.3.3 Almost-sure termination of the loop (A–D). Based on the discussion so far, we prove the almost-sure termination of the loop (A–D). We first elaborate on the observation table of L^*_{MDP} . As we mentioned in Section 3.2, the notion of closedness and consistency in L^*_{MDP} is based on the statistical row comparison with Hoeffding bound [16]. Let us write $\text{compatible}(r, r')$ if the rows indexed by r and r' are deemed equal based on the comparison with the Hoeffding bound. We also write $\text{eqRow}(r, r')$ if the rows indexed by r and r' should be deemed equal based on the *true* probability of \mathcal{M} . We expect that $\text{compatible}(r, r')$ is equivalent to $\text{eqRow}(r, r')$ after sufficiently many sampling of traces, which indeed holds.

LEMMA 4.4. *If \mathcal{S} contains sufficiently many traces, then $\text{compatible}(r, r') \iff \text{eqRow}(r, r')$ with probability 1 for any rows indexed by r and r' in the observation table.*

PROOF. From Theorem 5 of [33]. □

We also use the following property of L^*_{MDP} .

LEMMA 4.5. *If $\text{compatible}(r, r') \iff \text{eqRow}(r, r')$ for each r and r' in the row indices P of the observation table, then the MDP derived from the observation table is the smallest among ones that are consistent with the table.*

PROOF. From Lemma 13 of [33]. □

We show the proof sketch of the following theorem.

THEOREM 4.6. *The loop (A–D) of ProbBBC terminates with probability 1 irrespective of the strategies synthesized by (B).*

Proof sketch. Let $\hat{\mathcal{M}}^1, \hat{\mathcal{M}}^2, \dots$ be the sequence of MDPs learned by (A) in each iteration of the execution of (A–D). Let s_1, s_2, \dots be the sequence of strategies synthesized by (C) in each iteration. Let $\sigma_1, \sigma_2, \dots$ be the sequence of traces returned by Algorithm 5 used in (D). Let us write $\hat{\mathcal{M}}^i_{s_i}$ (resp., \mathcal{M}_{s_i}) for the composition of $\hat{\mathcal{M}}^i$ (resp., \mathcal{M}) with strategy s_i . After sufficiently many iterations of the loop, we can assume that $\text{compatible}(r, r') \iff \text{eqRow}(r, r')$ for any indexes r and r' in the observation table from Lemma 4.4.

Suppose the procedure (D) executes Algorithm 5 with $\hat{\mathcal{M}}^i$ and \mathcal{S} as inputs. If this execution results in \perp , then the loop (A–D) would terminate; therefore, assume that Algorithm 5 returns $\sigma := \sigma^- \cdot o$, where $\sigma^- \in (\Sigma^{\text{out}} \times \Sigma^{\text{in}})^*$ and $o \in \Sigma^{\text{out}}$. Let $p_{\hat{\mathcal{M}}^i, \sigma^-, o}$ be the probability of observing o in $\hat{\mathcal{M}}^i$ after σ^- and $p_{\mathcal{S}, \sigma^-, o}$ be $\frac{S(\sigma^-, o)}{S(\sigma^-)}$. Then, $|p_{\hat{\mathcal{M}}^i, \sigma^-, o} - p_{\mathcal{S}, \sigma^-, o}|$ is greater than $\sqrt{\frac{\ln(2) - \ln(\delta)}{2S(\sigma^-)}}$.

Table 2. Summary of the benchmarks: the number $|Q|$ of states, the size $|\Sigma^{\text{in}}|$ of the inputs, the size $|\Sigma^{\text{out}}|$ of the outputs, and the tested LTL formulas.

	$ Q $	$ \Sigma^{\text{in}} $	$ \Sigma^{\text{out}} $	LTL formulas
Slot	471	4	31	$\diamond_{[0,n]} \text{BAR3}$, with $n \in \{5, 8, 11, 14, 17\}$
SlotLimitedObs	471	4	12	$\diamond_{[0,n]} \text{BAR3}$, with $n \in \{5, 8, 11, 14, 17\}$
MQTT	62	9	50	$\diamond_{[0,n]} \text{crash}$, with $n \in \{5, 8, 11, 14, 17\}$
TCP	156	12	12	$\diamond_{[0,n]} \text{crash}$, with $n \in \{5, 8, 11, 14, 17\}$
GridWorldSmall	35	4	7	$\diamond_{[0,10]} \text{goal}$
GridWorldLarge	72	4	7	$\diamond_{[0,13]} \text{goal}$
SharedCoin	272	2	47	$\diamond_{[0,n]} \text{finished}$, with $n \in \{14, 20\}$
RandomGridWorld	{16, 64, 100, 144, 196}	4	7	$(-hole) \mathcal{U} \text{goal}$

Let $p_{\mathcal{M},\sigma^-,o}$ be the true probability of \mathcal{M} outputting o after observing σ^- . Since $p_{S,\sigma^-,o}$ is an unbiased estimation of $p_{\mathcal{M},\sigma^-,o}$, $\mathbb{P}(|p_{\mathcal{M},\sigma^-,o} - p_{S,\sigma^-,o}| < \epsilon) > 1 - 2e^{-2\epsilon^2 S(\sigma^-)}$ for an arbitrary positive real ϵ from the property of Chernoff bound in Section 4.2.1. Notice that $|p_{\hat{\mathcal{M}}^i,\sigma^-,o} - p_{S,\sigma^-,o}| \leq |p_{\hat{\mathcal{M}}^i,\sigma^-,o} - p_{\mathcal{M},\sigma^-,o}| + |p_{\mathcal{M},\sigma^-,o} - p_{S,\sigma^-,o}|$. Therefore, $\mathbb{P}(|p_{\hat{\mathcal{M}}^i,\sigma^-,o} - p_{\mathcal{M},\sigma^-,o}| > \sqrt{\frac{\ln(2) - \ln(\delta)}{2S(\sigma^-)}} - \epsilon) > 1 - 2e^{-2\epsilon^2 S(\sigma^-)}$. If we choose ϵ so that $\epsilon \in \left(\sqrt{\frac{\ln(2) - \ln(2\delta - \delta^2)}{2S(\sigma^-)}}, \sqrt{\frac{\ln(2) - \ln(\delta)}{2S(\sigma^-)}} \right)$, we have $\mathbb{P}(|p_{\hat{\mathcal{M}}^i,\sigma^-,o} - p_{\mathcal{M},\sigma^-,o}| > \sqrt{\frac{\ln(2) - \ln(\delta)}{2S(\sigma^-)}} - \epsilon) > (1 - \delta)^2$. This implies that $p_{\hat{\mathcal{M}}^i,\sigma^-,o}$ and $p_{\mathcal{M},\sigma^-,o}$ are indeed different with probability greater than $(1 - \delta)^2$. Being $p_{\hat{\mathcal{M}}^i,\sigma^-,o} \neq p_{\mathcal{M},\sigma^-,o}$ implies that, with the assumption that $\text{compatible}(r, r') \iff \text{eqRow}(r, r')$ for any r and r' , $\text{compatible}(\sigma^- \cdot o, r)$ does not hold for any r in the observation table. Therefore, adding $\sigma^- \cdot o$ to \mathcal{S} identifies a new state in $\hat{\mathcal{M}}^{i+1}$ with probability at least $(1 - \delta)^2$.

From the above discussion, there is an infinite sequence of MDPs $\hat{\mathcal{M}}^{i_1}, \hat{\mathcal{M}}^{i_2}, \dots$ with probability 1 such that $\hat{\mathcal{M}}^{i_{j+1}}$ has strictly more states than $\hat{\mathcal{M}}^{i_j}$. However, this contradicts Lemma 4.4 and Lemma 4.5; due to these lemmas, after sufficiently many iterations of (A–D), all $\hat{\mathcal{M}}^i$ should have less number of states than \mathcal{M} . \square

5 EXPERIMENTAL EVALUATION

We have developed a prototype tool implementing ProBBC in Python². We used AALpy [27] for active MDP learning and PRISM [19] for quantitative probabilistic model checking.

We conducted experiments to answer the following research questions.

- RQ1.** Does ProBBC produce a strategy close to the optimal one?
- RQ2.** Does ProBBC produce a better strategy than the existing method [4]?
- RQ3.** Is ProBBC robust to the observability of the output?
- RQ4.** Can ProBBC estimate a good strategy with a small sample size?
- RQ5.** Is ProBBC sensitive to the parameters?
- RQ6.** Does the strategy-guided comparison in Section 4.2 improve the performance of ProBBC?
- RQ7.** Is ProBBC scalable with respect to the system’s complexity?

5.1 Benchmarks

For the evaluation, we use eight benchmarks: Slot, SlotLimitedObs, MQTT, TCP, GridWorldSmall, GridWorldLarge, SharedCoin, and RandomGridWorld. Table 2 summarizes them. Each benchmark consists of an MDP and LTL formulas that are the same except for the timing parameter n . Slot and

²The artifact of the experiment is available on <https://doi.org/10.5281/zenodo.7997524>.

SlotLimitedObs are benchmarks with the same MDP except for observability. MQTT, TCP, and SharedCoin are benchmarks on communication protocols related to IoT applications. GridWorldSmall and GridWorldLarge are benchmarks on controller synthesis of a robot navigation system. Among the eight benchmarks, MQTT and TCP are benchmarks aiming at testing, whereas others are on controller synthesis. We use RandomGridWorld to answer RQ7 and the others to answer the other RQs.

Slot. The MDP in Slot is taken from [24], also used in [4, 33]. The slot machine has three reels, which are initially “blank”. The player can spin each reel independently. After a spin, each reel shows either an “apple” or a “bar”. The probability of having a “bar” decreases as the number of spins increases. A player is given a maximum number 5 of spins. When the game is over, a prize is given depending on the reel configuration. The player can also “stop” the game: With probability 0.5, the player obtains two extra spins (up to 5 in total); With probability 0.5, the player finishes the game and receives the award. Overall, there are four inputs: “reel1”, “reel2”, “reel3”, and “stop”. The outputs show the current reel configuration (during the game) and the received prize (at the end of the game). We use LTL formulas that are true if we obtain the prize *BAR3* for displaying “bar” at all three reels within specific numbers of total reels.

SlotLimitedObs. The LTL formulas in SlotLimitedObs are the same as Slot. The MDP in SlotLimitedObs is also the same as Slot except for the output: In SlotLimitedObs, only the reels displaying “bar” are observable; In Slot, the status of the reels is fully observable. For example, the MDP in SlotLimitedObs has the same output for “bar blank apple” and “bar apple blank”, which are distinguished in Slot. We use SlotLimitedObs primarily to answer RQ3.

MQTT. The MDP in MQTT is taken from [4], which models a broker in the MQTT protocol [1] with stochastic failures. Namely, the MDP models a broker crashing with probability 0.1. The MDP has nine inputs, e. g., for message types, and 50 outputs for the internal state of the broker. We use LTL formulas that are true if we have the output *crash* representing the stochastic failure within specific numbers of messages.

TCP. The MDP in TCP is also taken from [4], which models a TCP server [2] with stochastic failures. The probability of crashing is 0.05. We use LTL formulas that are true if we have the output *crash* representing the stochastic failure within specific numbers of messages.

GridWorldSmall. The MDP in GridWorldSmall is taken from [4, 33], which models a robot on a grid world with probabilistic error in its movement. For example, when a robot tries to move to the east, it may also move to the northeast or the southeast with small probabilities depending on the condition of the current position. The MDP has four inputs (“East”, “South”, “West”, and “North”) for the direction of a move and seven outputs (“Concrete”, “Grass”, “Wall”, “Mud”, “Pavement”, “Gravel”, and “Sand”) for the condition of the robot’s current position. We use the LTL formula $\diamond_{[0,10]}goal$ that is true if the robot reaches the goal position within ten steps.

GridWorldLarge. GridWorldLarge is a variant of GridWorldSmall, also from [4, 33]. The state space of GridWorldLarge is larger, and thus, the estimation of the MDP is harder. The inputs and the outputs are the same as GridWorldSmall. We use the LTL formula $\diamond_{[0,13]}goal$ that is true if the robot reaches the goal position within 13 steps.

SharedCoin. The MDP in SharedCoin is taken from [33], which models a randomized consensus protocol [6] with two processes. The MDP has two inputs for the process to execute and 47 outputs, for example, for the status of the coins. We used LTL formulas that are true if the algorithm finishes within specific numbers of messages.

RandomGridWorld. RandomGridWorld is our original benchmark inspired from GridWorldSmall and GridWorldLarge. RandomGridWorld consists of 25 randomly generated benchmarks: we fixed five sizes of the grid world and randomly generated five MDPs for each size. RandomGridWorld is primarily used to evaluate the scalability of ProBBBC. We use the LTL formula $(\neg \text{hole}) \mathcal{U} \text{goal}$ that is true if the robot reaches the goal position without entering the hole position.

5.2 Experiments

To answer RQ1–RQ4, we compared the performance of ProBBBC with ProbBlackReach [4]³. ProbBlackReach is another testing method for black-box MDPs based on MDP learning and probabilistic model checking. The main difference from ProBBBC is in the learning algorithm and the sampling method: ProbBlackReach passively learns an MDP using traces sampled by an ε -greedy algorithm. See Section 2 for a detailed comparison. To answer RQ5, we compared the performance of ProBBBC with different parameters. To answer RQ6, we compared the performance of ProBBBC with a variant (we call ProBBBC w/o str. comp.) of ProBBBC without the strategy-guided comparison, i. e., a variant such that the validation phase immediately starts equivalence testing by uniform sampling. As a ground truth, we also compute the optimal satisfaction probability with PRISM [19].

For each benchmark, we ran each method (i. e., ProBBBC with multiple parameters, ProbBlackReach, and ProBBBC w/o str. comp.) for 20 times. Since each method produces a strategy for each execution of probabilistic model checking, we have a bunch of strategies for each execution. For each strategy, we estimated the satisfaction probability of the LTL formula φ by the SUT \mathcal{M} using SMC with sample size 5,000.

We conducted all the experiments on a Google Cloud Platform c2-standard-4 instance (4 vCPU, 16GB RAM) running Debian 11 bullseye. We used Python 3.10.9, AALpy v.1.3.0, and PRISM version 4.7. The parameters in the validation phase are as follows: The sample size N in Algorithm 4 is 5000; The threshold Δ for Student’s t-testing (line 8 of Algorithm 4) is 0.025; The bound Δ' in the comparison of probabilities (line 8 of Algorithm 5) is 0.025. For the experiments to answer RQ5, we used all the combinations of $N \in \{2500, 5000, 10000\}$ and $\Delta = \Delta' \in \{0.01, 0.025, 0.05\}$, i. e., we used nine parameters in total. For the parameters of ProbBlackReach, we used the values in [4].

Table 3 summarizes the maximum satisfaction probabilities of each LTL formula estimated by ProBBBC (with the strategy-guided comparison) and ProbBlackReach as well as the true maximum probabilities computed by PRISM. Fig. 8 shows the number of steps on the SUT and the largest estimated maximum probabilities before the point for some benchmarks. Table 4 shows the mean of the maximum satisfaction probabilities of each LTL formula estimated by ProBBBC with various choices of parameters N and Δ . Table 5 summarizes the maximum satisfaction probabilities of each LTL formula estimated by ProBBBC without the strategy-guided comparison. Fig. 9 shows the number of the states and the mean execution time to estimate a probability that is larger than 97.5% of the true probability for RandomGridWorld.

5.3 RQ1: Quality of the estimated probabilities

In the columns “PRISM” and “ProBBBC/mean” of Table 3, we observe that for any LTL formula in our benchmarks, the satisfaction probability estimated by ProBBBC is close to the true probabilities on average. We note that the estimated probabilities can be slightly larger than the true probabilities due to the statistical error in the estimation with SMC.

Moreover, in the columns “PRISM” and “ProBBBC/min”, we observe that for some benchmarks (e. g., GridWorldSmall, GridWorldLarge, and SharedCoin), the satisfaction probability estimated

³We used the implementation of ProbBlackReach available on <https://github.com/mtappler/prob-black-reach> with additional codes for experiments.

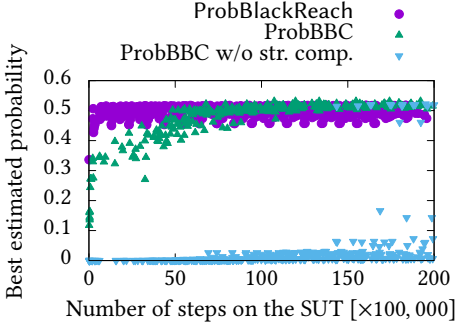
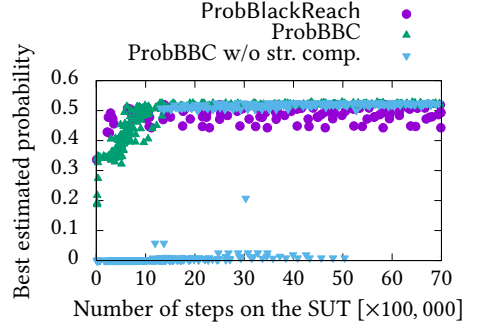
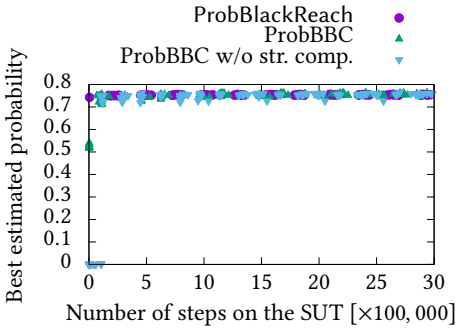
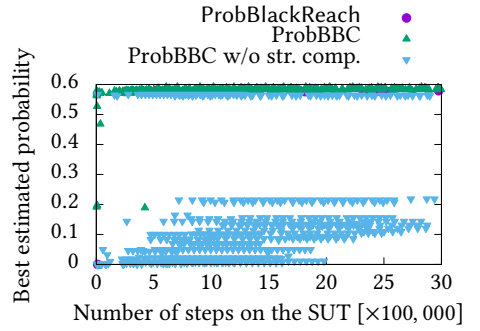
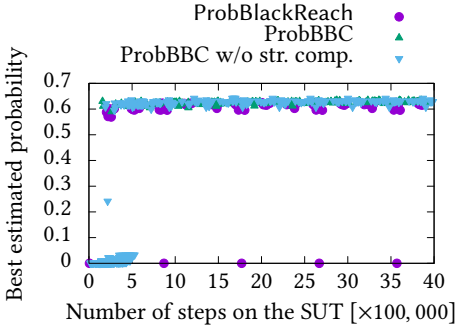
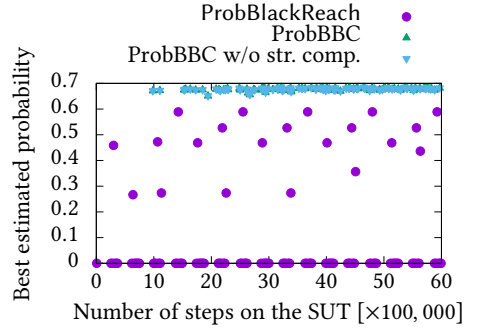
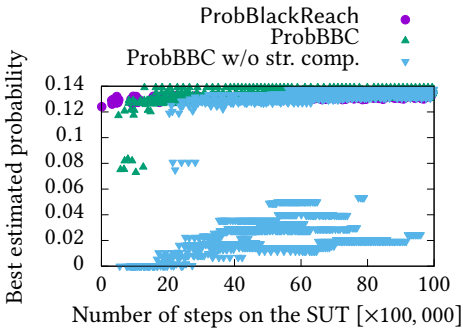
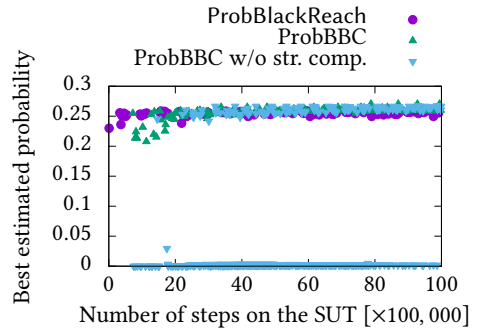
(a) Slot with $\varphi = \diamond_{[0,17]} \text{BAR3}$.(b) SlotLimitedObs with $\varphi = \diamond_{[0,17]} \text{BAR3}$.(c) MQTT with $\varphi = \diamond_{[0,14]} \text{crash}$.(d) TCP with $\varphi = \diamond_{[0,11]} \text{crash}$.(e) GridWorldSmall with $\varphi = \diamond_{[0,10]} \text{goal}$.(f) GridWorldLarge with $\varphi = \diamond_{[0,13]} \text{goal}$.(g) SharedCoin with $\varphi = \diamond_{[0,14]} \text{finished}$.(h) SharedCoin with $\varphi = \diamond_{[0,20]} \text{finished}$.

Fig. 8. Estimated probability after each execution step on the SUT.

Table 3. Summary of the estimated maximum satisfaction probabilities of φ after the number of steps displayed in the column “# of steps”. The column n shows the parameter n of the LTL formulas in Table 2. The column “PRISM” shows the true probabilities computed by PRISM. The columns “ProbBBC” and “ProbBlackReach” show the estimated probabilities computed by ProbBBC and ProbBlackReach, respectively. The columns “mean”, “std”, and “min” show the mean, the standard deviation, and the minimum value of the results. We highlight the cells if the mean of the estimated probabilities is larger than 97.5% of the true probability.

	n	PRISM	ProbBBC			ProbBlackReach			# of steps
		mean	std	min	mean	std	min		
Slot	5	8.23e-02	8.21e-02	3.22e-03	7.64e-02	8.23e-02	1.26e-03	7.91e-02	15,000,000
	8	3.32e-01	3.28e-01	6.73e-03	3.11e-01	3.21e-01	8.29e-03	2.97e-01	15,000,000
	11	4.60e-01	4.59e-01	8.57e-03	4.42e-01	4.40e-01	1.56e-02	4.01e-01	15,000,000
	14	4.99e-01	4.92e-01	1.21e-02	4.53e-01	4.82e-01	1.30e-02	4.41e-01	15,000,000
	17	5.10e-01	5.07e-01	9.64e-03	4.83e-01	4.80e-01	2.37e-02	4.32e-01	15,000,000
SlotLimitedObs	5	8.23e-02	8.14e-02	3.46e-03	7.58e-02	8.25e-02	1.62e-03	7.94e-02	7,000,000
	8	3.32e-01	3.31e-01	1.00e-02	3.17e-01	3.22e-01	5.80e-03	3.11e-01	7,000,000
	11	4.60e-01	4.61e-01	4.24e-03	4.52e-01	4.38e-01	1.44e-02	3.94e-01	7,000,000
	14	4.99e-01	4.95e-01	1.11e-02	4.62e-01	4.27e-01	4.92e-02	3.11e-01	7,000,000
	17	5.10e-01	5.09e-01	5.46e-03	5.01e-01	4.48e-01	4.15e-02	3.44e-01	7,000,000
MQTT	5	3.44e-01	3.43e-01	8.49e-03	3.31e-01	3.44e-01	3.19e-03	3.36e-01	3,000,000
	8	5.22e-01	5.22e-01	6.46e-03	5.11e-01	5.20e-01	3.03e-03	5.16e-01	3,000,000
	11	6.51e-01	6.45e-01	1.68e-02	6.08e-01	6.50e-01	2.46e-03	6.45e-01	3,000,000
	14	7.46e-01	7.45e-01	6.90e-03	7.31e-01	7.47e-01	3.17e-03	7.41e-01	3,000,000
	17	8.15e-01	8.08e-01	9.79e-03	7.88e-01	8.15e-01	2.20e-03	8.11e-01	3,000,000
TCP	5	1.90e-01	1.90e-01	5.38e-03	1.78e-01	1.90e-01	2.13e-03	1.86e-01	1,200,000
	8	4.10e-01	4.10e-01	6.63e-03	4.00e-01	4.10e-01	2.73e-03	4.05e-01	1,200,000
	11	5.70e-01	5.69e-01	5.51e-03	5.59e-01	5.69e-01	2.96e-03	5.63e-01	1,200,000
	14	6.86e-01	6.88e-01	6.71e-03	6.75e-01	6.84e-01	9.74e-03	6.45e-01	1,200,000
	17	7.71e-01	7.68e-01	1.22e-02	7.40e-01	7.71e-01	2.52e-03	7.66e-01	1,200,000
GridWorldSmall		6.18e-01	6.17e-01	9.57e-03	5.98e-01	5.69e-01	1.35e-01	0.00e+00	4,000,000
GridWorldLarge		6.71e-01	6.72e-01	6.88e-03	6.56e-01	6.83e-02	1.70e-01	0.00e+00	1,500,000
SharedCoin	14	1.25e-01	1.25e-01	5.18e-03	1.14e-01	1.09e-01	2.71e-02	6.15e-02	4,000,000
	20	2.50e-01	2.51e-01	5.16e-03	2.41e-01	2.18e-01	2.74e-02	1.55e-01	4,000,000

by ProbBBC is close to the true probabilities even in the worst case. This is likely because of the relatively small state space ($|Q| < 300$) and the small input size ($|\Sigma^{\text{in}}| \leq 4$) of the MDPs in these benchmarks, which make the maximum size of the observation table relatively small and the L^*_{MDP} algorithm identifies the target MDP with a mild number of queries.

These results suggest that the maximum satisfaction probabilities estimated by ProbBBC are usually close to the optimal value. Overall, we answer RQ1 as follows.

Answer to RQ1: ProbBBC usually estimates the maximum satisfaction probabilities close to the true one. For some benchmarks, the estimated probabilities are very close to the optimal value even in the worst case.

5.4 RQ2: Quality of the estimated probabilities compared with ProbBlackReach

In the columns “PRISM” and “ProbBlackReach/mean” of Table 3, we observe that ProbBlackReach often fails to estimate the maximum satisfaction probability close to the true one. For example, for 11 out of 24 LTL formulas, the maximum satisfaction probability of the formula estimated by ProbBlackReach is less than 97.5% of the true value (i. e., not highlighted in Table 3). This is likely because ProbBlackReach often gets stuck on a local optimum due to the greedy nature of its trace

sampling. Such a tendency is indeed observed in Fig. 8 for some benchmarks. In contrast, as we observe in Section 5.3, for all the LTL formulas, the probability estimated by ProbBBC is close to the true one. This is because the MDP learned by the L^*_{MDP} algorithm eventually converges to the SUT, as shown in [33]. Such convergence is also observed in Fig. 8.

In the column “ProbBlackReach/std” of Table 3, we observe that for GridWorldSmall and GridWorldLarge, the standard deviation of the probabilities estimated by ProbBlackReach is large (e. g., greater than 0.1). In the column “ProbBlackReach/min” of Table 3, we also observe that for GridWorldSmall and GridWorldLarge, the satisfaction probability estimated by ProbBlackReach can be 0. This is likely because synthesizing a reasonable strategy for a grid world environment requires a relatively precise estimation of the environment, which is not always done by ProbBlackReach. Such a tendency is more evident in GridWorldLarge, where the environment is larger, and its precise estimation is more challenging. In contrast, ProbBBC always synthesizes a reasonable strategy up to our observation. Overall, we answer RQ2 as follows.

Answer to RQ2: ProbBlackReach often fails to estimate a reasonable maximum satisfaction probability. Moreover, it sometimes fails to learn a reasonable strategy for GridWorldSmall and GridWorldLarge.

5.5 RQ3: Robustness with respect to the observability

In the rows “Slot” and “SlotLimitedObs” of Table 3, we observe that the probabilities estimated by SlotLimitedObs tend to be smaller with limited observations, especially when n is large. This is likely because ProbBlackReach samples the training data with an ϵ -greedy algorithm, and their variety is limited. When an MDP is learned, two candidate states are deemed identical if there is no clear counterexample, where outputs are used to compare the states. Therefore, to correctly distinguish the states, there must be sufficient variety in the training data to contain an evidence. The training data need more variety when many states have the same output, e. g., more variety is required in SlotLimitedObs than in Slot.

In contrast, the probabilities estimated by ProbBBC do not have such a deviation. This is likely because the L^*_{MDP} algorithm tries to cover various inputs to have sufficient information to estimate the transition probabilities. Moreover, ProbBBC tends to estimate *better* probabilities with limited observability. This is likely because of the following reason: The MDP of Slot has states that have different outputs but behave the same; For example, the states corresponding to “bar blank apple” and “bar apple blank” behave the same, but their outputs are different in Slot; In the MDP of SlotLimitedObs these states have the same outputs, and they are deemed identical in MDP learning; Since the target MDP under learning is virtually smaller, the L^*_{MDP} algorithm can easily converge to the optimal one; Therefore, ProbBBC can perform better with limited observability. We indeed observe that the resulting MDPs tend to be smaller for SlotLimitedObs than Slot (about 50 vs. 160 states on average). Overall, we answer RQ3 as follows.

Answer to RQ3: ProbBBC estimates a near-optimal satisfaction probability even if the observability in the SUT is limited, whereas ProbBlackReach often fails to estimate it in such a situation. Moreover, ProbBBC often estimates a better probability when the observability is limited because the state space of the SUT can be virtually smaller.

5.6 RQ4: Efficiency of the estimation

In Figs. 8a, 8b, 8e and 8f, we observe that the probabilities estimated by ProbBlackReach often remain suboptimal, as discussed in Section 5.4, whereas those estimated by ProbBBC usually converge (with some exceptions, e. g., some executions for MQTT with $n = 14$ shown in Fig. 8c). In

Table 4. Mean of the estimated maximum satisfaction probabilities of φ after the number of steps displayed in the column “# of steps” of Table 3. The column n shows the parameter n of the LTL formulas in Table 2. We highlight the cells whose values are larger than 97.5% of the true probability in Table 3.

	n	$N = 2500$			$N = 5000$			$N = 10000$		
		$\Delta = 0.01$	$\Delta = 0.025$	$\Delta = 0.05$	$\Delta = 0.01$	$\Delta = 0.05$	$\Delta = 0.01$	$\Delta = 0.025$	$\Delta = 0.05$	
Slot	5	8.19e-02	8.15e-02	8.21e-02	8.23e-02	8.19e-02	8.29e-02	8.22e-02	8.11e-02	
	8	3.33e-01	3.34e-01	3.32e-01	3.31e-01	3.31e-01	3.26e-01	3.30e-01	3.31e-01	
	11	4.58e-01	4.58e-01	4.58e-01	4.57e-01	4.59e-01	4.57e-01	4.55e-01	4.59e-01	
	14	4.94e-01	4.94e-01	4.97e-01	4.92e-01	4.95e-01	4.94e-01	4.91e-01	4.95e-01	
	17	5.05e-01	5.01e-01	5.04e-01	5.08e-01	5.01e-01	5.05e-01	5.02e-01	5.06e-01	
SlotLimitedObs	5	8.07e-02	8.07e-02	8.07e-02	8.07e-02	8.07e-02	8.07e-02	8.07e-02	8.07e-02	
	8	3.33e-01	3.33e-01	3.33e-01	3.33e-01	3.33e-01	3.33e-01	3.33e-01	3.33e-01	
	11	4.60e-01	4.60e-01	4.60e-01	4.60e-01	4.60e-01	4.60e-01	4.60e-01	4.60e-01	
	14	4.97e-01	4.97e-01	4.97e-01	4.97e-01	4.97e-01	4.97e-01	4.97e-01	4.97e-01	
	17	5.10e-01	5.10e-01	5.10e-01	5.10e-01	5.10e-01	5.10e-01	5.10e-01	5.10e-01	
MQTT	5	3.43e-01	3.43e-01	3.43e-01	3.43e-01	3.43e-01	3.43e-01	3.43e-01	3.43e-01	
	8	5.22e-01	5.22e-01	5.22e-01	5.22e-01	5.22e-01	5.22e-01	5.22e-01	5.22e-01	
	11	6.45e-01	6.45e-01	6.45e-01	6.45e-01	6.45e-01	6.45e-01	6.45e-01	6.45e-01	
	14	7.45e-01	7.45e-01	7.45e-01	7.45e-01	7.45e-01	7.45e-01	7.45e-01	7.45e-01	
	17	8.08e-01	8.08e-01	8.08e-01	8.08e-01	8.08e-01	8.08e-01	8.08e-01	8.08e-01	
TCP	5	1.90e-01	1.91e-01	1.90e-01	1.90e-01	1.90e-01	1.88e-01	1.90e-01	1.91e-01	
	8	4.11e-01	4.08e-01	4.05e-01	4.09e-01	4.09e-01	4.09e-01	4.09e-01	4.07e-01	
	11	5.61e-01	5.62e-01	5.69e-01	5.69e-01	5.58e-01	5.68e-01	5.65e-01	5.67e-01	
	14	6.76e-01	6.74e-01	6.84e-01	6.80e-01	6.76e-01	6.81e-01	6.83e-01	6.79e-01	
	17	7.64e-01	7.60e-01	7.68e-01	7.68e-01	7.67e-01	7.65e-01	7.66e-01	7.61e-01	
GridWorldSmall		6.14e-01	6.14e-01	6.15e-01	6.14e-01	6.17e-01	6.18e-01	6.17e-01	6.18e-01	
GridWorldLarge		6.73e-01	6.71e-01	6.68e-01	6.70e-01	6.70e-01	6.69e-01	6.70e-01	6.69e-01	
SharedCoin	14	1.24e-01	1.24e-01	1.24e-01	1.25e-01	1.24e-01	1.26e-01	1.24e-01	1.24e-01	
	20	2.50e-01	2.50e-01	2.46e-01	2.52e-01	2.49e-01	2.53e-01	2.49e-01	2.50e-01	

contrast, in Figs. 8a, 8b, 8g and 8h, we observe that the initial rising up of the probabilities estimated by ProbBBC is slower than that by ProbBlackReach. This is also likely because of the use of by an ε -greedy algorithm in the training data construction: The ε -greedy algorithm samples the traces deemed to increase the satisfaction probability of the given LTL formula. If such traces indeed increase the satisfaction probability, the estimated probability increases with a small number of traces. However, this is at the cost of the quality of the final estimation, and the delay of the initial rising up is not very large except for Fig. 8a. Overall, we answer RQ4 as follows.

Answer to RQ4: ProbBlackReach is often faster to start up than ProbBBC at the cost of the quality of the estimated probability. Moreover, the delay of the initial rising up is usually not large.

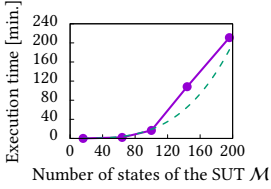
5.7 RQ5: Sensitivity to the parameters

In Table 4, we observe that for any combination of N and Δ , and for any LTL formula in our benchmarks, the satisfaction probability estimated by ProbBBC is close to the true probabilities on average. If N is small or Δ is large, the probability of the strategy-guided comparison to return a false witness (i. e., a trace σ whose occurrence probability is supposed to be different between \mathcal{M} and $\hat{\mathcal{M}}$ but not) is also large. However, if the probability to return a false witness is not too large, the learned MDP converges in a reasonable number of steps. Overall, we answer RQ5 as follows.

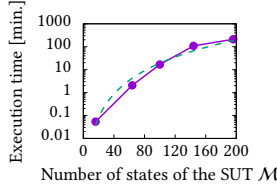
Answer to RQ5: The validation phase in ProbBBC is not much parameter sensitive.

Table 5. Summary of the maximum satisfaction probabilities of φ estimated by ProbBBC without the strategy-guided comparison after the number of steps displayed in the column “# of steps” of Table 3. The column n shows the parameter n of the LTL formulas in Table 2. The columns “mean”, “std”, and “min” show the mean, the standard deviation, and the minimum value of the results. We highlight the cells if the mean of the estimated probabilities is larger than 97.5% of the true probability in Table 3.

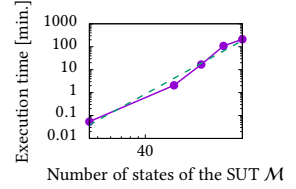
	n	mean	std	min		n	mean	std	min	
Slot	5	8.12e-02	4.67e-03	7.50e-02	MQTT	11	6.45e-01	1.42e-02	6.16e-01	
	8	2.43e-01	1.48e-01	0.00e+00		14	7.43e-01	9.48e-03	7.16e-01	
	11	4.98e-02	1.43e-01	0.00e+00		17	8.05e-01	1.52e-02	7.82e-01	
	14	4.85e-03	7.53e-03	0.00e+00		TCP	5	1.89e-01	5.46e-03	1.77e-01
	17	3.03e-02	1.12e-01	0.00e+00			8	1.66e-02	3.92e-02	0.00e+00
SlotLimitedObs	5	8.22e-02	4.03e-03	7.42e-02	11		1.42e-02	2.61e-02	0.00e+00	
	8	3.34e-01	8.15e-03	3.21e-01	14		1.27e-02	1.79e-02	0.00e+00	
	11	4.34e-01	1.02e-01	2.00e-04	17	1.85e-02	2.68e-02	2.00e-04		
	14	4.24e-01	1.81e-01	8.00e-04	GridWorldSmall	6.16e-01	1.14e-02	5.79e-01		
MQTT	17	4.84e-01	1.13e-01	4.40e-03	GridWorldLarge	1.00e-01	2.43e-01	0.00e+00		
	5	3.43e-01	5.62e-03	3.26e-01	SharedCoin	14	7.64e-02	5.88e-02	1.00e-03	
	8	5.20e-01	1.33e-02	4.70e-01		20	1.13e-01	1.28e-01	0.00e+00	



(a) linear-linear plot



(b) linear-log plot



(c) log-log plot

Fig. 9. State size of the SUT and the mean execution time to estimate a probability larger than 97.5% of the true one, per SUT size in RandomGridWorld. The dashed curve is $y = 1.99 \times 10^{-4} x^{3.38}$, obtained by regression.

5.8 RQ6: Effect of the strategy-guided comparison

In the column “mean” of Table 5, we observe that ProbBBC often fails to estimate the maximum satisfaction probability close to the true one if we do not use the strategy-guided comparison. In Fig. 8, we observe that the initial rising up of the probabilities estimated by ProbBBC w/o str. comp. is often slower than that by ProbBBC. These tendencies are likely because, without the strategy-guided comparison, the validation phase in ProbBBC is solely by uniform random sampling. In contrast, if we use the strategy-guided comparison, the validation phase can focus on a part of the MDP relevant to maximize the satisfaction probability of the given LTL formula, which tends to accelerate the probability estimation. We note that, unlike ProbBlackReach, ProbBBC remains correct even without the strategy-guided comparison thanks to the convergence of L^*_{MDP} . Overall, we answer RQ6 as follows.

Answer to RQ6: The strategy-guided comparison in Section 4.2 usually allows to generate a controller achieving high satisfaction probability with fewer steps on the SUT.

5.9 RQ7: Scalability to the system complexity

In Fig. 9, we observe that, on average, ProbBBC can find a near-optimal controller for SUTs with 196 states within around 3.5 hours. In Fig. 9, we also observe that the time to find a near-optimal controller nearly follows a polynomial curve, which coincides with the polynomial complexity of L^* [5]. Therefore, we answer RQ7 as follows.

Answer to RQ7: Experiments suggest that ProbBBC can generate near-optimal controllers in polynomial time with respect to system size.

In the execution log, we find that ProbBBC often takes quite a long time to make the observation table satisfy the requirements to construct an MDP. These requirements prevent us from generating MDPs different from the target one, for example, if the transition function is inconsistent. Nevertheless, It is a possible future direction to construct an MDP even if the observation table does not satisfy some of the requirements, e. g., using passive MDP learning [23, 24]. Such an MDP is imprecise but may be still useful for generating a near-optimal strategy.

6 CONCLUSIONS AND FUTURE WORK

We propose *probabilistic black-box checking (ProbBBC)*, an extension of BBC for stochastic systems, by combining active MDP learning, probabilistic model checking, and statistical hypothesis testing. We conducted experiments to evaluate the performance of ProbBBC using benchmarks related to CPS or IoT scenarios. Our experiment results suggest that ProbBBC outperforms an existing approach [4], especially when the SUT is complex (e. g., GridWorldLarge) or the observability is limited (e. g., SlotLimitedObs).

As formulas of specifications, ProbBBC uses safety LTL. Using other logic, such as probabilistic computation tree logic (PCTL) [7], or PCTL* [8], is one of the future directions. Another future direction is conducting a case study with larger and more practical benchmarks.

ACKNOWLEDGMENTS

This work was partially supported by JST CREST Grant No. JPMJCR2012, JST PRESTO Grant No. JPMJPR22CA, JST ACT-X Grant No. JPMJAX200U, and JSPS KAKENHI Grant No. 22K17873 & 19H04084.

REFERENCES

- [1] [n. d.]. MQTT Version 3.1.1. Edited by Andrew Banks and Rahul Gupta. 29 October 2014. OASIS Standard. <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html>. Latest version: <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>.
- [2] [n. d.]. TCP models. <https://gitlab.science.ru.nl/pfiteraubrosteian/tcp-learner/tree/cav-aec/models>. Accessed 20 Jan 2023.
- [3] Gul Agha and Karl Palmkog. 2018. A Survey of Statistical Model Checking. *ACM Trans. Model. Comput. Simul.* 28, 1 (2018), 6:1–6:39.
- [4] Bernhard K. Aichernig and Martin Tappler. 2019. Probabilistic black-box reachability checking (extended version). *Formal Methods Syst. Des.* 54, 3 (2019), 416–448.
- [5] Dana Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75, 2 (1987), 87–106.
- [6] James Aspnes and Maurice Herlihy. 1990. Fast Randomized Consensus Using Shared Memory. *J. Algorithms* 11, 3 (1990), 441–461.
- [7] Christel Baier, Luca de Alfaro, Vojtech Forejt, and Marta Kwiatkowska. 2018. Model Checking Probabilistic Systems. In *Handbook of Model Checking*, Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem (Eds.). Springer, 963–999.
- [8] Andrea Bianco and Luca de Alfaro. 1995. Model Checking of Probabilistic and Nondeterministic Systems. In *Foundations of Software Technology and Theoretical Computer Science, 15th Conference, Proceedings (Lecture Notes in Computer Science, Vol. 1026)*, P. S. Thiagarajan (Ed.). Springer, 499–513.
- [9] Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Kretínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. 2014. Verification of Markov Decision Processes Using Learning Algorithms. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Proceedings (Lecture Notes in Computer Science, Vol. 8837)*, Franck Cassez and Jean-François Raskin (Eds.). Springer, 98–114.
- [10] Carlos Canal and Akram Idani (Eds.). 2015. *Software Engineering and Formal Methods - SEFM 2014 Collocated Workshops: HOFM, SAFOME, OpenCert, MoKMaSD, WS-FMDS, Revised Selected Papers*. Lecture Notes in Computer Science, Vol. 8938. Springer.

- [11] Tsun S. Chow. 1978. Testing Software Design Modeled by Finite-State Machines. *IEEE Trans. Software Eng.* 4, 3 (1978), 178–187. <https://doi.org/10.1109/TSE.1978.231496>
- [12] Pedro R. D’Argenio, Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. 2015. Smart sampling for lightweight verification of Markov decision processes. *Int. J. Softw. Tools Technol. Transf.* 17, 4 (2015), 469–484.
- [13] Edith Elkind, Blaise Genest, Doron A. Peled, and Hongyang Qu. 2006. Grey-Box Checking. In *Formal Techniques for Networked and Distributed Systems - FORTE 2006, 26th IFIP WG 6.1 International Conference (Lecture Notes in Computer Science, Vol. 4229)*, Elie Najm, Jean-François Pradat-Peyre, and Véronique Donzeau-Gouge (Eds.). Springer, 420–435.
- [14] Vojtech Forejt, Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. Automated Verification Techniques for Probabilistic Systems. In *Formal Methods for Eternal Networked Software Systems - 11th International School on Formal Methods for the Design of Computer, Communication and Software Systems, SFM 2011. Advanced Lectures (Lecture Notes in Computer Science, Vol. 6659)*, Marco Bernardo and Valérie Issarny (Eds.). Springer, 53–113.
- [15] Alex Groce, Doron A. Peled, and Mihalis Yannakakis. 2002. Adaptive Model Checking. In *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Proceedings (Lecture Notes in Computer Science, Vol. 2280)*, Joost-Pieter Katoen and Perdita Stevens (Eds.). Springer, 357–370.
- [16] Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association* 58, 301 (1963), 13–30.
- [17] Damir Kalpic, Nikica Hlupic, and Miodrag Lovric. 2011. Student’s t-Tests. In *International Encyclopedia of Statistical Science*, Miodrag Lovric (Ed.). Springer, 1559–1563.
- [18] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2007. Stochastic Model Checking. In *Formal Methods for Performance Evaluation, 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007. Advanced Lectures (Lecture Notes in Computer Science, Vol. 4486)*, Marco Bernardo and Jane Hillston (Eds.). Springer, 220–270.
- [19] Marta Z. Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6806)*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer, 585–591.
- [20] Marta Z. Kwiatkowska and David Parker. 2013. Automated Verification and Strategy Synthesis for Probabilistic Systems. In *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013. Proceedings (Lecture Notes in Computer Science, Vol. 8172)*, Dang Van Hung and Mizuhito Ogawa (Eds.). Springer, 5–22.
- [21] Kim G. Larsen and Axel Legay. 2016. Statistical Model Checking: Past, Present, and Future. In *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISOFA 2016, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 9952)*, Tiziana Margaria and Bernhard Steffen (Eds.). 3–15.
- [22] Axel Legay, Sean Sedwards, and Louis-Marie Traonouez. 2014. Scalable Verification of Markov Decision Processes, See [10], 350–362.
- [23] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. 2012. Learning Markov Decision Processes for Model Checking. In *Proceedings Quantities in Formal Methods, QFM 2012 (EPTCS, Vol. 103)*, Uli Fahrenberg, Axel Legay, and Claus R. Thrane (Eds.). 49–63.
- [24] Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen. 2016. Learning deterministic probabilistic automata from a model checking perspective. *Mach. Learn.* 105, 2 (2016), 255–299.
- [25] Jeroen Meijer and Jaco van de Pol. 2019. Sound black-box checking in the LearnLib. *Innov. Syst. Softw. Eng.* 15, 3-4 (2019), 267–287.
- [26] Michael Mitzenmacher and Eli Upfal. 2017. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis* (2nd ed.). Cambridge University Press, USA.
- [27] Edi Muskardin, Bernhard K. Aichernig, Ingo Pill, Andrea Pferscher, and Martin Tappler. 2021. AALpy: An Active Automata Learning Library. In *Automated Technology for Verification and Analysis - 19th International Symposium, ATVA 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12971)*, Zhe Hou and Vijay Ganesh (Eds.). Springer, 67–73.
- [28] Masashi Okamoto. 1959. Some inequalities relating to the partial sum of binomial probabilities. *AnnInstStat Math* 10 (1959), 29–35.
- [29] Doron A. Peled, Moshe Y. Vardi, and Mihalis Yannakakis. 1999. Black Box Checking. In *Formal Methods for Protocol Engineering and Distributed Systems, FORTE XII / PSTV XIX’99, IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX) (IFIP Conference Proceedings, Vol. 156)*, Jianping Wu, Samuel T. Chanson, and Qiang Gao (Eds.). Kluwer, 225–240.
- [30] Amir Pnueli. 1977. The Temporal Logic of Programs. In *18th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 46–57.
- [31] Junya Shijubo, Masaki Waga, and Kohei Suenaga. 2021. Efficient Black-Box Checking via Model Checking with Strengthened Specifications. In *Runtime Verification - 21st International Conference, RV 2021, Proceedings (Lecture Notes*

- in *Computer Science, Vol. 12974*), Lu Feng and Dana Fisman (Eds.). Springer, 100–120.
- [32] Alexander L. Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L. Littman. 2006. PAC model-free reinforcement learning. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006) (ACM International Conference Proceeding Series, Vol. 148)*, William W. Cohen and Andrew W. Moore (Eds.). ACM, 881–888.
 - [33] Martin Tappler, Bernhard K. Aichernig, Giovanni Bacci, Maria Eichlseder, and Kim G. Larsen. 2021. L^{*}-based learning of Markov decision processes (extended version). *Formal Aspects Comput.* 33, 4-5 (2021), 575–615.
 - [34] Moshe Y. Vardi. 1995. An Automata-Theoretic Approach to Linear Temporal Logic. In *Logics for Concurrency - Structure versus Automata (8th Banff Higher Order Workshop, Banff, Canada, August 27 - September 3, 1995, Proceedings) (Lecture Notes in Computer Science, Vol. 1043)*, Faron Moller and Graham M. Birtwistle (Eds.). Springer, 238–266.
 - [35] Masaki Waga. 2020. Falsification of cyber-physical systems with robustness-guided black-box checking. In *HSCC '20: 23rd ACM International Conference on Hybrid Systems: Computation and Control*, Aaron D. Ames, Sanjit A. Seshia, and Jyotirmoy Deshmukh (Eds.). ACM, 11:1–11:13.
 - [36] Christopher J. C. H. Watkins and Peter Dayan. 1992. Technical Note Q-Learning. *Mach. Learn.* 8 (1992), 279–292.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009