*-based learning of Markov decision processes

Extended version

Tappler, Martin; Aichernig, Bernhard K.; Bacci, Giovanni; Eichlseder, Maria; Larsen, Kim Guldstrand

Published in:
Formal Aspects of Computing

DOI (link to publication from Publisher):
10.1007/s00165-021-00536-5

Creative Commons License
CC BY 4.0

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

Link to publication from Aalborg University

Citation for published version (APA):
Tappler, M., Aichernig, B. K., Bacci, G., Eichlseder, M., & Larsen, K. G. (2021). *-based learning of Markov decision processes: Extended version. *Formal Aspects of Computing*, *33*(4-5), 575-615. https://doi.org/10.1007/s00165-021-00536-5

https://doi.org/10.1007/s00165-021-00536-5
The Author(s) © 2021
Formal Aspects of Computing (2021) 33: 575–615

Formal Aspects of Computing

L^*-based learning of Markov decision processes (extended version)

Martin Tappler[1,2], Bernhard K. Aichernig[1], Giovanni Bacci[4], Maria Eichlseder[3], and Kim G. Larsen[4]

[1] Institute of Software Technology, Graz University of Technology, Graz, Austria,
[2] Schaffhausen Institute of Technology, Schaffhausen, Switzerland,
[3] Institute of Applied Information Processing and Communications, Graz University of Technology, Graz, Austria
[4] Department of Computer Science, Aalborg University, Aalborg, Denmark

Abstract. Automata learning techniques automatically generate system models from test observations. Typically, these techniques fall into two categories: passive and active. On the one hand, passive learning assumes no interaction with the system under learning and uses a predetermined training set, e.g., system logs. On the other hand, active learning techniques collect training data by actively querying the system under learning, allowing one to steer the discovery of meaningful information about the system under learning leading to effective learning strategies. A notable example of active learning technique for regular languages is Angluin's L^*-algorithm. The L^*-algorithm describes the strategy of a student who learns the minimal deterministic finite automaton of an unknown regular language L by asking a succinct number of queries to a teacher who knows L.

In this work, we study L^*-based learning of deterministic Markov decision processes, a class of Markov decision processes where an observation following an action uniquely determines a successor state. For this purpose, we first assume an ideal setting with a teacher who provides perfect information to the student. Then, we relax this assumption and present a novel learning algorithm that collects information by sampling execution traces of the system via testing.

Experiments performed on an implementation of our sampling-based algorithm suggest that our method achieves better accuracy than state-of-the-art passive learning techniques using the same amount of test observations. In contrast to existing learning algorithms which assume a predefined number of states, our algorithm learns the complete model structure including the state space.

Keywords: Model inference, Active automata learning, Markov decision processes

Correspondence to: Bernhard K. Aichernig, e-mail: aichernig@ist.tugraz.at

1. Introduction

Automata learning automatically generates models from system observations such as test logs. Hence, it enables model-based verification for black-box software systems [HS18, AMM$^+$18], e.g. via model checking. Automata learning techniques generally fall into two categories: passive and active learning. Passive algorithms take a given sample of system traces as input and generate models consistent with the sample. The quality and comprehensiveness of learned models therefore largely depend on the given sample. In contrast, active algorithms actively query the system under learning (SUL) to sample system traces. This enables to steer the trace generation towards parts of the SUL's state space that have not been thoroughly covered, potentially finding yet unknown aspects of the SUL.

Many active automata learning algorithms are based on Angluin's L^* algorithm [Ang87]. It was originally proposed for learning deterministic finite automata (DFA) accepting regular languages and later applied to learn models of reactive systems, by considering that system traces form regular languages [HNS03]. L^* has been extended to formalisms better suited for modelling reactive systems such as Mealy machines [MNRS04, SG09] and extended finite state-machines [CHJS16]. Most L^*-based algorithms, however, target deterministic models, with the exceptions of algorithms for non-deterministic Mealy machines [KT14] and non-deterministic input-output transition systems [VT15]. Both techniques are based on testing, but abstract away the observed frequency of events, thus they do not use all available information.

Here, we present an L^*-based approach for learning models of stochastic systems with transitions that happen with some probability depending on non-deterministically chosen inputs. More concretely, we learn deterministic Markov decision processes (MDPs), like IoAlergia [MCJ$^+$12, MCJ$^+$16], a state-of-the-art passive learning algorithm. *Deterministic MDPs enjoy the property that a given observation following a specific action uniquely determines a successor state.* MDPs are commonly used to model randomised distributed algorithms [BK08], e.g. in protocol verification [KNP08, NS06].

Overview. In this article, we present L^*-based learning of MDPs from traces of stochastic black-box systems. For this purpose, we developed two learning algorithms. The first algorithm takes an ideal view assuming perfect knowledge about the exact distribution of system traces. The second algorithm relaxes this assumption, by sampling system traces to estimate their distribution. We refer to the former as *exact learning algorithm* $L^*_{\text{MDP}^e}$ and to the latter as *sampling-based learning algorithm* L^*_{MDP}. We implemented L^*_{MDP} and evaluated it by comparing it to IoAlergia [MCJ$^+$12, MCJ$^+$16]. The experiments presented in this article showed favourable performance of L^*_{MDP}. It produced more accurate models than IoAlergia given approximately the same amount of data. Hence, the answer to our motivating question stated above is positive. Active learning can improve accuracy, compared to passive learning. To the best of our knowledge, L^*_{MDP} is the first L^*-based learning algorithm for MDPs that can be implemented via testing. Our contributions in this context span the algorithmic development of two learning algorithms, their analysis with respect to convergence and the implementation as well as the evaluation of learning algorithms.

This article is an extended version of our conference paper "L^*-based Learning of Markov Decision Processes" presented at the 23rd Symposium on Formal Methods (FM 2019) [TAB$^+$19]. The present article includes the following additional material:

- a thorough description of the sampling-based teacher implementation for L^*_{MDP},
- an extended evaluation including additional experiments,
- we provide proofs for the exact algorithm $L^*_{\text{MDP}^e}$ and show that the model learned by L^*_{MDP} converges in the limit to an MDP isomorphic to the canonical MDP representing the SUL.

Parts of this article have been included in the doctoral thesis of one of the authors [Tap19].

Synopsis. The paper is structured as follows. Section 2 introduces the notation used in the rest of the paper as well as some preliminary notions of measure theory and MDPs. In Sect. 3 we consider different types of observation sequences of MDPs and provide a semantic characterisation of MDPs in terms of observation sequences. Section 4 presents the $L^*_{\text{MDP}^e}$ algorithm: an L^*-based technique that learns an *exact* and minimal MDP modelling the SUL under the assumption that the teacher has perfect knowledge of the probability distribution on traces. Then, in Sect. 5 we drop the latter assumption and describe the L^*_{MDP} algorithm, a sampling-based extension of $L^*_{\text{MDP}^e}$. In Sect. 6 we show that the model learned by L^*_{MDP} converges to true canonical model in the limit of the number of sampled traces.

Table 1. Notational conventions

Notation	Condition	Meaning
S^*	S is a non-empty set	arbitrary-length sequences of elements in S
S^l		sequences of elements in S with length l
$s \cdot s'$	$s, s' \in S^*$	concatenation of s and s'
ϵ		empty sequence $\epsilon \in S^*$
$\mid s \mid$		length of sequence s
$\mid S \mid$		cardinality of set S
e	$e \in S$	sequence $e \in S^*$ of length one (elements lifted to sequences)
$s[i]$	$s = s_1 \cdots s_n \in S^*$	$s[i]$ is the ith element s_i ... one-based indexed access
$s[< i]$		$s[< i] = s_1 \cdots s_{<i}$ ... subsequence of s with indexes $j < i$
$s[\leq i]$		$s[\leq i] = s_1 \cdots s_{\leq i}$ ... subsequence of s with indexes $j \leq i$
$s[\geq i]$		$s[\geq i] = s_{\geq i} \cdots s_n$ ... subsequence of s with indexes $j \geq i$
$s[> i]$		$s[> i] = s_{>i} \cdots s_n$ ... subsequence of s with indexes $j > i$
$s \ll s'$		s is a prefix of s', i.e. $\exists\, t \in S^* : s \cdot t = s'$
$s \gg s'$		s is a suffix of s', i.e. $\exists\, t \in S^* : t \cdot s = s'$
$A \cdot B$	$A, B \subseteq S^*$	pair-wise concatenation $A \cdot B = \{a \cdot b \mid a \in A, b \in B\}$
prefixes(A)		prefixes of sequences in A $\{a' \in S^* \mid \exists\, a \in A : a' \ll a\}$
suffixes(A)		suffixes of sequences in A $\{a' \in S^* \mid \exists\, a \in A : a' \gg a\}$
$\mathcal{P}(C)$	C is a set	power set of C
$\mathbb{P}(a)$ and $p(a)$	a is an event	probability of a
$\mathcal{T}(e)$	$\mathcal{T}$ is a multiset	multiplicity of e in $\mathcal{T}$

In Sect. 7 we perform an empirical analysis of our sampling-based algorithm and compare it with IoAlergia [MCJ+12, MCJ+16]. Section 8 reviews related work. We summarise our results in Sect. 9 and conclude in Sect. 10.

2. Preliminaries

We introduce background material similar to our previous work on stochastic automata learning [AT17b, AT19b, TAB+19], partly following the presentation by Mao et al. [MCJ+16] and Forejt et al. [FKNP11]. An important difference in the latter works [MCJ+16, FKNP11] is that we mainly consider finite traces and finite paths, as we learn from finite traces that we sample via testing. In contrast to that, model checking mostly considers properties defined over infinite sequences [FKNP11]. We also use slightly different notation.

2.1. Notation

Table 1 introduces some notational conventions that we use throughout this article. The left column introduces some notation, the middle column specifies conditions on variables used in the table and the right column describes the corresponding meaning of the notation. Note that conditions in Table 1 are cumulative, i.e., conditions introduced in a row also hold in the rows below. In addition to that, we introduce some terminology and auxiliary functions below.

2.1.1. Terminology

A set of sequences $A \subseteq S^*$ is prefix-closed, if it contains all prefixes of all sequences in A, i.e., $A = \mathit{prefixes}(A)$. Analogously, A is suffix-closed if it contains all suffixes of all sequences in A, i.e., $A = \mathit{suffixes}(A)$.

2.1.2. Auxiliary functions

In this article, we will use three random functions to perform probabilistic choices, for instance, to decide whether test-case generation should be stopped. The function *coinFlip* implements a biased coin flip for performing a binary probabilistic choice. It is defined for $p \in [0, 1] \cap \mathbb{Q}$ by $\mathbb{P}(\mathit{coinFlip}(p) = \mathit{true}) = p$ and $\mathbb{P}(\mathit{coinFlip}(p) = \mathit{false}) = 1 - p$. The function *rSel* selects a single sample e from a set S according to a uniform distribution, that is, $\forall\, e \in S \colon \mathbb{P}(\mathit{rSel}(S) = e) = \frac{1}{|S|}$. The function *rSeq* takes a set S and a length bound $b \in \mathbb{N}$ to create a random sequence s of elements in S. The length $l \leq b$ of s is chosen uniformly from $[0\,..\,b]$ and each element of s is chosen via $\mathit{rSel}(S)$.

2.2. Probability distributions and random functions

Given a set S, we denote by $\mathit{Dist}(S)$ the set of probability distributions over S, thus for all μ in $\mathit{Dist}(S)$ we have $\mu \colon S \to [0, 1] \cap \mathbb{Q}$ such that $\sum_{s \in S} \mu(s) = 1$. For simplicity, we consider only rational probabilities. Distributions μ considered in this article may be partial functions, in which case we implicitly set $\mu(e) = 0$ if μ is not defined for e. For $A \subseteq S$, $\mathbf{1}_A$ denotes the indicator function of A, i.e. $\mathbf{1}_A(e) = 1$ if $e \in A$ and $\mathbf{1}_A(e) = 0$ otherwise. Hence, $\mathbf{1}_{\{e\}}$ for $e \in S$ is the probability distribution assigning probability 1 to e.

2.3. String notation

Let I and O be sets of input and output symbols. As explained below, outputs label states and inputs label edges in MDPs, like in Moore machines. Hence, traces of MDPs are usually alternating sequences of outputs and inputs that start and end with an output, because paths corresponding to traces start and end in a state. A trace is therefore an input/output string s which is an alternating sequence of inputs and outputs, starting with an output, i.e. $s \in O \times (I \times O)^*$. We extend the general notational conventions for sequences introduced above, like concatenation and indexed access, to such input/output string. The first element in such a string is generally the first input-output pair and we access the initial output explicitly, if required. Furthermore, the length of an input/output string is its number of pairs, thus traces consisting of only an initial output have length zero. Prefix- and suffix-closedness are adapted analogously. In slight abuse of notation, we use $A \times B$ and $A \cdot B$ interchangeably to simplify notation.

2.4. Active automata learning

We actively learn MDPs in the minimally adequate teacher (MAT) framework introduced by Angluin for the L^* algorithm [Ang87]. Algorithms in this framework interact with an MAT to learn automata accepting some unknown regular language or modelling a black-box SUL. In the following, we describe how algorithms in the MAT framework work by abstractly describing L^*.

2.4.1. Minimally adequate teacher framework

An MAT usually needs to be able to provide answers to two types of queries that are posed by learning algorithms. These two types of queries are commonly called *membership queries* and *equivalence queries*; see Fig. 1 for a schematic depiction of the interaction between the learning algorithm, also called *learner*, and the MAT, also called *teacher*. In order to understand the basic notions of queries, consider that Angluin's original L^* algorithm is used to learn a DFA representing a regular language known to the teacher [Ang87]. Given some alphabet, the L^* algorithm repeatedly selects strings and performs membership queries to check whether these strings are in the language to be learned. The teacher may answer with either *yes* or *no*.

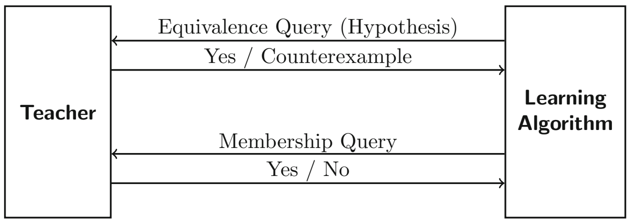

Fig. 1. The interaction between a learning algorithm and an MAT [Vaa17]

After some queries, the learning algorithm uses the knowledge gained so far and forms a hypothesis. A hypothesis is a DFA consistent with the obtained information which is supposed to accept the regular language under consideration. The algorithm presents the hypothesis to the teacher and issues an equivalence query in order to check whether the language to be learned is equivalent to the language accepted by the hypothesis automaton. The response to this kind of query is either *yes*, signalling that a correct DFA has been learned, or a counterexample to equivalence. Such a counterexample serves as a witness showing that the learned model is not yet correct, that is, it is a word in the symmetric difference of the language under learning and the language accepted by the hypothesis.

After processing a counterexample, the learner starts a new *round* of learning. The new round again involves membership queries and a concluding equivalence query. This general mode of operation is basically used by all algorithms in the MAT framework with some adaptations.

2.5. Markov decision processes

MDPs allow modelling reactive systems with probabilistic responses. An MDP starts in an initial state. During execution, the environment chooses and executes inputs non-deterministically upon which the system reacts according to its current state and its probabilistic transition function. For that, the system changes its state and produces an output.

Definition 2.1 (Markov decision process (MDP)). A labelled Markov decision process (MDP) is a tuple $\mathcal{M} = \langle Q, I, O, q_0, \delta, L\rangle$ where

- Q is a finite set of states,
- I is a finite set of input symbols,
- O is a finite set of output symbols,
- $q_0 \in Q$ is the initial state,
- $\delta\colon Q \times I \to Dist(Q)$ is the probabilistic transition function, and
- $L\colon Q \to O$ is the labelling function.

The transition function δ must be defined for all $q \in Q$ and $i \in I$, thus MDPs are input enabled in our definition. We consider only deterministic MDPs, therefore it must hold that

$$\forall\, q, q', q'' \in Q, \forall\, i\colon (\delta(q,i)(q') > 0 \wedge \delta(q,i)(q'') > 0) \to (q' = q'' \vee L(q') \neq L(q'')).$$

We generally consider deterministic labelled MDPs. Labelling of states with outputs allows us to distinguish states in a black-box setting, thus it is essential for us. Deterministic MDPs define at most one successor state for each source state and input-output pair, which ensures that a given trace always reaches the same state (see also below). This assumption simplifies learning such that learning algorithms, like IoAlergia [MCJ+16], generally place this assumption on SULs. We refer to deterministic labelled MDPs uniformly as MDPs.

As a shorthand notation, we use $\Delta\colon Q \times I \times O \to Q \cup \{\bot\}$ to compute successor states for a given source state and an input-output pair. The function is defined by $\Delta(q, i, o) = q' \in Q$ with $L(q') = o$ and $\delta(q,i)(q') > 0$ if there exists such a q', otherwise Δ returns $\bot$.

Additionally to requiring determinism and labelling, Definition 2.1 requires MDPs to be input enabled. They must not block or reject inputs. Since we assume stochastic SULs to be MDPs, this allows us to execute any input at any point in time. This is a common assumption in model-based testing [Tre08].

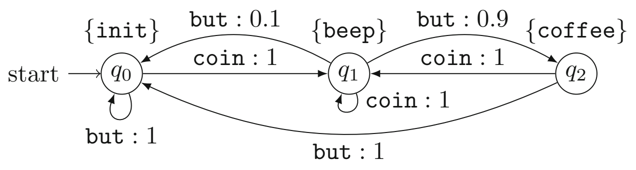

Fig. 2. An MDP modelling a faulty coffee machine

.

Example 2.1 (Faulty coffee machine). Figure 2 shows an MDP modelling a faulty coffee machine. Edge labels denote input symbols and corresponding transition probabilities, whereas output labels in curly braces are placed above states. After providing the inputs `coin` and `but`, the coffee machine MDP produces the output `coffee` with probability 0.9, but with probability 0.1 it resets itself producing the output `init`.

Relation to other automata definitions. Other common definitions of MDPs do not include labels, but assign rewards to transitions and states [Put94]. Since our goal is to learn models from observations of stochastic systems, we use labels that represent those observations. We do not not consider rewards, as they are not required in this context. In the context of verification, labels often denote propositions that hold in states [FKNP11, BK08], that is, $O = \mathcal{P}(AP)$, where AP is a set of relevant (atomic) propositions, and $L(q)$ returns the propositions that hold in state q. In Sect. 7, we formulate model-checking queries over labels.

Closely related to MDPs are probabilistic automata [SL95]. General probabilistic automata actually subsume MDPs [Sto02], but they usually do not include state labels. In particular, our definition of MDPs is similar to reactive probabilistic automata [SdV04], with the main differences being the existence of state labels and that we do not allow partial transition functions. As noted above, MDPs must be input-enabled.

2.6. Execution of Markov decision processes

A (finite) path ρ through an MDP is an alternating sequence of states and inputs starting in the initial state and ending in some state $q_n \in Q$, that is, $\rho = q_0 \cdot i_1 \cdot q_1 \cdots i_{n-1} \cdot q_{n-1} \cdot i_n q_n \in Q \times (I \times Q)^*$. The set of all paths of an MDP $\mathcal{M}$ is denoted by $Path_{\mathcal{M}}$. In each state q_k, the next input i_{k+1} is chosen non-deterministically and based on that, the next state q_{k+1} is chosen probabilistically according to $\delta(q_k, i_{k+1})$. Hence, we have for each k that $\delta(q_k, i_{k+1}) > 0$. In contrast to finite paths, infinite paths do not have a dedicated end state. An infinite path $\hat{\rho}$ is sequence $q_0 \cdot i_1 \cdot q_1 \cdot i_2 \cdots$ [FKNP11]. We denote the set of infinite paths of $\mathcal{M}$ by $IPath_{\mathcal{M}}$. Unless otherwise noted, we refer to finite paths simply as paths, as we generally consider a test-based setting in which we execute finite paths.

The execution of an MDP is controlled by a so-called *scheduler*, resolving the non-deterministic choice of inputs. A scheduler, as defined below, specifies a distribution over the next input given the current execution path. In other words, they basically choose the next input action (probabilistically) given a history of visited states. Schedulers are also referred to as adversaries or strategies [MCJ+16].

Definition 2.2 (Scheduler). Given an MDP $\mathcal{M} = \langle Q, I, O, q_0, \delta, L\rangle$, a scheduler for $\mathcal{M}$ is a function $s\colon Path_{\mathcal{M}} \to Dist(I)$.

The composition of an MDP $\mathcal{M}$ and a scheduler s induces a deterministic Markov chain [FKNP11]. A Markov chain is a fully probabilistic system allowing to define a probability measure over paths. Below, we define a probability measure over finite paths based on the definition in our article on learning-based testing [AT19b].

Probability distributions on paths. For a probability distribution over finite paths of an MDP $\mathcal{M}$ controlled by scheduler s, we also need a probability distribution $p_l \in Dist(\mathbb{N}_0)$ over the path lengths.

Definition 2.3 (Path probabilities). An MDP $\mathcal{M} = \langle Q, I, O, q_0, \delta, L\rangle$, a scheduler $s\colon Path_{\mathcal{M}} \to Dist(I)$, and a path length probability distribution p_l induce a probability distribution $\mathbb{P}^l_{\mathcal{M},s}$ on finite paths $Path_{\mathcal{M}}$ defined by:

$$\mathbb{P}^l_{\mathcal{M},s}(q_0 \cdot i_1 \cdot q_1 \cdots i_n \cdot q_n) = p_l(n) \cdot \left(\prod_{j=1}^{n} s(q_0 \cdots i_{j-1} q_{j-1})(i_j) \cdot \delta(q_{j-1}, i_j)(q_j)\right) \tag{1}$$

Alternatively to p_l, probability distributions over finite paths may, e.g., include state-dependent termination probabilities [MCJ$^+$11]. We take a path-based view because we actively sample from MDPs. Moreover, probability distributions are often defined for each state and are therefore parameterised by states [FKNP11]. Since we sample all SUL traces starting from the initial state q_0, Eq. 1 defines only probabilities of paths starting in q_0.

Scheduler subclasses. There are subclasses of schedulers that are relevant to us. We implement equivalence queries of L^*_{MDP} through conformance testing, where we aim to cover/reach states in intermediate hypothesis MDPs. Equivalence queries therefore require the computation of schedulers for probabilistic reachability properties. These properties do not need general schedulers, but can be satisfied optimally with *memoryless deterministic* schedulers [KP13]. A scheduler is memoryless if its choice of inputs depends only on the current state, thus it is a function from the states Q to $Dist(I)$. A scheduler s is deterministic if for all $\rho \in Path_{\mathcal{M}}$, there is exactly one $i \in I$ such that $s(\rho)(i) = 1$. Otherwise, it is called randomised. Example 2.2 describes an MDP and a scheduler for our faulty coffee machine introduced in Example 2.1.

Example 2.2 (Scheduler for coffee machine). A deterministic memoryless scheduler s may provide the inputs `coin` and `but` in alternation to the coffee machine of Example 2.1. Formally, $s(q_0) = 1_{\{\texttt{coin}\}}$, $s(q_1) = 1_{\{\texttt{but}\}}$, and $s(q_2) = 1_{\{\texttt{coin}\}}$. By setting the length probability distribution to $p_l = \mathbf{1}_{\{2\}}$, all strings must have length 2, such that, for instance, $\mathbb{P}^l_{\mathcal{M},s}(\rho) = 0.9$ for $\rho = q_0 \cdot \texttt{coin} \cdot q_1 \cdot \texttt{but} \cdot q_2$.

Traces. During the execution of a finite path ρ, we observe a trace $L(\rho) = t$. As mentioned above, a trace is an alternating sequence of inputs and outputs starting with an output. The trace t observed during the execution of $\rho = q_0 \cdot i_1 \cdot q_1 \cdots i_{n-1} \cdot q_{n-1} \cdot i_n \cdot q_n$ is given by $t = o_0 \cdot i_1 \cdot o_1 \cdots i_{n-1} \cdot o_{n-1} \cdot i_n \cdot o_n$ where $L(q_i) = o_i$.

We can define a probability distribution $\mathbb{P}t^l_{\mathcal{M},s}$ over traces by $\mathbb{P}t^l_{\mathcal{M},s} = \mathbb{P}^l_{\mathcal{M},s} \circ L^{-1}$. By controlling path length via p_l, we also control the length of observed traces via p_l. If it is clear from the context, we will write $\mathbb{P}^l_{\mathcal{M},s}$ for traces as well instead of $\mathbb{P}t^l_{\mathcal{M},s}$.

3. MDP observations

In the following, we first introduce different types of observation sequences of MDPs and some related terminology. Then, we formally define the semantics of MDPs in terms of observation sequences.

3.1. Sequences of observations

Let I and O be finite sets of inputs and outputs. Recall that a trace observed during the execution of an MDP is an input-output sequence in $O \times (I \times O)^*$ (see also Sect. 2.5). We say that a trace t is *observable* if there exists a path ρ with $L(\rho) = t$, thus there is a scheduler s and a length distribution p_l such that $\mathbb{P}^l_{\mathcal{M},s}(t) > 0$. In a deterministic MDP $\mathcal{M}$, each observable trace t uniquely defines a state of $\mathcal{M}$ that is reached by executing t from the initial state q_0. We compute this state by $\delta^*(t) = \delta^*(q_0, t)$ where

$$\begin{aligned}\delta^*(q, L(q)) &= q,\\ \delta^*(q, o_0 i_1 o_1 \cdots i_{n-1} o_{n-1} i_n o_n) &= \Delta(\delta^*(q, o_0 i_1 o_1 \cdots i_{n-1} o_{n-1}), i_n, o_n).\end{aligned}$$

If t is not observable, then there is no path ρ with $t = L(\rho)$, denoted by $\delta^*(t) = \bot$. Therefore, we extend Δ by defining $\Delta(\bot, i, o) = \bot$. The last output o_n of a trace $t = o_0 \cdots i_n o_n$ is denoted by $last(t)$.

We use three types of observation sequences with shorthand notations in this article:

- *Traces:* abbreviated by $\mathcal{TR} = O \times (I \times O)^*$
- *Test sequences:* abbreviated by $\mathcal{TS} = (O \times I)^*$
- *Continuation sequences:* abbreviated by $\mathcal{CS} = I \times (O \times I)^* = I \times \mathcal{TS}$

These sequence types alternate between inputs and outputs, thus they are related among each other. As mentioned in Sect. 2.3, we extend the sequence notations and the notion of prefixes to sequences containing both inputs and outputs in the context of MDPs. Therefore, we extend these concepts also to $\mathcal{TR}$, $\mathcal{TS}$, and $\mathcal{CS}$. For instance, test sequences and traces are related by $\mathcal{TR} = \mathcal{TS} \cdot O$.

As noted above, an observable trace in $\mathcal{TR}$ leads to a unique state of an MDP $\mathcal{M}$. A test sequence in $s \in \mathcal{TS}$ of length n consists of a trace in $t \in \mathcal{TR}$ with n outputs and an input $i \in I$ with $s = t \cdot i$; thus executing the test sequence $s = t \cdot i$ puts $\mathcal{M}$ into the state reached by t and tests $\mathcal{M}$'s reaction to i. Extending the notion of observability, we say that the test sequence s is observable if t is observable. A continuation sequence $c \in \mathcal{CS}$ begins and ends with an input, i.e. concatenating a trace $t \in \mathcal{TR}$ and c creates a test sequence $t \cdot c$ in $\mathcal{TS}$. Informally, continuation sequences test $\mathcal{M}$'s reaction in response to multiple consecutive inputs.

The following lemma states that an extension of a non-observable traces is also not observable. We will apply this lemma in the context of test-based learning.

Lemma 3.1 If trace $t \in \mathcal{TR}$ is not observable, then any $t' \in \mathcal{TR}$ such that $t \ll t'$ is not observable as well.

Lemma 3.1 follows directly from Eq. 1. For a non-observable trace t, we have $\forall\, s, p_l : \mathbb{P}^l_{\mathcal{M},s}(t) = 0$ and extending t to create t' only adds further factors to Eq. 1. The same property also holds for test sequences.

3.2. Semantics of MDPs

We can interpret an MDP as a function $M : \mathcal{TS} \to \mathit{Dist}(O) \cup \{\bot\}$, mapping test sequences s to output distributions or undefined behaviour for non-observable s. This follows the interpretation of Mealy machines as functions from input sequences to outputs [SHM11]. Likewise, we will define which function M captures the semantics of an MDP by adapting the Myhill-Nerode theorem on regular languages [Ner58]. In the remainder of this article, we denote the set of test sequences s where $M(s) \neq \bot$ as defined domain $dd(M)$ of M.

Definition 3.1 (MDP Semantics). The semantics of an MDP $\langle Q, I, O, q_0, \delta, L\rangle$, is a function M, defined for $i \in I$, $o \in O$, and $t \in \mathcal{TR}$ as follows:

$$\begin{aligned}
M(\epsilon)(L(q_0)) &= 1\\
M(t \cdot i) &= \bot \text{ if } \delta^*(t) = \bot\\
M(t \cdot i)(o) &= p \text{ if } \delta^*(t) \neq \bot \wedge \delta(\delta^*(t), i)(q) = p > 0 \wedge L(q) = o\\
M(t \cdot i)(o) &= 0 \text{ if } \delta^*(t) \neq \bot \wedge \nexists\, q : \delta(\delta^*(t), i)(q) = p > 0 \wedge L(q) = o.
\end{aligned}$$

Definition 3.2 (M-equivalence of traces). Two traces $t_1, t_2 \in \mathcal{TR}$ are equivalent with respect to $M : \mathcal{TS} \to \mathit{Dist}(O) \cup \{\bot\}$, denoted $t_1 \equiv_M t_2$, iff

- $\mathit{last}(t_1) = \mathit{last}(t_2)$ and
- it holds for all continuations $v \in \mathcal{CS}$ that $M(t_1 \cdot v) = M(t_2 \cdot v)$.

A function M defines an equivalence relation on traces, like the Myhill-Nerode equivalence for formal languages [Ner58]. Two traces are M-equivalent if they end in the same output and if their behaviour in response to future inputs is the same. Two traces leading to the same MDP state are in the same equivalence class of $\equiv_M$, analogously to the adapted Myhill-Nerode equivalence for Mealy machines [SHM11].

We can now state which functions characterise MDPs, as an adaptation of the Myhill-Nerode theorem for regular languages [Ner58], like for Mealy machines [SHM11].

Theorem 3.1 (Characterisation). A function $M : \mathcal{TS} \to \mathit{Dist}(O) \cup \{\bot\}$ represents the semantics of an MDP iff

1. $\equiv_M$ has finite index, . . . finite number of states
2. $M(\epsilon) = \mathbf{1}_{\{o\}}$ for an $o \in O$, . . . initial output

3. $dd(M)$ is prefix-closed, and
4. $\forall\, t \in \mathcal{TR}$: either $\forall\, i \in I : M(t \cdot i) \neq \bot$ or $\forall\, i \in I : M(t \cdot i) = \bot$. ... input enabledness

Proof *Direction* $\Rightarrow$*:* first we show that the semantics M of an MDP $\mathcal{M} = \langle Q, I, O, q_0, \delta, L\rangle$ fulfils the four conditions of Theorem 3.1. Let $t \in \mathcal{TR}$ be an observable trace, then we have for $i \in I$, $o \in O$: $M(t \cdot i)(o) = \delta(q', i)(q) = p$, where $q' = \delta^*(t)$, if $p > 0$ and $L(q) = o$. As $\mathcal{M}$ contains finitely many states q', $\delta(q', i)$ and therefore also $M(t \cdot i)$ takes only finitely many values. M-equivalence of traces t_i depends on the outcomes of M and on their last outputs $last(t_i)$. Since the set of possible outputs is finite like the possible outcomes of M, M-equivalence defines finitely many equivalence classes for observable traces. For non-observable $t \in \mathcal{TR}$ we have $\delta^*(t) = \bot$ which implies $M(t \cdot i) = \bot$. As a consequence of Lemma 3.1, we also have $M(t \cdot c) = \bot$ for any $c \in \mathcal{CS}$. Hence, non-observable traces are equivalent with respect to M if they end in the same output, therefore M defines finitely many equivalence classes for non-observable traces. In summary, $\equiv_M$ has finite index, thus fulfilling the first condition.

According to Definition 3.1, it holds that $M(\epsilon)(L(q_0)) = 1$, thus the second condition is fulfilled.

Prefix-closedness of the defined domain $dd(M)$ of M follows from Lemma 3.1. Any extension of a non-observable test sequence is also non-observable, thus M fulfils the third condition.

For the fourth condition, we again distinguish two cases. If t is a non-observable trace, i.e. $\delta^*(t) = \bot$, then $M(t \cdot i) = \bot$ for all $i \in I$ according to Definition 3.1, which fulfils the second sub-condition. For observable t, the distribution $M(t \cdot i)$ depends on $\delta(\delta^*(t), i)$, which is defined for all i due to input enabledness of $\mathcal{M}$, satisfying the first subcondition.

The semantics of an MDP satisfies all four conditions listed in Theorem 3.1.

Direction $\Leftarrow$*:* from an M satisfying the conditions given in Theorem 3.1, we can construct an MDP $\mathcal{M}_c = \langle Q, I, O, q_0, \delta, L\rangle$ by:

- $Q = (\mathcal{TR}/\equiv_M)\backslash\{[t] \mid t \in \mathcal{TR}, \exists\, i \in I : M(t \cdot i) = \bot\}$
- $q_0 = [o_0]$, where $o_0 \in O$ such that $M(\epsilon) = \mathbf{1}_{\{o_0\}}$
- $L([s \cdot o]) = o$ where $s \in \mathcal{TS}$ and $o \in O$, by Definition 3.2 all traces in the same equivalence class end with the same output
- for $[t] \in Q$:
  $\delta([t], i)([t \cdot i \cdot o]) = M(t \cdot i)(o)$, defined by fourth condition of Theorem 3.1.

Each equivalence class of $\equiv_M$ gives rise to exactly one state in Q, except for the equivalence classes of non-observable traces. □

The MDP $\mathcal{M}_c$ in the construction above is minimal with respect to the number of states and unique up to isomorphism. Therefore, we refer to an MDP constructed in this way as canonical MDP $can(M)$ for MDP semantics M. $\mathcal{M}_c$ is minimal, because any other MDP $\mathcal{M}'$ with a lower number of states necessarily contains a state that is reached by traces from different M-equivalence classes. Consequently, $\mathcal{M}'$ cannot be consistent with the semantics M.

Viewing MDPs as reactive systems, we consider two MDPs to be equivalent if we make the same observations on both.

Definition 3.3 (Output-distribution equivalence). MDPs $\mathcal{M}_1$ and $\mathcal{M}_2$ over I and O with semantics M_1 and M_2 are output-distribution equivalent, denoted $\mathcal{M}_1 \equiv_{\text{od}} \mathcal{M}_2$, iff

$$\forall\, s \in \mathcal{TS} : M_1(s) = M_2(s).$$

4. Exact learning of MDPs

This section presents $L^*_{\text{MDP}^e}$, an exact active learning algorithm for MDPs. $L^*_{\text{MDP}^e}$ serves as basis for the sampling-based learning algorithm presented in Sect. 5. In contrast to sampling, $L^*_{\text{MDP}^e}$ assumes the existence of a teacher with perfect knowledge about the SUL that is able to answer two types of queries: *output distribution* queries and *equivalence* queries. The former asks for the exact distribution of outputs following the execution of a test sequence on the SUL. The latter takes a hypothesis MDP as input and responds either with *yes* iff the hypothesis is observationally equivalent to the SUL or with a counterexample to equivalence. A counterexample is a test sequence leading to different output distributions in hypothesis and SUL.

Table 2. An observation table created during learning of the faulty coffee machine (Fig. 2)

		but	coin
S	init	{init $\mapsto$ 1}	{beep $\mapsto$ 1}
	init · coin · beep	{coffee $\mapsto$ 0.9, init $\mapsto$ 0.1}	{beep $\mapsto$ 1}
	init · coin · beep · but · coffee	{init $\mapsto$ 1}	{beep $\mapsto$ 1}
$Lt(S)$	init · but · init	{init $\mapsto$ 1}	{beep $\mapsto$ 1}
	init · coin · beep · but · init	{init $\mapsto$ 1}	{beep $\mapsto$ 1}
	init · coin · beep · coin · beep	{coffee $\mapsto$ 0.9, init $\mapsto$ 0.1}	{beep $\mapsto$ 1}
	init · coin · beep · but · coffee· but · init	{init $\mapsto$ 1}	{beep $\mapsto$ 1}
	init · coin · beep · but · coffee· coin · beep	{coffee $\mapsto$ 0.9, init $\mapsto$ 0.1}	{beep $\mapsto$ 1}

4.1. Queries

Before discussing learning, we formally define the queries available to the learner that focus on the observable behaviour of MDPs. Assume that we want to learn a model of a black-box deterministic MDP $\mathcal{M}$, with semantics M. Output distribution queries (**odq**) and equivalence queries (**eq**) are then defined as follows:

- *output distribution query (*$\mathbf{odq}$*)*: an $\mathbf{odq}(s)$ returns $M(s)$ for input $s \in \mathcal{TS}$.
- *equivalence query (*$\mathbf{eq}$*)*: an **eq** query takes a hypothesis MDP $\mathcal{H}$ with semantics H as input and returns *yes* if $\mathcal{H} \equiv_{\text{od}} \mathcal{M}$; otherwise it returns an $s \in \mathcal{TS}$ such that $H(s) \neq M(s)$ and $M(s) \neq \bot$.

Lemma 4.1 (Counterexample Observability). For any counterexample s to $\mathcal{H} \equiv_{\text{od}} \mathcal{M}$ with $M(s) = \bot$, there exists a prefix s' of s with $H(s') \neq M(s')$ and $M(s') \neq \bot$, thus s' is also a counterexample but observable on the SUL with semantics M. Hence, we can restrict potential counterexamples to be observable test sequences.

Proof Since s is a counterexample and $M(s) = \bot$, we have $H(s) \neq \bot$. Let s'' be the the longest prefix of s such that $M(s'') = \bot$, thus s'' is of the form $s'' = s' \cdot o \cdot i$ with $M(s')(o) = 0$. Due to prefix-closedness of $dd(H)$, $H(s) \neq \bot$ implies $H(s'') \neq \bot$, therefore $H(s')(o) > 0$. Hence, s' with $M(s') \neq \bot$ is also a counterexample because $H(s')(o) \neq M(s')(o) \Rightarrow H(s') \neq M(s')$. $\square$

4.2. Observation tables

Like Angluin's L^* [Ang87], we store information in observation table triples $\langle S, E, T \rangle$, where:

- $S \subseteq \mathcal{TR}$ is a prefix-closed set of traces, initialised to $\{o_0\}$, a singleton set containing the trace consisting of the initial output o_0 of the SUL, given by $\mathbf{odq}(\epsilon)(o_0) = 1$,
- $E \subseteq \mathcal{CS}$ is a suffix-closed set of continuation sequences, initialised to I,
- $T\colon (S \cup Lt(S)) \cdot E \to \mathit{Dist}(O) \cup \{\bot\}$ is a mapping from test sequences to output distributions or to $\bot$, denoting undefined behaviour. This mapping basically stores a finite subset of M. The set $Lt(S) \subseteq S \cdot I \cdot O$ is given by $Lt(S) = \{s \cdot i \cdot o \mid s \in S, i \in I, o \in O, \mathbf{odq}(s \cdot i)(o) > 0\}$.

We can view an observation table as a two-dimensional array with rows labelled by traces in $S \cup Lt(S)$ and columns labelled by E. We refer to traces in S as short traces and to their extensions in $Lt(S)$ as long traces. An extension $s \cdot i \cdot o$ of a short trace s is in $Lt(S)$ if $s \cdot i \cdot o$ is observable. Analogously to traces, we refer to rows labelled by S as short rows and we refer to rows labelled by $Lt(S)$ as long rows. The table cells store the mapping defined by T. To represent rows labelled by traces s we use functions $row(s)\colon E \to \mathit{Dist}(O) \cup \{\bot\}$ for $s \in S \cup Lt(S)$ with $row(s)(e) = T(s \cdot e)$. Equivalence of rows labelled by traces s_1, s_2, denoted $\mathbf{eqRow}_E(s_1, s_2)$, holds iff $row(s_1) = row(s_2) \wedge last(s_1) = last(s_2)$. It approximates M-equivalence $s_1 \equiv_M s_2$ by considering only continuations in E, hence $s_1 \equiv_M s_2$ implies $\mathbf{eqRow}_E(s_1, s_2)$. The observation table content defines the structure of hypothesis MDPs based on the following principle: we create one state per equivalence class of $S/\mathbf{eqRow}_E$, thus we identify states with traces in S reaching them and we distinguish states by their future behaviour in response to sequences in E. The long traces $Lt(S)$ serve to define transitions. This is a common approach to hypothesis construction in active automata learning [SHM11]. Transition probabilities are given by the distributions in the mapping T.

Algorithm 1 Function for making an observation table closed and consistent

```
1: function MakeClosedAndConsistent($\langle S, E, T\rangle$)
2:     if $\langle S, E, T\rangle$ is not closed then
3:         $l \leftarrow l' \in Lt(S)$ such that $\forall s \in S : row(s) \neq row(l') \vee last(s) \neq last(l')$
4:         $S \leftarrow S \cup \{l\}$
5:         return $\langle S, E, T\rangle$
6:     else if $\langle S, E, T\rangle$ is not consistent then
7:         for all $s_1, s_2 \in S$ such that $\mathbf{eqRow}_E(s_1, s_2)$ do
8:             for all $i \in I, o \in O$ do
9:                 if $T(s_1 \cdot i)(o) > 0$ and $\neg\ \mathbf{eqRow}_E(s_1 \cdot i \cdot o, s_2 \cdot i \cdot o)$ then
10:                    $e \leftarrow e' \in E$ such that $T(s_1 \cdot i \cdot o \cdot e') \neq T(s_2 \cdot i \cdot o \cdot e')$
11:                    $E \leftarrow E \cup \{i \cdot o \cdot e\}$
12:                    return $\langle S, E, T\rangle$
13:                end if
14:            end for
15:        end for
16:    end if
17:    return $\langle S, E, T\rangle$                ▷ reached if already closed and consistent
18: end function
```

Example 4.1 (Observation table for coffee machine). Table 2 shows an observation table created during learning of the coffee machine shown in Fig. 2. The hypothesis derived from that observable table is already equivalent to the true model. The set S includes a trace for each state of the coffee machine. Note that these traces are pairwise inequivalent with respect to $\mathbf{eqRow}_E$, where $E = I = \{\texttt{but}, \texttt{coin}\}$. As noted above, the traces labelled by $Lt(S)$ define transitions in hypotheses. For instance, the row labelled by `init` · `but` · `init` gives rise to the self-loop transition in the initial state with the input `but` and probability 1. This is the case because the rows labelled by `init` · `but` · `init` and `init` are equivalent, where `init` $\in S$ corresponds to the initial state of the hypothesis.

Definition 4.1 (Closedness). An observation table $\langle S, E, T\rangle$ is closed if for all $l \in Lt(S)$ there is an $s \in S$ such that $\mathbf{eqRow}_E(l, s)$.

Definition 4.2 (Consistency). An observation table $\langle S, E, T\rangle$ is consistent if for all $s_1, s_2 \in S$, $i \in I$, $o \in O$ such that $\mathbf{eqRow}_E(s_1, s_2)$ it holds either that (1)[1] $T(s_1 \cdot i)(o) = 0 \wedge T(s_2 \cdot i)(o) = 0$ or (2) $\mathbf{eqRow}_E(s_1 \cdot i \cdot o, s_2 \cdot i \cdot o)$.

Closedness and consistency are required to derive well-formed hypotheses, analogously to L^* [Ang87]. We require closedness to create transitions for all inputs in all states and we require consistency to be able to derive deterministic hypotheses. During learning, we apply Algorithm 1 repeatedly to establish closedness and consistency of observation tables. The algorithm adds a new short trace if the table is not closed and adds a new column if the table is not consistent.

We derive a hypothesis $\mathcal{H} = \langle Q_\mathrm{h}, I, O, q_{0\mathrm{h}}, \delta_\mathrm{h}, L_\mathrm{h}\rangle$ from a closed and consistent observation table $\langle S, E, T\rangle$, denoted $\mathcal{H} = \mathrm{hyp}(S, E, T)$, as follows:

- $Q_\mathrm{h} = \{\langle last(s), row(s)\rangle \mid s \in S\}$
- $q_{0\mathrm{h}} = \langle o_0, row(o_0)\rangle$, $o_0 \in S$ is the trace consisting of the initial SUL output
- for $s \in S$, $i \in I$ and $o \in O$:
  $\delta_\mathrm{h}(\langle last(s), row(s)\rangle, i)(\langle o, row(s \cdot i \cdot o)\rangle) = p$ if $T(s \cdot i)(o) = p > 0$ and 0 otherwise
- for $s \in S$: $L_\mathrm{h}(\langle last(s), row(s)\rangle) = last(s)$.

4.3. Learning algorithm

Algorithm 2 implements $L^*_{\mathrm{MDP}^e}$ using queries **odq** and **eq**. First, the algorithm initialises the observation table and fills the table cells with output distribution queries (lines 1 to 4). The main loop in lines 5 to 19 makes the observation table closed and consistent, derives a hypothesis $\mathcal{H}$ and performs an equivalence query $\mathbf{eq}(\mathcal{H})$.

[1]Note that $s_1 \in S$ implies that $T(s_1 \cdot i) \neq \perp$ such that $T(s_2 \cdot i)(o) = 0$ follows from $\mathbf{eqRow}_E(s_1, s_2)$ and $T(s_1 \cdot i)(o) = 0$.

Algorithm 2 The main algorithm implementing $L^*_{\mathrm{MDP}^e}$

```
Input: $I$, exact teacher capable of answering $\mathbf{odq}$ and $\mathbf{eq}$
Output: learned model $\mathcal{H}$ (final hypothesis)
 1: $o_\Gamma \leftarrow o$ such that $\mathbf{odq}(\epsilon)(o) = 1$
 2: $S \leftarrow \{o_0\}$
 3: $E \leftarrow I$                                            ▷ Initialise observation table
 4: FILL($S, E, T$)
 5: repeat
 6:     while $\langle S, E, T\rangle$ not closed or not consistent do
 7:         $\langle S, E, T\rangle \leftarrow$ MAKECLOSEDANDCONSISTENT($\langle S, E, T\rangle$)
 8:         FILL($S, E, T$)
 9:     end while
10:     $\mathcal{H} \leftarrow$ hyp($S, E, T$)
11:     $eqResult \leftarrow \mathbf{eq}(\mathcal{H})$
12:     if $eqResult \neq yes$ then
13:         $cex \leftarrow eqResult$
14:         for all $(t \cdot i) \in prefixes(cex)$ with $i \in I$ do
15:             $S \leftarrow S \cup \{t\}$
16:         end for
17:         FILL($S, E, T$)
18:     end if
19: until $eqResult = yes$
20: return hyp($S, E, T$)

21: procedure FILL($S, E, T$)
22:     for all $s \in S \cup Lt(S)$, $e \in E$ do
23:         if $T(s \cdot e)$ undefined then            ▷ we have no information about $T(s \cdot e)$ yet
24:             $T(s \cdot e) \leftarrow \mathbf{odq}(s \cdot e)$
25:         end if
26:     end for
27: end procedure
```

If $\mathbf{eq}(\mathcal{H})$ returns a counterexample cex, we add all its prefix traces as short traces to S, otherwise Algorithm 2 returns the final hypothesis, as it is output-distribution equivalent to the SUL. Whenever the observation table contains empty cells, the FILL procedure assigns values to these cells via $\mathbf{odq}$.

Note that we use the classic L^*-style counterexample processing, because other techniques to process counterexamples are hard to apply efficiently in a sampling-based setting. It would be possible to adapt Rivest and Schapire's counterexample processing [RS93] in the exact setting considered in this section, but the main purpose of introducing $L^*_{\mathrm{MDP}^e}$ is to provide a basis for the sampling-based L^*_{MDP}.

4.4. Correctness and termination

In the following, we will show that $L^*_{\mathrm{MDP}^e}$ terminates and learns correct models. We consider learned models correct if they are output-distribution equivalent to the SUL. Like Angluin [Ang87], we will show that the derived hypotheses are consistent with queried information and that they are minimal with respect to the number of states. For the remainder of this section, let M be the semantics of the MDP underlying the SUL and let $\mathcal{M} = can(M)$ be the corresponding canonical MDP and let $\mathcal{H} = \langle Q, I, O, q_0, \delta, L\rangle$ denote hypotheses. The first two lemmas relate to observability of traces.

Lemma 4.2 For all $s \in \mathcal{TS}$, $o \in O$, $e \in \mathcal{CS}$: $M(s)(o) = 0 \Rightarrow M(s \cdot o \cdot e) = \bot$.

Proof. Let δ_M be the probabilistic transition relation of $\mathcal{M}$. $M(s)(o) = 0$ with $s = t \cdot i$ for $i \in I$ implies that there is no state labelled o reachable by executing i in the state $\delta^*_M(t)$ (Definition 3.1), thus $\delta^*_M(t \cdot i \cdot o) = \delta^*_M(s \cdot o) = \bot$. By Definition 3.1, $M(s \cdot o \cdot i') = \bot$ for any i'. Due to prefix-closedness of $dd(M)$, we have $M(s \cdot o \cdot e) = \bot$ for all $e \in \mathcal{CS}$. $\square$

Lemma 4.3 Let $\langle S, E, T\rangle$ be a closed and consistent observation table created by Algorithm 2. Then for $s \in S$ and $s \cdot i \cdot o \in S \cup Lt(S)$ we have $T(s \cdot i)(o) > 0$.

Proof The lemma states that traces labelling rows are observable. Algorithm 2 adds elements to S and consequently to $Lt(S)$ in two cases: (1) if an equivalence query returns a counterexample and (2) to make observation tables closed.

Case 1. Counterexamples $c \in \mathcal{TS}$ returned by equivalence queries $\mathbf{eq}(\mathcal{H})$ satisfy $M(c) \neq \bot$ (see also Lemma 4.1). In Line 15 of Algorithm 2, we add t_p to S for each $t_p \cdot i_p \in \mathit{prefixes}(c)$. Due to prefix-closedness of $dd(M)$, $M(t_p \cdot i_p) \neq \bot$ for all $t_p \cdot i_p \in \mathit{prefixes}(c)$, and therefore $M(s \cdot i)(o) = T(s \cdot i)(o) > 0$ for each added trace t_p of the form $t_p = s \cdot i \cdot o$ with $i \in I$ and $o \in O$. Due to its definition, the set $Lt(S)$ is implicitly extended by all observable extensions of added t_p. By this definition, $Lt(S)$ contains only traces $t = s \cdot i \cdot o$ such that $T(s \cdot i)(o) > 0$.
Case 2. If an observation table is not closed, we add traces from $Lt(S)$ to S. As noted above, all traces $t = s \cdot i \cdot o$ in $Lt(S)$ satisfy $T(s \cdot i)(o) > 0$. Consequently, all traces added to S also satisfy this property. □

Theorem 4.1 (Consistency and minimality). Let $\langle S, E, T\rangle$ be a closed and consistent observation table and let $\mathcal{H} = \mathrm{hyp}(S, E, T)$ be a hypothesis derived from that table with semantics H. Then $\mathcal{H}$ is consistent with T, that is, $\forall\, s \in (S \cup Lt(s)) \cdot E\colon\ T(s) = H(s)$, and any other MDP consistent with T but inequivalent to $\mathcal{H}$ must have more states.

The following four lemmas are necessary to prove Theorem 4.1.

Lemma 4.4 Let $\langle S, E, T\rangle$ be a closed and consistent observation table. For $\mathcal{H} = \mathrm{hyp}(S, E, T)$ and every $s \in S \cup Lt(S)$, we have $\delta^*(q_0, s) = \langle \mathit{last}(s), \mathit{row}(s)\rangle$.

Proof Similarly to Angluin [Ang87], we prove this by induction on the length k of s. In Sect. 2.3, we defined the length of a trace as its number of input-output pairs. Hence, for the base case of $k = 0$, we consider the trace s consisting of only the initial output o, that is, $s = o$. In this case, we have

$$\delta^*(q_0, o) = \delta^*(\langle o, \mathit{row}(o)\rangle, o) = \langle o, \mathit{row}(o)\rangle.$$

Assume that for every $s \in S \cup Lt(S)$ of length at most k, $\delta^*(q_0, s) = \langle \mathit{last}(s), \mathit{row}(s)\rangle$. Let $t \in S \cup Lt(S)$ be a trace of length $k + 1$. Such a t is of the form $t = s \cdot i \cdot o_t$, with $s \in \mathcal{TR}$, $i \in I$, $o_t \in O$. If $t \in Lt(S)$ then s must be in S, and if $t \in S$, then also $s \in S$ because S is prefix-closed.

$$\begin{aligned}
\delta^*(q_0, s \cdot i \cdot o_t) &= \Delta(\delta^*(q_0, s), i, o_t) && \text{(definition of } \delta^*\text{)}\\
&= \Delta(\langle \mathit{last}(s), \mathit{row}(s)\rangle, i, o_t) && \text{(by induction hypothesis)}\\
&= \langle o_t, \mathit{row}(s \cdot i \cdot o_t)\rangle && \text{(definition of } \Delta\text{)}\\
&\quad \text{if } \delta(\langle \mathit{last}(s), \mathit{row}(s)\rangle, i)(\langle o_t, \mathit{row}(s \cdot i \cdot o_t)\rangle) > 0\\
&\quad \text{and } L(\langle o_t, \mathit{row}(s \cdot i \cdot o_t)\rangle) = o_t
\end{aligned}$$

$$\begin{aligned}
&\delta(\langle \mathit{last}(s), \mathit{row}(s)\rangle, i)(\langle o_t, \mathit{row}(s \cdot i \cdot o_t)\rangle) > 0\\
&\qquad \Leftrightarrow T(s \cdot i)(o_t) > 0 && \text{(construction of } \delta\text{)}\\
&\qquad \Leftrightarrow \mathbf{true} && \text{(Lemma 4.3)}\\
&L(\langle o_t, \mathit{row}(s \cdot i \cdot o_t)\rangle) = o_t\\
&\qquad \Leftrightarrow \mathbf{true} && \text{(construction of } L\text{)}
\end{aligned}$$

Lemma 4.5 Let (S, E, T) be a closed and consistent observation table. Then $\mathrm{hyp}(S, E, T)$ is consistent with T, i.e. for every $s \in S \cup Lt(S)$ and $e \in E$ we have $T(s \cdot e) = H(s \cdot e)$.

Proof We will prove this by induction on k, the number of inputs in e. As induction hypothesis, we assume that $T(s \cdot e) = H(s \cdot e)$ for all $s \in S \cup Lt(S)$ and $e \in E$ containing at most k inputs. For the base case with $k = 1$, we consider e consisting of a single input, that is, $e \in I$. From Definition 3.1 we can derive that $H(s \cdot i) \neq \bot$ if

$\delta^*(s) \neq \bot$, which holds for $s \in S \cup Lt(S)$. Then we have:

$$\begin{aligned} H(s \cdot i)(o) &= \delta(\delta^*(s), i)(q) \text{ with } L(q) = o \\ &= \delta(\langle last(s), row(s)\rangle, i)(q) \text{ with } L(q) = o && \text{(Lemma 4.4)} \\ &= \delta(\langle last(s), row(s)\rangle, i)(\langle o, row(s \cdot i \cdot o)\rangle) && \text{(hypothesis construction)} \\ &= T(s \cdot i)(o) && \text{(hypothesis construction)} \end{aligned}$$

For the induction step, let $e \in E$ be such that it contains $k + 1$ inputs, thus it is of the form $e = i \cdot o \cdot e_k$ for $i \in I$, $o \in O$, and due to suffix-closedness of E, $e_k \in E$. Note that e_k contains k inputs. We have to show that $T(s \cdot e) = H(s \cdot e)$ for $s \in S \cup Lt(S)$. Let $s' \in S$ such that $\mathbf{eqRow}_E(s, s')$, which exists due to observation table closedness. Traces s and s' lead to the same hypothesis state because:

$$\begin{aligned} \delta^*(q_0, s) &= \langle last(s), row(s)\rangle && \text{(Lemma 4.4)} \\ &= \langle last(s'), row(s')\rangle && (\mathbf{eqRow}_E(s, s')) \\ &= \delta^*(q_0, s') && \text{(Lemma 4.4)} \end{aligned}$$

Thus, s and s' are H-equivalent and therefore $H(s \cdot e) = H(s' \cdot e)$. Due to $\mathbf{eqRow}_E(s, s')$, $T(s \cdot e) = T(s' \cdot e)$ and in combination:

$$T(s \cdot e) = H(s \cdot e) \Leftrightarrow T(s' \cdot e) = H(s' \cdot e) \Leftrightarrow T(s' \cdot i \cdot o \cdot e_k) = H(s' \cdot i \cdot o \cdot e_k).$$

We will now show that $T(s' \cdot i \cdot o \cdot e_k) = H(s' \cdot i \cdot o \cdot e_k)$ holds by considering two cases.
Case 1. Suppose that $s' \cdot i \cdot o \in S \cup Lt(S)$. Then, $T(s' \cdot i \cdot o \cdot e_k) = H(s' \cdot i \cdot o \cdot e_k)$ holds by the induction hypothesis, as e_k contains k inputs.
Case 2. Suppose that $s' \cdot i \cdot o \notin S \cup Lt(S)$. Due to $s' \in S$ and the definition of $Lt(S)$, we have

$$\mathbf{odq}(s' \cdot i)(o) = M(s' \cdot i)(o) = 0.$$

By Lemma 4.2, it follows that $M(s' \cdot i \cdot o \cdot e) = \bot$ for any continuation $e \in \mathcal{CS}$. Since the observation table is filled via $\mathbf{odq}$ we have

$$T(s' \cdot i \cdot o \cdot e_k) = \mathbf{odq}(s' \cdot i \cdot o \cdot e_k) = M(s' \cdot i \cdot o \cdot e_k) = \bot.$$

By the induction base, we have $H(s' \cdot i) = T(s' \cdot i)$ for $i \in I$, thus $H(s' \cdot i)(o) = T(s' \cdot i)(o)$, for which we know $T(s' \cdot i)(o) = 0$, because $s' \cdot i \cdot o \notin S \cup Lt(S)$. Combined with Lemma 4.2, we can conclude that $H(s' \cdot i \cdot o \cdot e_k) = \bot$. In both cases, it holds that $H(s' \cdot i \cdot o \cdot e_k) = T(s' \cdot i \cdot o \cdot e_k)$ which is equivalent to $H(s \cdot e) = T(s \cdot e)$. □

With Lemma 4.5, we have shown consistency between the derived hypotheses and the queried information stored in T. Now, we show that hypotheses are minimal with respect to the number of states.

Lemma 4.6 Let $\langle S, E, T\rangle$ be a closed and consistent observation table and let n be the number of different values of $\langle last(s), row(s)\rangle$, i.e. the hypothesis hyp(S, E, T) has n states due to its construction. Then any MDP consistent with T must have at least n states.

Proof Let $\mathcal{M}' = \langle Q', I, O, q_0', \delta', L'\rangle$ with semantics M' be an MDP that is consistent with T. Let $s_1, s_2 \in S$ be such that $\neg\, \mathbf{eqRow}_E(s_1, s_2)$, then (1) $last(s_1) \neq last(s_2)$ or (2) $row(s_1) \neq row(s_2)$. If $last(s_1) \neq last(s_2)$, then s_1 and s_2 cannot reach the same state in $\mathcal{M}'$, because the states reached by s_1 and s_2 need to be labelled differently. If $row(s_1) \neq row(s_2)$, then there exists an $e \in E$ such that $T(s_1 \cdot e) \neq T(s_2 \cdot e)$. Since $\mathcal{M}'$ is consistent with T, it holds also that $M'(s_1 \cdot e) \neq M'(s_2 \cdot e)$. In this case, s_1 and s_2 cannot reach the same state as well, as the observed future behaviour is different.

Consequently, $\mathcal{M}'$ has at least n states, because there must exist at least one state for each value of $\langle last(s), row(s)\rangle$. Two traces s_1 and s_2 satisfying $\langle last(s_1), row(s_1)\rangle \neq \langle last(s_2), row(s_2)\rangle$ cannot reach the same state. □

Lemma 4.7 Let $\langle S, E, T\rangle$ be a closed and consistent observation table and $\mathcal{H} = $ hyp(S, E, T) be a hypothesis with n states derived from it. Any other MDP $\mathcal{M}' = \langle Q', I, O, q_0', \delta', L'\rangle$ with semantics M' consistent with T, initial output $L'(q_0') = L(q_0)$, and with n or fewer states is isomorphic to $\mathcal{H}$.

Proof From Lemma 4.6, it follows that $\mathcal{M}'$ has at least n states, therefore we examine $\mathcal{M}'$ with exactly n states. For each state of $\mathcal{H}$, i.e. for each unique row value $\langle last(s), row(s)\rangle$ labelled by some $s \in S$, there exists a unique state in Q'. Let ϕ be a mapping from short traces to Q' given by $\phi(\langle last(s), row(s)\rangle) = \delta'^*(q'_0, s)$ for s in S. This mapping is bijective and we will now show that it maps q_0 to q'_0, that it preserves the probabilistic transition relation and that it preserves labelling.
First, we start with the initial state and show $\phi(q_0) = q'_0$:

$$\begin{aligned}\phi(q_0) &= \phi(\langle o, row(o)\rangle) \text{ where } o \text{ is the initial output of the SUL}\\ &= \delta'^*(q'_0, o)\\ &= q'_0 && \text{(definition of } \delta'^*)\end{aligned}$$

For each s in S, i in I and $o \in O$. We have:

$$\begin{aligned}&\delta(\langle last(s), row(s)\rangle, i)(\langle last(s\cdot i\cdot o), row(s\cdot i\cdot o)\rangle)\\ &\quad= T(s\cdot i)(o) && \text{(hypothesis construction)}\\ &\quad\text{and}\\ &\delta'(\phi(\langle last(s), row(s)\rangle), i)(\phi(\langle last(s\cdot i\cdot o), row(s\cdot i\cdot o)\rangle))\\ &\quad= \delta'(\delta'^*(q'_0, s), i)(\delta'^*(q'_0, s\cdot i\cdot o))\\ &\quad= M'(s\cdot i)(o) && \text{(Definition 3.1)}\\ &\quad= T(s\cdot i)(o) && (\mathcal{M}' \text{ is consistent with } T)\end{aligned}$$

Transition probabilities are preserved by the mapping ϕ.

Finally, we show that labelling is preserved. For all s in S:

$$\begin{aligned}L'(\phi(\langle last(s), row(s)\rangle)) &= L'(\delta'^*(q'_0, s))\\ &= last(s) && \text{(definition of } \delta'^*)\\ &\text{and}\\ L(\langle last(s), row(s)\rangle) &= last(s) && \text{(definition of } L)\\ \Rightarrow L'(\phi(\langle last(s), row(s)\rangle)) &= L(\langle last(s), row(s)\rangle)\end{aligned}$$

Labelling is preserved by the mapping ϕ.

Hence, ϕ is an isomorphism between $\mathcal{H}$ and $\mathcal{M}'$. □

This concludes the proof of Theorem 4.1. Lemma 4.5 shows consistency between the hypotheses and the queried information stored in $\langle S, E, T\rangle$. From Lemma 4.6 and Lemma 4.7, we can infer that any MDP consistent with $\langle S, E, T\rangle$, but inequivalent to hypothesis hyp(S, E, T), must have more states than hyp(S, E, T).

Theorem 4.2 (Termination and correctness). The algorithm $L^*_{\text{MDP}^e}$ terminates and returns an MDP $\mathcal{H}$ isomorphic to $\mathcal{M}$, thus it is minimal and it also satisfies $\mathcal{M} \equiv_{\text{od}} \mathcal{H}$.

Proof
Termination. Let $\langle S, E, T\rangle$ be a closed and consistent observation table and let $c \in \mathcal{TS}$ be a counterexample to equivalence between $\mathcal{M}$ and hypothesis hyp(S, E, T) with semantics H. Since c is a counterexample, $M(c) \neq H(c)$. Now let $\langle S', E', T'\rangle$ be an observation table extended by adding all prefix traces of c to S and (re-)establishing closedness and consistency. For hyp$(S', E', T') = \mathcal{H}'$ with semantics H', we have $T'(c) = M(c)$ due to output distribution queries. Since $\mathcal{H}'$ is consistent with T', we have $T'(c) = H'(c) = M(c)$. Hence, $H'(c) \neq H(c)$, which shows that $\mathcal{H}'$ is not equivalent to $\mathcal{H}$, with c being a counterexample to equivalence. We do not remove elements from S, E, or T, thus $\mathcal{H}'$ is also consistent with T. Therefore, $\mathcal{H}'$ must have at least one state more than $\mathcal{H}$ according to Theorem 4.1. It follows that each round of learning, which finds a counterexample, adds at least one state. Since Algorithm 2 derives minimal hypotheses and $\mathcal{M}$ can be modelled with finitely many states, there can only be finitely many rounds that find counterexamples. Hence, Algorithm 2 terminates after a finite number of rounds, because it returns the final hypothesis as soon as an equivalence query **eq** returns *yes*, which happens once no counterexample can be found.

We now need to show that every individual round terminates after finite time. The loops starting in Line 14 and Line 22 terminate as they iterate over finite sets. The loop starting in Line 6 terminates, as every execution of the loop either creates a new short row (making the table closed) or creates at least one new row that may be short or long (making the table consistent). The former essentially creates a new hypothesis state and the latter either creates a new state directly or violates closedness, which eventually creates a new state. Since we create minimal hypotheses of finite-state MDPs, these two operations cannot be carried out indefinitely.

Algorithm 2 terminates because it performs a finite number of learning rounds and every individual round terminates.

Correctness. The algorithm terminates when an equivalence query $\mathbf{eq}(\mathcal{H})$ does not find a counterexample between the final hypothesis $\mathcal{H}$ and $\mathcal{M}$. Since there is no counterexample at this point, we have $\mathcal{H} \equiv_{\text{od}} \mathcal{M}$. Theorem 4.1 states that $\mathcal{H}$ is minimal and $\mathcal{M} = can(M)$ is consistent with T, because T is filled through output queries. Consequently, it follows from Lemma 4.7 that $\mathcal{H}$ is isomorphic to $\mathcal{M}$, the canonical MDP modelling the SUL. $\square$

5. Learning MDPs by sampling

In this section, we introduce L^*_{MDP}, a sampling-based learning algorithm for MDPs derived from $L^*_{\text{MDP}^e}$. In contrast to $L^*_{\text{MDP}^e}$, which requires exact information, L^*_{MDP} places weaker assumptions on the teacher. It does not require exact *output distribution* queries and *equivalence* queries, but approximates these queries via sampling by directed testing of the SUL. Distribution comparisons are consequently approximated through statistical tests. While we use similar data structures as in Sect. 4, we alter the learning algorithm structure. Since large amounts of data are required to produce accurate models, the sampling-based L^*_{MDP} allows to derive an approximate model at any time, unlike most other L^*-based algorithms. This section is split into three parts: first, we present a sampling-based interface between teacher and learner, as well as the interface between teacher and SUL. The second part and the third part describe the adapted learner and the implementation of the teacher, respectively.

Confidence parameter. We apply statistical tests in both learner and teacher. The confidence levels of these tests (see Condition 2.b of Definition 5.2) depend on a parameter α [CO99], which we assume to be globally accessible to simplify presentation. For convergence proofs, we let α depend on the number of sampled traces in $\mathcal{S}$, but we observed slightly better performance for small constant values, such as 0.05; see also Sect. 7. In general, learning performance can be assumed to be robust with respect to the exact setting of α. Carrasco and Oncina pointed out that: “The algorithm behaved robustly with respect to the choice of parameter α, due to its logarithmic dependence on the parameter.” [CO99]. Similar observations have been made by Mao et al. [MCJ+16], who applied Hoeffding bounds as well.

5.1. Queries

The sampling-based teacher maintains a multiset of traces $\mathcal{S}$ for the estimation of output distributions. Whenever new traces are sampled in the course of learning, they are added to $\mathcal{S}$. In contrast to the exact setting, the teacher offers four queries, namely an equivalence query and three queries relating to output distributions and samples $\mathcal{S}$:

- *frequency query* (**fq**): given a test sequence $s \in \mathcal{TS}$, $\mathbf{fq}(s) \colon O \to \mathbb{N}_0$ are output frequencies observed after s, where $\mathbf{fq}(s)(o) = \mathcal{S}(s \cdot o)$ for $o \in O$.
- *complete query* (**cq**): given a test sequence $s \in \mathcal{TS}$, $\mathbf{cq}(s)$ returns $\mathbf{true}$ if sufficient information is available to estimate an output distribution from $\mathbf{fq}(s)$; returns $\mathbf{false}$ otherwise.
- *refine query* (**rfq**): instructs the teacher to refine its knowledge of the SUL by testing it directed towards rarely observed samples. Traces sampled by **rfq** are added to $\mathcal{S}$, increasing the accuracy of subsequent probability estimations.
- *equivalence query* (**eq**): given a hypothesis $\mathcal{H}$, **eq** tests for output-distribution equivalence between the SUL and $\mathcal{H}$; returns a counterexample from $\mathcal{TS}$ showing non-equivalence or returns *none* if no counterexample was found. As is common in active automata learning, we approximate equivalence queries in L^*_{MDP} via testing [SHM11]. For this reason, the approximate equivalence queries return *none*, rather than *yes*, if no counterexample is detected.

The sampling-based teacher thus needs to implement two different testing strategies, one for increasing the accuracy of probability estimations along observed traces (*refine query*) and one for finding discrepancies between a hypothesis and the SUL (*equivalence query*). The *frequency query* and the *complete query* are used for hypothesis construction by the learner.

To test the SUL, we require the ability to (1) reset it and to (2) perform an input action and observe the produced output. For the remainder of this section, let $\mathcal{M} = \langle Q, I, O, q_0, \delta, L \rangle$ be the canonical MDP underlying the SUL with semantics M. Based on $q \in Q$, the current execution state of $\mathcal{M}$, we define two operations available to the teacher:

- **reset:** resets $\mathcal{M}$ to the initial state by setting $q = q_0$ and returns $L(q_0)$.
- **step:** takes an input $i \in I$ and selects a new state q' according to $\delta(q, i)(q')$. The **step** operation then updates the execution state to q' and returns $L(q')$.

Note that we consider $\mathcal{M}$ to be a black box, assuming its structure and transition probabilities to be unknown. We are only able to perform inputs and observe output labels. For instance, we observe the initial SUL output $L(q_0)$ after performing a **reset**.

5.1.1. Queries: requirements and properties

The presentation of queries is tailored to our implementation L^*_{MDP}, as we need to take practical insights regarding the cost of sampling into account for efficient learning. Since this implementation-biased presentation may be an obstacle towards the adoption and adaptation of L^*_{MDP}, we provide general requirements and implementation guidelines for queries in the following.

Trace storage. We assume that the teacher stores a multiset of traces. This assumption helps us to simplify the presentation, but it is not required. To enable efficient learning and convergence, we need to ensure that all sampled traces are stored somewhere. For example, a valid alternative approach would be that the learner keeps track of all sampled information. In this scenario, the refine query would need to return a multiset of sampled traces just like the equivalence queries, which would optionally also return a counterexample. In summary, there needs to be a multiset $\mathcal{S}$ which contains all sampled traces. If the teacher keeps track of $\mathcal{S}$, there needs to be a way to query the information stored in $\mathcal{S}$. Here, this is implemented through the frequency query that returns the current information stored in $\mathcal{S}$.

Convergence. Analogous to exact learning of DFA [dlH10], L^*_{MDP} can learn solely from equivalence queries, but it would be less efficient. To ensure convergence, implementations of L^*_{MDP} need to satisfy the below requirements concerning equivalence queries and completeness queries.

Equivalence queries generally sample traces of the SUL to find counterexamples to equivalence between the SUL and the hypothesis. At the same time, the sampling also collects information on already explored parts of the SUL's state space. To ensure convergence in the limit, equivalence queries need to sample every input in every reachable state infinitely often. To sample every input infinitely often, we assign a non-zero probability to every input in all input selections performed during equivalence queries. To ensure that every state can be reached, the length of traces must not be bounded, that is, every trace length must have a non-zero probability. Our implementation of equivalence queries satisfies this requirement by using a geometric distribution of trace length unless a counterexample is detected. Note that equivalence queries will eventually stop detecting new counterexamples. IoAlergia assumes a similar sampling regime for convergence [MCJ+12, MCJ+16]

For implementations of the completeness query, there must exist a natural number k, such that $\mathbf{cq}(s)$ returns true for a test sequence s if s has been sampled at least k times. Otherwise, $\mathbf{cq}(s)$ must return false. In other words, the completeness queries must return true after having observed finitely many samples and it must not return true for sequences that have not been observed.

Efficiency. Finally, we would like to discuss how the implementation of queries affects efficiency.

Completeness queries determine at which point sampled information is used to distinguish states. A low k for determining completeness has the effect that more sequences are used to distinguish states. While theoretically any k can be used, a low k may introduce spurious states in intermediate hypotheses in practice, which affects on equivalence queries. Conversely, a large k may have the effect that a very large amount of samples is required to distinguish states. In practice, a trade-off between the quality of intermediate models and the required number of samples needs to be found.

The efficiency and accuracy of equivalence queries are affected by several factors, including the number of sampled traces and their lengths. In general, these parameters have to be determined in relation to the available sampling budget. Sampling-based equivalence queries potentially allow to formally specify an accuracy by choosing the number of sampled traces in relation to an error bound and confidence level [Ang87]. We refrain from doing so by employing an online conformance-testing approach, which is more efficient than pure, unbiased random sampling.

The refine query attempts to resample traces that have been observed before, but only rarely. It exploits known information to increase the accuracy of intermediate hypotheses. An efficient implementation should adapt to the SUL's responses, by trying to resample any rarely observed trace rather than a specific one, as the probability of resampling a specific trace is generally low.

Relation to other types of queries. Concept learning from samples often assumes the availability of either example queries $EX(f, D)$ [Val84, Kea98] or the weaker statistical queries $STAT(f, D)$ [Kea98], where f is a target concept and D is a distribution over inputs. The former have been used, for instance, in DFA learning [Ang87] and the latter for learning probabilistic finite automata [CG16]. Example queries return a positive example of the target concept [Val84], whereas statistical queries return accurate estimations of probabilities of examples generated by EX [Kea98]. Both queries have in common that they are defined relative to a distribution D over the examples, which remains fixed during learning. For this reason, these queries are not directly applicable in MDP learning, as the distribution of traces generated by an MDP changes depending on the applied scheduler. The refine and equivalence queries shall adapt to and exploit information gained throughout learning. As a result, these queries implicitly apply schedulers that change throughout learning.

We could fix the scheduler at the beginning of learning, for instance, by omitting refine queries and selecting inputs uniformly at random during equivalence queries. In this way, we would use example queries with a fixed distribution. However, this would come at the cost of efficiency and accuracy. Section 7 shows a comparison between L^*_{MDP} and IoAlergia, for which we use a uniformly random sampling regime. There is no direct counterpart of statistical queries in L^*_{MDP}.

5.2. Learner implementation

5.2.1. Sampling-based observation tables

L^*_{MDP} also uses observation tables, therefore we use the same terminology as in Sect. 4. Sampling-based observation tables carry similar information as their exact counterparts, but instead of output distributions in $Dist(O)$, they store non-negative integers denoting observed output frequencies. More concretely, observation tables in the sampling-based setting store functions in $(O \to \mathbb{N}_0)$, from which we estimate probability distributions.

Definition 5.1 (Sampling-based observation table). An observation table is a tuple $\langle S, E, \widehat{T}\rangle$, consisting of a prefix-closed set of traces $S \subseteq \mathcal{TR}$, a suffix-closed set of continuation sequences $E \subseteq \mathcal{CS}$, and a mapping $\widehat{T}\colon (S \cup Lt(S)) \cdot E \to (O \to \mathbb{N}_0)$, where $Lt(S) = \{s \cdot i \cdot o \mid s \in S, i \in I, o \in O\colon \mathbf{fq}(s \cdot i)(o) > 0\}$.

An observation table can be represented by a two-dimensional array, containing rows labelled with elements of S and $Lt(S)$ and columns labelled by E. Each table cell corresponds to a sequence $c = s \cdot e$, where $s \in S \cup Lt(S)$ is the row label of the cell and $e \in E$ is the column label. It stores queried output frequency counts $\widehat{T}(c) = \mathbf{fq}(c)$. To represent the content of rows, we define the function row on $S \cup Lt(S)$ by $row(s)(e) = \widehat{T}(s \cdot e)$. The traces in $Lt(S)$ are input-output-extensions of S which have been observed so far. We refer to traces in $S/Lt(S)$ as short/long traces. Analogously, we refer to rows labelled by corresponding traces as short and long rows.

As in Sect. 4, we identify states with traces reaching these states. These traces are stored in the prefix-closed set S. We distinguish states by their future behaviour in response to sequences in E. We initially set $S = \{L(q_0)\}$, where $L(q_0)$ is the initial output of the SUL, and $E = I$. Long traces, as extensions of access sequences in S, serve to define transitions of hypotheses.

5.2.2. *Compatibility of observations*

As in Sect. 4, observation tables need to be closed and consistent for a hypothesis to be constructed. Unlike before, we do not have exact information to check equivalence of rows. We need to statistically test if rows are different. Therefore, Definition 5.2 gives a condition to determine whether two sequences lead to statistically different observations, meaning that the corresponding output frequency samples come from different distributions. The condition is based on Hoeffding bounds [Hoe63] that are also used by Carrasco and Oncina [CO99]. In Definition 5.3, we further apply this condition in a check for approximate equivalence between cells and extend this check to rows. Using similar terminology as Carrasco and Oncina [CO99], we refer to such checks as compatibility checks and we say that two cells/rows are compatible if we determine that they are not statistically different. These notions of compatibility serve as the basis for slightly adapted definitions of closedness and consistency.

Definition 5.2 (Different). Two test sequences s and s' in $\mathcal{TS}$ produce statistically different output distributions with respect to $f : \mathcal{TS} \rightarrow (O \rightarrow \mathbb{N}_0)$, denoted $\mathit{diff}_f(s, s')$, iff

1. $\mathbf{cq}(s) \wedge \mathbf{cq}(s') \wedge n_1 > 0 \wedge n_2 > 0$[2], where $n_1 = \sum_{o \in O} f(s)(o)$, $n_2 = \sum_{o \in O} f(s')(o)$, and one of the following conditions holds:

2.a. $\exists\, o \in O\colon \neg\, (f(s)(o) > 0 \Leftrightarrow f(s')(o) > 0)$, or

2.b. $\exists\, o \in O\colon \left|\frac{f(s)(o)}{n_1} - \frac{f(s')(o)}{n_2}\right| > \left(\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_2}}\right) \cdot \sqrt{\frac{1}{2}\ln\frac{2}{\alpha}}$, where α specifies the confidence level $(1-\alpha)^2$ for testing each o separately based on a Hoeffding bound [CO99, Hoe63].

Definition 5.3 (Compatible). Two cells labelled by $c = s \cdot e$ and $c' = s' \cdot e'$ are compatible, denoted $\mathbf{compatible}(c, c')$, iff $\neg\, \mathit{diff}_{\widehat{T}}(c, c')$. Two rows labelled by s and s' are compatible, denoted $\mathbf{compatible}_E(s, s')$ iff $\mathit{last}(s) = \mathit{last}(s')$ and the cells corresponding to all $e \in E$ are compatible, i.e. $\forall\, e \in E\colon \mathbf{compatible}(s \cdot e, s' \cdot e)$.

5.2.3. *Compatibility classes*

In Sect. 4, we formed equivalence classes of traces with respect to $\mathbf{eqRow}_E$ to create one hypothesis state per equivalence class. Now we partition rows labelled by S based on compatibility. Compatibility given by Definition 5.3, however, is not an equivalence relation, as it is not transitive in general. As a result, we cannot simply create equivalence classes.

We apply the heuristic implemented by Algorithm 3 to partition S. This heuristic chooses a representative r for each block in the partition based on the amount of information available for r. The available information is computed by the rank function in Algorithm 3, which basically checks how many input-output extensions of r have been sampled. As in equivalence classes, every trace in a block is compatible with its block representative.

Algorithm 3 first assigns a rank to each trace in S. Then, it partitions S by iteratively selecting the trace r with the largest rank and computing a *compatibility class* $cg(r)$ for r. The trace r is the (canonical) representative for s in $cg(r)$, which we denote by $\mathit{rep}(s)$ (Line 11). Each representative r is stored in the set of representative traces R. In contrast to equivalence classes, elements in a compatibility class need not be pairwise compatible and an s may be compatible with multiple representatives, where the unique representative $\mathit{rep}(s)$ of s has the largest rank. However, in the limit $\mathbf{compatible}_E$ based on Hoeffding bounds converges to an equivalence relation [CO99] and therefore compatibility classes are equivalence classes in the limit; see Sect. 6.

Definition 5.4 (Sampling closedness). An observation table $\langle S, E, \widehat{T}\rangle$ is closed if for all $l \in \mathit{Lt}(S)$ there is a representative $s \in R$ with $\mathbf{compatible}_E(l, s)$.

Definition 5.5 (Sampling consistency). An observation table $\langle S, E, \widehat{T}\rangle$ is consistent if for all compatible pairs of short traces s, s' in S and all input-output pairs $i \cdot o \in I \cdot O$, we have that (1) at least one of the extensions has not been observed yet, i.e. $\widehat{T}(s \cdot i)(o) = 0$ or $\widehat{T}(s' \cdot i)(o) = 0$, or (2) both extensions are compatible, i.e. $\mathbf{compatible}_E(s \cdot i \cdot o, s' \cdot i \cdot o)$.

[2] A similar condition is also used in the compatibility checks by Carrasco and Oncina [CO99], which assumes no difference if there are no observations for either s or s'.

Algorithm 3 Creation of compatibility classes

```
1:  for all $s \in S$ do
2:      $\mathrm{rank}(s) \leftarrow \sum_{i \in I} \sum_{o \in O} \widehat{T}(s \cdot i)(o)$
3:  end for
4:  $unpartitioned \leftarrow S$
5:  $R \leftarrow \emptyset$
6:  while $unpartitioned \neq \emptyset$ do
7:      $r \leftarrow m$ where $m \in unpartitioned$ with largest $\mathrm{rank}(m)$
8:      $R \leftarrow R \cup \{r\}$
9:      $cg(r) \leftarrow \{s \in unpartitioned \mid \mathbf{compatible}_E(s, r)\}$
10:     for all $s \in cg(r)$ do
11:         $rep(s) \leftarrow r$
12:     end for
13:     $unpartitioned \leftarrow unpartitioned \setminus cg(r)$
14: end while
```

The adapted notions of closedness and consistency of observation tables are based on compatibility and compatibility classes. Note that the first condition of consistency may be satisfied because of incomplete information.

5.2.4. *Hypothesis construction*

Given a closed and consistent observation table $\langle S, E, \widehat{T} \rangle$, we derive a hypothesis $\mathcal{H} = \mathrm{hyp}(S, E, \widehat{T})$ through the steps below. Note that extensions $s \cdot i \cdot o$ of s in $R \subseteq S$ define transitions. Some extensions may have few observations, such that $\sum_{o \in O} \widehat{T}(s \cdot i)(o)$ is low and $\mathbf{cq}(s \cdot i) = \mathbf{false}$. In case of such uncertainties, we add transitions to a special sink state labelled by "chaos", an output not in the original alphabet[3]. A hypothesis is a tuple $\mathcal{H} = \langle Q_\mathrm{h}, I, O \cup \{\mathrm{chaos}\}, q_{0\mathrm{h}}, \delta_\mathrm{h}, L_\mathrm{h} \rangle$ where:

- representatives for long traces $l \in Lt(S)$ are given by (see Algorithm 3):
  $rep(l) = r$ where $r \in \{r' \in R \mid \mathbf{compatible}_E(l, r')\}$ with largest $\mathrm{rank}(r)$
- $Q_\mathrm{h} = \{\langle last(s), row(s) \rangle \mid s \in R\} \cup \{q_\mathrm{chaos}\}$,
  - for $q = \langle o, row(s) \rangle \in Q_\mathrm{h} \setminus \{q_\mathrm{chaos}\}$: $L_\mathrm{h}(q) = o$
  - for q_chaos: $L_\mathrm{h}(q_\mathrm{chaos}) = \mathrm{chaos}$ and for all $i \in I$: $\delta_\mathrm{h}(q_\mathrm{chaos}, i)(q_\mathrm{chaos}) = 1$
- $q_{0\mathrm{h}} = \langle L(q_0), row(L(q_0)) \rangle$ (note that $L(q_0) \in S$ due to initialisation)
- for $q = \langle last(s), row(s) \rangle \in Q_\mathrm{h} \setminus \{q_\mathrm{chaos}\}$ and $i \in I$ (note that $I \subseteq E$):
  1. If $\neg\, \mathbf{cq}(s \cdot i)$: $\delta(q, i)(q_\mathrm{chaos}) = 1$, i.e. move to chaos
  2. Otherwise estimate a distribution $\mu = \delta_\mathrm{h}(q, i)$ over the successor states:
     for $o \in O$ with $\widehat{T}(s \cdot i)(o) > 0$: $\mu(\langle o, row(rep(s \cdot i \cdot o)) \rangle) = \frac{\widehat{T}(s \cdot i)(o)}{\sum_{o' \in O} \widehat{T}(s \cdot i)(o')}$

5.2.5. *Updating the observation table*

Closedness and consistency. Analogously to Sect. 4, we make observation tables closed by adding new short rows and we establish consistency by adding new columns. We refer to the function implementing that also as MAKECLOSEDANDCONSISTENT. While $L^*_{\mathrm{MDP}^e}$ implemented by Algorithm 2 needs to fill the observation table after executing MAKECLOSEDANDCONSISTENT, this is not required in the sampling-based setting due to the adapted notions of closedness and consistency.

[3]This is inspired by the introduction of chaos states in **ioco**-based learning [VT15]. We also added such a state in our previous work on stochastic automata learning [AT19b].

Trimming the observation table. Observation table size greatly affects learning performance, therefore it is common to avoid adding redundant information [RS93, IHS14]. Due to inexact information, this is hard to apply in a stochastic setting. We instead remove rows and columns via a function TRIM once we are certain that removing them does not change the hypothesis.

First, let us consider removing rows. Given an observation table $\langle S, E, \widehat{T}\rangle$, we remove s and all s' such that $s \ll s'$ from S if:

1. there is exactly one $r \in R$ such that $\mathbf{compatible}_E(s, r)$
2. $s \notin R$ and $\forall\, r \in R : \neg\,(s \ll r)$
3. and $\forall\, s' \in S, i \in I$, with $s \ll s'$: $\mathit{diff}_{\mathbf{fq}}(s' \cdot i, r \cdot i) = \mathbf{false}$, where $r \in R$ such that $\delta^*_{\mathrm{h}}(r) = \delta^*_{\mathrm{h}}(s')$, $\langle \mathit{last}(r), \mathit{row}(r)\rangle = \delta^*_{\mathrm{h}}(r)$, and δ_{h} is the transition relation of $\mathrm{hyp}(S, E, \widehat{T})$.

The first condition is motivated by the observation that if s is compatible with exactly one r, then all extensions of s can be assumed to reach the same states as the extensions of r. In this case, s presumably corresponds to the same SUL state as r, therefore we do not need to store s in the observation table. The other conditions make sure that we do not remove required rows, because of a spurious compatibility check in the first condition. The second condition ensures that we do not remove representatives and the third condition is related to the implementation of equivalence queries. It basically checks for compatibility between the hypothesis $\mathrm{hyp}(S, E, \widehat{T})$ and the frequency information available via *frequency* queries. This is done by checking if an extension s' of s reveals a difference between the output frequencies observed after s' (queried via $\mathbf{fq}$) and the output frequencies observed after the representative r reaching the same hypothesis state as s'. Put differently, we check if s' is a counterexample to equivalence between hypothesis and SUL. Note that removed rows do not affect hypothesis construction.

After we removed all rows matching the above criteria, we check if we can remove columns.[4] The basic intuition behind removing columns is that a continuation sequence e may have been added spuriously to E. This may happen if the sequence e resolves an inconsistency that is present due to inaccurate information. In later rounds, e may not be required anymore. Therefore, we remove every $e \in E$ that can be removed without introducing inconsistencies and without violating suffix-closedness of E.

More concretely, we perform the following steps on an observation table $\langle S, E, \widehat{T}\rangle$. First, we sort E by descending length. Then, we check for every e in the sorted E: (1) is $\langle S, E\backslash\{e\}, \widehat{T}\rangle$ consistent and (2) is $E\backslash\{e\}$ suffix-closed? If both conditions evaluate to $\mathbf{true}$, we remove e by updating the observation tables to $\langle S, E\backslash\{e\}, \widehat{T}\rangle$.

5.2.6. *Learning algorithm*

Algorithm 4 implements L^*_{MDP}. It first initialises an observation table $\langle S, E, \widehat{T}\rangle$ with the initial SUL output as first row and with the inputs I as columns (Line 1). Lines 2 to 5 perform a refine query and then update $\langle S, E, \widehat{T}\rangle$ with output frequency information, which corresponds to output distribution queries in $L^*_{\mathrm{MDP}^e}$. Here, the teacher resamples the only known trace $L(q_0)$. Resampling that trace consists of observing $L(q_0)$, performing some input, and observing another output.

After that, we perform Line 7 to Line 24 until a stopping criterion is reached. We establish closedness and consistency of $\langle S, E, \widehat{T}\rangle$ in Line 10 to build a hypothesis $\mathcal{H}$ in Line 12. After that, we remove redundant rows and columns of the observation table via TRIM in Line 13. Then, we perform an equivalence query, testing for equivalence between SUL and $\mathcal{H}$. If we find a counterexample, we add all its prefix traces as rows to the observation table as in $L^*_{\mathrm{MDP}^e}$. Finally, we sample new system traces via $\mathbf{rfq}$ to gain more accurate information about the SUL (Lines 20 to 23). Once we stop, we output the final hypothesis.

Stopping. $L^*_{\mathrm{MDP}^e}$ and deterministic automata learning usually stop learning once equivalence between the learned hypothesis and the SUL is achieved. In implementations of equivalence queries via testing, equivalence is usually assumed to hold when no counterexample can be found. Here, we employ a different stopping criterion, because equivalence can hardly be achieved via sampling. Furthermore, we may wish to carry on resampling via $\mathbf{rfq}$ although we did not find a counterexample. Resampling may improve the accuracy of a hypothesis such that subsequent equivalence queries reveal counterexamples.

[4]This optimisation was not part of the original conference paper [TAB+19].

Algorithm 4 The main algorithm implementing L^*_{MDP}

```
Input: sampling-based teacher capable of answering $\mathbf{fq}$, $\mathbf{rfq}$, $\mathbf{eq}$ and $\mathbf{cq}$
Output: final learned model hyp$(S, E, \widehat{T})$
1:  $S \leftarrow \{L(q_0)\}$, $E \leftarrow I$, $\widehat{T} \leftarrow \{\}$          ▷ initialise observation table
2:  perform $\mathbf{rfq}(\langle S, E, \widehat{T}\rangle)$          ▷ sample traces for initial observation table
3:  for all $s \in S \cup Lt(S)$, $e \in E$ do
4:      $\widehat{T}(s \cdot e) \leftarrow \mathbf{fq}(s \cdot e)$          ▷ update observation table with frequency information
5:  end for
6:  $round \leftarrow 0$
7:  repeat
8:      $round \leftarrow round + 1$
9:      while $\langle S, E, \widehat{T}\rangle$ not closed or not consistent do
10:         $\langle S, E, \widehat{T}\rangle \leftarrow$ MakeClosedAndConsistent$(\langle S, E, \widehat{T}\rangle)$
11:     end while
12:     $\mathcal{H} \leftarrow$ hyp$(S, E, \widehat{T})$          ▷ create hypothesis
13:     $\langle S, E, \widehat{T}\rangle \leftarrow$ trim$(\langle S, E, \widehat{T}\rangle, \mathcal{H})$          ▷ remove rows that are not needed
14:     $cex \leftarrow \mathbf{eq}(\mathcal{H})$
15:     if $cex \neq none$ then          ▷ we found a counterexample
16:         for all $(t \cdot i) \in prefixes(cex)$ with $i \in I$ do
17:             $S \leftarrow S \cup \{t\}$          ▷ add all prefixes of the counterexample
18:         end for
19:     end if
20:     perform $\mathbf{rfq}(\langle S, E, \widehat{T}\rangle)$          ▷ sample traces to refine knowledge about SUL
21:     for all $s \in S \cup Lt(S)$, $e \in E$ do
22:         $\widehat{T}(s \cdot e) \leftarrow \mathbf{fq}(s \cdot e)$          ▷ update observation table with frequency information
23:     end for
24: until stop$(\langle S, E, \widehat{T}\rangle, \mathcal{H}, round)$
25: return hyp$(S, E, \widehat{T})$          ▷ output final hypothesis
```

Our stopping criterion takes statistical uncertainty[5] in compatibility checks into account. As previously noted, rows may be compatible to multiple other rows. In particular, a row labelled by s may be compatible to multiple representatives. In such a case, we are not certain which state is reached by the trace s. We address this issue by stopping based on the ratio r_{unamb} of unambiguous traces to all traces, which we compute by:

$$r_{\text{unamb}} = \frac{|\{s \in S \cup Lt(S) \colon \text{unambiguous}(s)\}|}{|S \cup Lt(S)|} \text{ where}$$

$$\text{unambiguous}(s) \Leftrightarrow |\{r \in R \colon \mathbf{compatible}_E(s, r)\}| = 1$$

More concretely, we stop if:

1.a. at least $r_{\min}$ rounds have been executed and
1.b. the chaos state q_{chaos} is unreachable and
1.c. and $r_{\text{unamb}} \geq t_{\text{unamb}}$, where t_{unamb} is a user-defined threshold,
or
2.a. alternatively we stop after a maximum number of rounds $r_{\max}$.

[5]This form of uncertainty does not refer to uncertain behaviour in general, but to uncertainty related to decisions based on statistical tests.

5.3. Teacher implementation

In the following, we describe the implementation of each of the four queries provided by the teacher. Recall that we interact with the SUL with semantics M via the two operations **reset** and **step**; see Sect. 5.1. The canonical MDP $can(M) = \mathcal{M} = \langle Q, I, O, q_0, \delta, L\rangle$ has the same observable behaviour as the SUL.

5.3.1. Frequency query

The teacher keeps track of a multiset of sampled system traces $\mathcal{S}$. Whenever a new a trace is added, all its prefixes are added as well, as they have been observed as well. Therefore, we have for $t \in \mathcal{TR}$, $t' \in \mathit{prefixes}(t)\colon \mathcal{S}(t) \leq \mathcal{S}(t')$. The frequency query $\mathbf{fq}(s)\colon O \to \mathbb{N}_0$ for $s \in \mathcal{TS}$ returns output frequencies observed after s:

$$\forall\, o \in O\colon \mathbf{fq}(s)(o) = \mathcal{S}(s \cdot o)$$

5.3.2. Complete query

Trace frequencies retrieved via $\mathbf{fq}$ are generally used to compute empirical output distributions μ following a sequence s in $\mathcal{TS}$, i.e. the learner computes $\mu(o) = \frac{\mathbf{fq}(s)(o)}{\sum_{o' \in O}\mathbf{fq}(s)(o')}$ to approximate $M(s)(o)$. The *complete* query $\mathbf{cq}$ takes a sequence s as input and signals whether s should be used to approximate $M(s)$, for instance, to perform statistical tests[6]. We base $\mathbf{cq}$ on a threshold $n_c > 0$ by defining:

$$\mathbf{cq}(s) = \begin{cases} \mathbf{true} & \text{if } \sum_{o\in O}\mathcal{S}(s\cdot o) \geq n_c \\ \mathbf{true} & \text{if } \exists\, s', o, i\colon s'\cdot o\cdot i \ll s \wedge \mathbf{cq}(s') \wedge \mathcal{S}(s\cdot o) = 0 \\ \mathbf{false} & \text{otherwise} \end{cases}$$

Note that for a complete s, all prefixes of s are also complete. Additionally if $\mathbf{cq}(s)$, then we assume that we have seen all extensions of s. Therefore, we set for each o with $\mathcal{S}(s\cdot o) = 0$ all extensions of $s \cdot o$ to be complete (second clause). The threshold n_c is user-specifiable in our implementation.

5.3.3. Refine query

Refine queries serve the purpose of refining our knowledge about output distributions along previously observed traces. Therefore, we select rarely observed traces and resample them to perform this query. We implemented this through the procedure outlined in Algorithm 5.

First, we build a prefix tree from rarely observed traces (Line 1 and Line 2), where edges are labelled by input-output pairs and nodes are labelled by traces reaching the nodes. This tree is then used for directed online-testing of the SUL via sampleSul (Lines 7 to 20) with the goal of reaching a leaf of the tree. In this way, we create n_{resample} new samples and add them to the multiset of samples $\mathcal{S}$.

5.3.4. Equivalence query

Equivalence queries are often implemented via (conformance) testing in active automata, for instance, using the W-method [Cho78] method for deterministic models. Such testing techniques generally execute some test suite to find counterexamples to conformance between a model and the SUL. In our setup, a counterexample is a test sequence inducing a different output distribution in the hypothesis $\mathcal{H}$ than in the SUL. Since we cannot directly observe those distributions, we perform equivalence queries by applying two strategies to find counterexamples. First, we search for counterexamples with respect to the structure of $\mathcal{H}$ via testing. Secondly, we check for statistical conformance between all traces $\mathcal{S}$ sampled so far and $\mathcal{H}$, which allows us to detect incorrect output distributions.

Note that all traces to the state q_{chaos} are guaranteed to be counterexamples, as its label chaos is not part of the original output alphabet O. For this reason, we do not search for other counterexamples if q_{chaos} is reachable in $\mathcal{H}$. In slight abuse of terminology, we implement this by returning *none* from $\mathbf{eq}(\mathcal{H})$. L^*_{MDP} in Algorithm 4 will then issue further $\mathbf{rfq}$ queries, lowering uncertainty about state transitions, which in turn causes q_{chaos} to be unreachable eventually.

[6]This query serves a similar purpose as the *completeness queries* used by Volpato and Tretmans [VT15].

Algorithm 5 *Refine* query

```
Input: observation table $\langle S, E, \widehat{T}\rangle$, number of requested new traces $n_{\text{resample}}$
Output: updated multiset of traces $\mathcal{S}$
 1: $rare \leftarrow \{s \mid s \in (S \cup Lt(S)) \cdot E : \neg\, \mathbf{cq}(s)\}$          ▷ identify incomplete sequences
 2: $prefixTree \leftarrow$ BUILDPREFIXTREE($rare$)
 3: for $i \leftarrow 1$ to $n_{\text{resample}}$ do                              ▷ collect $n_{\text{resample}}$ new samples
 4:     $newTrace \leftarrow$ SAMPLESUL($prefixTree$)
 5:     $\mathcal{S} \leftarrow \mathcal{S} \uplus \{newTrace\}$
 6: end for
 7: function SAMPLESUL($prefixTree$)
 8:     $node \leftarrow root(prefixTree)$
 9:     $trace \leftarrow \mathbf{reset}$                                  ▷ initialise SUL and observe initial output
10:     loop
11:         $input \leftarrow rSel(\{i \in I \mid \exists\, o \in O, n : node \xrightarrow{i,o} n\})$      ▷ random input
12:         $output \leftarrow \mathbf{step}(i)$                          ▷ execute SUL and observe output
13:         $trace \leftarrow trace \cdot i \cdot o$
14:         if $trace \notin prefixTree$ or $trace$ labels leaf then       ▷ did we leave the tree?
15:             return $trace$
16:         end if
17:         $node' \leftarrow n$ with $node \xrightarrow{i,o} n$                ▷ move in tree
18:         $node \leftarrow node'$
19:     end loop
20: end function
```

Testing of structure. Our goal in testing is to sample a trace of the SUL that is not observable on the hypothesis. For this purpose, we adapted the randomised *transition coverage* testing strategy that we presented in previous work [AT17a, AT19a] from Mealy machines to MDPs. We adapted this strategy rather than the *mutation* strategy, because it is unclear how to adapt the *mutation* strategy and our evaluation showed that *transition coverage* was effective in most cases [AT19a]. It reliably found counterexamples with a low number of tests. Since MDPs produce outputs in states, whereas Mealy machines produce outputs on transitions, we target *state coverage* in MDP-based testing instead of transition coverage.

To test stochastic SULs, we generate test cases that interleave random walks in hypotheses with paths leading to randomly chosen states. By performing many of these tests, we aim at covering hypotheses adequately, while exploring new parts of the SUL's state space through random testing. Due to the stochastic behaviour of the SUL, we perform online testing. The online-testing procedure is outlined in Algorithm 6.

The algorithm takes a hypothesis and *qSched* as input where *qSched* is a mapping from states to schedulers. Given $q \in Q$, *qSched*(q) is a scheduler maximising the probability of reaching q, therefore it selects inputs optimally with respect to the reachability of q. As noted in Sect. 2, there exist memoryless and deterministic schedulers for optimal reachability [FKNP11]. Such schedulers take only the last state in the current execution path into account and they select inputs non-probabilistically. Hence, a scheduler *qSched*(q) is a function $s\colon Q \to I$.

In Algorithm 6, we start by randomly choosing a target state q_{target} from the states reachable from the initial state (Line 3). These states are given by $reachable(Q, q_{\text{curr}})$. Then, we execute the SUL, either with random inputs (Line 6) or with inputs leading to the target (Line 8), which are computed using schedulers. If we observe an output which is not possible in the hypothesis, we return a counterexample (Line 13). Alternatively, we may stop with probability p_{stop} (Line 17). If we reach the target or it becomes unreachable, we simply choose a new target state (Line 20).

For each equivalence query, we repeat Algorithm 6 up to n_{test} times and report the first counterexample that we find, if any. In case we find a counterexample c, we resample it up to n_{retest} times or until $\mathbf{cq}(c)$, to get more accurate information about it. If we do not find a counterexample, we check for conformance between the current hypothesis and the sampled traces $\mathcal{S}$, as described below.

Checking conformance to $\mathcal{S}$. For each sequence $t \cdot i \in \mathcal{TS}$ with $i \in I$ such that $\mathbf{cq}(t \cdot i)$, we check for consistency between the information stored in $\mathcal{S}$ and the current hypothesis $\mathcal{H}$ by evaluating two conditions:

Algorithm 6 State-coverage-based testing for counterexample detection

Input: $\mathcal{H} = \langle Q_h, I, O_h, q_{0h}, \delta_h, L_h \rangle$, schedulers $qSched$
Output: counterexample test sequence $s \in \mathcal{TS}$ or *none*

```
1:  $q_{curr} \leftarrow q_{0h}$                                           ▷ current state
2:  $trace \leftarrow$ reset
3:  $q_{target} \leftarrow rSel(reachable(Q_h, q_{curr}))$                    ▷ choose a target state
4:  loop
5:      if $coinFlip(p_{rand})$ then
6:          $in \leftarrow rSel(I)$                                         ▷ random next input
7:      else
8:          $in \leftarrow qSched(q_{target})$                               ▷ next input leads towards target
9:      end if
10:     $out \leftarrow$ step$(in)$                                          ▷ perform input
11:     $q_{curr} \leftarrow \Delta_h(q_{curr}, in \cdot out)$                    ▷ move in hypothesis
12:     if $q_{curr} = \bot$ then                                          ▷ output not possible in hypothesis
13:         return $trace \cdot in$                                       ▷ return counterexample
14:     end if
15:     $trace \leftarrow trace \cdot in \cdot out$
16:     if $coinFlip(p_{stop})$ then                                        ▷ stop with probability $p_{stop}$
17:         return none
18:     end if
19:     if $q_{curr} = q_{target}$ or $q_{target} \notin reachable(Q_h, q_{curr})$ then
20:         $q_{target} \leftarrow rSel(reachable(Q_h, q_{curr}))$              ▷ choose new target
21:     end if
22: end loop
```

1. Is t observable on $\mathcal{H}$? If it is not, then we determine the longest observable prefix t' of t such that $t' \cdot i' \cdot v = t$, where i' is a single input, and return $t' \cdot i'$ as counterexample from $\mathbf{eq}(\mathcal{H})$. The sequence $t' \cdot i'$ is a counterexample, because at least one of its extensions has been observed ($\mathbf{cq}(t \cdot i) \Rightarrow \mathcal{S}(t) > 0$), but none of its extensions is observable on the hypothesis.
2. If t is observable, we determine $q = \langle o, row(r) \rangle$ reached by t in $\mathcal{H}$, where $r \in R$, and return $t \cdot i$ as counterexample if $diff_{\mathbf{fq}}(t \cdot i, r \cdot i)$ is true. This statistical check approximates the comparison $M(t \cdot i) \neq M(r \cdot i)$, to check if $t \not\equiv_M r$. Therefore, it implicitly checks $M(t \cdot i) \neq H(t \cdot i)$, as $t \equiv_H r$.

If neither of the two equivalence checking strategies, i.e. testing of structure and checking conformance to $\mathcal{S}$, finds a counterexample, we return *none* from the corresponding equivalence query.

6. Convergence of $L^*_{\mathbf{mdp}}$

In the following, we will show that the sampling-based L^*_{MDP} learns a correct MDP. Based on the notion of language identification in grammatical inference [dlH10], we describe our goal as producing an MDP isomorphic to the canonical MDP modelling the SUL with probability one in the limit.

To show identification in the limit, we introduce slight simplifications. First, we disable trimming of the observation table (see Sect. 5.2), thus we do not remove rows or columns. Secondly, we set $p_{\text{rand}} = 1$ for equivalence testing and we do not stop testing at the first detected difference between SUL and hypothesis, but we stop solely based on a $p_{\text{stop}} < 1$. As a result, all input choices are distributed uniformly at random (i.e. equivalence testing does not use schedulers for achieving state coverage) and the length of each test is geometrically distributed with p_{stop}. This is motivated by the common assumption that sampling distributions do not change during learning [dlH10]. Thirdly, we change the function rank in Algorithm 3 to assign ranks based on a lexicographic ordering of traces rather than based on observed frequencies, such that the trace consisting only of the initial SUL output has the highest rank. We actually implemented both types of rank functions and found that the frequency-based function led to better accuracy, but would require more complex proofs. We let the number of samples for learning approach infinity, therefore we do not use a stopping criterion. Finally, we concretely instantiate the complete query $\mathbf{cq}$ by setting $n_c = 1$, since n_c is only relevant for applications in practice.

6.1. Proof structure

We show convergence in two major steps: (1) first we show that the hypothesis structure derived from a sampling-based observation table converges to the hypothesis structure derived from the corresponding observation table with exact information. (2) Then, we show that if counterexamples exist, we will eventually find them. Through that, we eventually arrive at a hypothesis with the same structure as the canonical MDP $\mathit{can}(M)$, where M is the SUL semantics. Given a hypothesis with correct structure, it follows by the law of large numbers that the estimated transition probabilities converge to the true probabilities, thus the hypotheses converge to an MDP isomorphic to $\mathit{can}(M)$.

A key point of the proofs concerns the convergence of the statistical test applied by diff_f, which is based on Hoeffding bounds [Hoe63]. With regard to that, we apply similar arguments as Carrasco and Oncina [CO99, p. 11–13 & Appendix]. Given convergence of diff_f, we also rely on the convergence of the exact learning algorithm $L^*_{\mathrm{MDP}^e}$ discussed in Sect. 4.4. Another important point is that the shortest traces in each equivalence class of $S/{\equiv_M}$ do not form loops in $\mathit{can}(M)$. Hence, there are finitely many such traces. Furthermore, for a given $\mathit{can}(M)$ and some hypothesis MDP, the shortest counterexample has bounded length, therefore it suffices to check finitely many test sequences to check for overall equivalence.

6.1.1. *Definitions and notation*

We show convergence in the limit of the number of sampled system traces n. We take n into account through a data-dependent α_n for the Hoeffding bounds used by diff_f defined in Definition 5.2. More concretely, let $\alpha_n = \frac{1}{n^r}$ for $r > 2$ as used by Mao et al. [MCJ+16], which implies $\sum_n \alpha_n n < \infty$. For the remainder of this section, let $\langle S_n, E_n, \widehat{T}_n \rangle$ be the closed and consistent observation table containing the first n samples stored by the teacher in the multiset $\mathcal{S}_n$. Furthermore, let $\mathcal{H}_n$ be the hypothesis $\mathrm{hyp}(S_n, E_n, \widehat{T}_n)$, let the semantics of the SUL be M and let $\mathcal{M}$ be the canonical MDP $\mathit{can}(M)$. We say that two MDPs have the same structure, if their underlying graphs are isomorphic, thus exact transition probabilities may be different. Now we can state the main theorem on convergence.

Theorem 6.1 (Convergence). Given a data-dependent $\alpha_n = \frac{1}{n^r}$ for $r > 2$, which implies $\sum_n \alpha_n n < \infty$, then with probability one, the hypothesis $\mathcal{H}_n$ is isomorphic to $\mathcal{M}$, except for finitely many n.

Hence, in the limit, we learn an MDP that is minimal with respect to the number of states and output-distribution equivalent to the SUL.

6.1.2. *Access sequences*

The exact learning algorithm $L^*_{\mathrm{MDP}^e}$ presented in Sect. 4 iteratively updates an observation table. Upon termination it arrives at an observation table $\langle S, E, T \rangle$ and the corresponding hypothesis $\mathcal{H} = \mathrm{hyp}(S, E, T) = \langle Q_{\mathrm{h}}, I, O, q_{0\mathrm{h}}, \delta_{\mathrm{h}}, L_{\mathrm{h}} \rangle$. Let $S_{\mathrm{acc}} \subseteq S$ be the set of shortest access sequences leading to states in Q given by $S_{\mathrm{acc}} = \{s \mid s \in S, \nexists s' \in S \colon s' \ll s \wedge s' \neq s \wedge \delta^*_{\mathrm{h}}(q_{0\mathrm{h}}, s) = \delta^*_{\mathrm{h}}(q_{0\mathrm{h}}, s')\}$ (the shortest traces in each equivalence class of $S/\mathbf{eqRow}_E = S/{\equiv_M}$). By this definition, S_{acc} forms a directed spanning tree in the structure of $\mathcal{H}$. There are finitely many different spanning trees for a given hypothesis, therefore there are finitely many different S_{acc}. Hypothesis models learned by $L^*_{\mathrm{MDP}^e}$ are isomorphic to $\mathcal{M}$, thus there are finitely many possible final hypotheses. Let $\overline{S}$ be the finite union of all access sequence sets S_{acc} forming spanning trees in all valid final hypotheses. Let $\overline{L} = \{s \cdot i \cdot o \mid s \in \overline{S}, i \in I, o \in O, M(s \cdot i)(o) > 0\}$ be one-step extensions of $\overline{S}$ with non-zero probability.

Note that for the construction of correct hypotheses in $L^*_{\mathrm{MDP}^e}$, it is sufficient for $\mathbf{eqRow}_E$ to approximate M-equivalence (see Definition 3.2) for traces in $\overline{L}$. Actually, if $\mathbf{eqRow}_E$ is equivalent to M-equivalence for traces in $\overline{L}$, then both are equivalent on all possible traces in $\mathcal{TR}$. Consequently, the approximation of $\mathbf{eqRow}_E$ via $\mathbf{compatible}_E$ needs to hold only for traces in $\overline{L}$.

Proof We now show that

$$\forall\, t_1, t_2 \in \mathcal{TR}\colon \mathbf{eqRow}_E(t_1, t_2) \Leftrightarrow t_1 \equiv_M t_2 \text{ follows from}$$
$$\forall\, t_1', t_2' \in \overline{L}\colon \mathbf{eqRow}_E(t_1', t_2') \Leftrightarrow t_1' \equiv_M t_2'.$$

Direction "⇐": By the definition of $\equiv_M$ and $\mathbf{eqRow}_E$, we have $\mathbf{eqRow}_E(t_1, t_2) \Leftarrow t_1 \equiv_M t_2$ for all $t_1, t_2 \in \mathcal{TR}$ and all $E \subseteq \mathcal{CS}$. Intuitively, two traces in the same M-equivalence class cannot be distinguished by a finite set of continuation sequences.
Direction "⇒": Hence, it remains to show that $\forall\, t_1, t_2 \in \mathcal{TR}\colon \mathbf{eqRow}_E(t_1, t_2) \Rightarrow t_1 \equiv_M t_2$ if $\forall\, t_1', t_2' \in \overline{L}\colon \mathbf{eqRow}_E(t_1', t_2') \Leftrightarrow t_1' \equiv_M t_2'$. We shall prove this by contradiction, thus we assume that there exist $t_1, t_2 \in \mathcal{TR}$ such that $\mathbf{eqRow}_E(t_1, t_2)$, but $t_1 \not\equiv_M t_2$, thus violating the implication.

Let t_1' and t_2' be traces in $\overline{L}$ such that $t_1' \equiv_M t_1$ and $t_2' \equiv_M t_2$. These traces exist, because $\overline{L}$ contains all input-output extensions of traces corresponding to simple paths in the structure of the canonical MDP $\mathcal{M}$. Put differently, all states can be reached by traces in $\overline{L}$. By our assumption and transitivity of $\equiv_M$, we have $t_1' \not\equiv_M t_2'$. Since $t_1', t_2' \in \overline{L}$ and $\mathbf{eqRow}_E(t_1', t_2') \Leftrightarrow t_1' \equiv_M t_2'$ for such traces, it follows that $\neg\, \mathbf{eqRow}_E(t_1', t_2')$.

Since $t_1' \equiv_M t_1$ and $t_2' \equiv_M t_2$, we have $\mathbf{eqRow}_E(t_1', t_1)$ and $\mathbf{eqRow}_E(t_2', t_2)$ (see Direction "⇐"). Due to the transitivity of $\mathbf{eqRow}_E$ and the assumption that $\mathbf{eqRow}_E(t_1, t_2)$ holds, we can deduce $\mathbf{eqRow}_E(t_1', t_2')$. This is a contradiction to the paragraph above. Hence, our assumption must be wrong. There does not exist a pair of traces $t_1, t_2 \in \mathcal{TR}$ such that $\mathbf{eqRow}_E(t_1, t_2)$, but $t_1 \not\equiv_M t_2$. □

6.2. Hoeffding-bound-based difference check

Before discussing hypothesis construction, we briefly discuss the Hoeffding-bound-based test applied by diff_f. Recall that for two test sequences s and s', we test for each $o \in O$ if the probability p of observing o after s is different than the probability p' of observing o after s'. This is implemented through:

$$\exists\, o \in O\colon \left|\frac{f(s)(o)}{n_1} - \frac{f(s')(o)}{n_2}\right| > \left(\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_2}}\right) \cdot \sqrt{\frac{1}{2}\ln\frac{2}{\alpha}} = \epsilon_\alpha(n_1, n_2)$$

As pointed out by Carrasco and Oncina [CO99, pp. 11–13 and Appendix], this test works with a confidence level above $(1-\alpha)^2$ and for large enough n_1 and n_2, it tests for difference and equivalence of p and p'. More concretely, for convergence, n_1 and n_2 must be such that $2\epsilon_\alpha(n_1, n_2)$ is smaller than the smallest absolute difference between any two different p and p'. As our data-dependent α_n decreases only polynomially, $\epsilon_\alpha(n_1, n_2)$ tends to zero for increasing n_1 and n_2. Hence, the test implemented by diff_f converges to an exact comparison between p and p'.

In the remainder of the paper, we ignore Condition 2.a for diff_f, which checks if the sampled distributions have the same support. By applying a data-dependent α_n, as defined above, Condition 2.b converges to an exact comparison between output distributions, thus 2.a is a consequence of 2.b in the limit. Therefore, we consider only the Hoeffding-bound-based tests of Condition 2.b.

6.3. Hypothesis construction

Theorem 6.2 (Compatibility convergence). Given α_n such that $\sum_n \alpha_n n < \infty$, then with probability one: $\mathbf{compatible}_E(s, s') \Leftrightarrow \mathbf{eqRow}_E(s, s')$ for all traces s, s' in $\overline{L}$, except for finitely many n.

Proof Let A_n be the event that $\mathbf{compatible}_E(s, s') \not\Leftrightarrow \mathbf{eqRow}_E(s, s')$ and $p(A_n)$ be the probability of this event. In the following, we derive a bound for $p(A_n)$ based on the confidence level of applied tests in Definition 5.2 which is above $(1-\alpha_n)^2$ [CO99]. An observation table stores $|\, S \cup Lt(S)\,| \cdot |\, E\,|$ cells, which gives us an upper bound on the number of tests performed for computing $\mathbf{compatible}_E(s, s')$ for two traces s and s'. However, note that cells do not store unique information; multiple cells may correspond to the same test sequence in $\mathcal{TS}$, therefore it is simpler to reason about the number of different tests in calls to $\mathit{diff}_{\widehat{\mathcal{T}}}(c, c') = \mathit{diff}_{\mathbf{fq}}(c, c')$ with respect to $\mathcal{S}_n$. A single call to $\mathit{diff}_{\mathbf{fq}}$ involves either 0 or $|\, O\,|$ tests. We apply tests only if we have observed both c and c' at least once, therefore we perform at most $2 \cdot |\, O\,| \cdot n$ different tests for all pairs of observed test sequences. The event

A_n may occur if any test produces an incorrect result, i.e. it yields a Boolean result different from the comparison between the true output distributions induced by c and c'. This leads to $p(A_n) \leq 2 \cdot \mid O \mid \cdot n \cdot (1-(1-\alpha_n)^2)$, which implies $p(A_n) \leq 4 \cdot \mid O \mid \cdot n \cdot \alpha_n$. By choosing α_n such that $\sum_n \alpha_n n < \infty$, we have $\sum_n p(A_n) < \infty$ and we can apply the Borel-Cantelli lemma like Carrasco and Oncina [CO99], which states A_n happens only finitely often. Hence, there is an N_{comp} such that for $n > N_{\text{comp}}$, we have $\mathbf{compatible}_E(s, s') \Leftrightarrow \mathbf{eqRow}_E(s, s')$ with respect to $\mathcal{S}_n$. □

Lemma 6.1 Under the assumed uniformly randomised equivalence testing strategy, it holds for every $s \cdot i \cdot o \in \overline{L}$: $\mathcal{S}_n(s \cdot i \cdot o) > 0$ after finitely many n.

Proof Informally, we will eventually sample all traces $l \in \overline{L}$. The probability p_L of sampling $l = o_0 \cdot i_1 \cdot o_1 \cdots o_n \cdot i \cdot o$ during a test, where $l[\leq k]$ is the prefix test sequence of l of length k, is given by (note that we may sample l as a prefix of another sequence):

$$p_L = \frac{1}{\mid I \mid^{n+1}} M(l[\leq 1])(o_1) \cdots M(l[\leq n])(o_n) \cdot M(l[\leq n+1])(o)(1-p_{\text{stop}})^n$$

Since every $l \in \overline{L}$ is observable, we have $M(l[\leq 1])(o_1) \cdots M(l[\leq n])(o_n) \cdot M(t[\leq n+1])(o) > 0$, thus $p_L > 0$. Hence, there is a finite N_L such that for all $s \cdot i \cdot o \in \overline{L}$: $\mathcal{S}_n(s \cdot i \cdot o) > 0$ for $n > N_L$. □

Lemma 6.2 If $\mathbf{compatible}_E(s, s') \Leftrightarrow \mathbf{eqRow}_E(s, s')$, then the set of representatives R computed by Algorithm 3 for the closed and consistent observation table $\langle S_n, E_n, \widehat{T}_n \rangle$ is prefix-closed.

Proof Recall that we assume the function rank to impose a lexicographic ordering on traces, thus all ranks are unique. This simplifies showing prefix-closedness of R, which we do by contradiction. Assume that R is not prefix-closed. In that case, there is a trace r of length n in R with a prefix r_p of length $n-1$ that is not in R. Since $r_p \notin R$ and because the representative $rep(r_p)$ has the largest rank in its class $cg(r_p)$, we have $r_p \neq rep(r_p)$ and $\text{rank}(r_p) < \text{rank}(rep(r_p))$.

As S_n is prefix-closed and $R \subseteq S_n$, we have $r_p \in S_n$. Let $i \in I$ and $o \in O$ be such that $r_p \cdot i \cdot o = r$. Algorithm 3 enforces $\mathbf{compatible}_E(r_p, rep(r_p))$ and due to consistency, we have that $\mathbf{compatible}_E(r_p \cdot i \cdot o, rep(r_p) \cdot i \cdot o) = \mathbf{compatible}_E(r, rep(r_p) \cdot i \cdot o)$. Since r is a representative in R, $rep(r_p) \cdot i \cdot o \in cg(r)$. Representatives r have the largest rank in their compatibility class $cg(r)$ and $r \neq rep(r_p) \cdot i \cdot o$, thus $\text{rank}(r) > \text{rank}(rep(r_p) \cdot i \cdot o)$.

In combination we have that

$$\text{rank}(r_p) < \text{rank}(rep(r_p)) \text{ and also}$$
$$\text{rank}(r) = \text{rank}(r_p \cdot i \cdot o) > \text{rank}(rep(r_p) \cdot i \cdot o).$$

This is a contradiction given the lexicographic ordering on traces imposed by rank. Consequently, R must be prefix-closed under the premises of Lemma 6.2. □

Lemma 6.3 Let $\langle S_n, E_n, T_n \rangle$ be the *exact* observation table corresponding to the sampling-based observation table $\langle S_n, E_n, \widehat{T}_n \rangle$, i.e. $T_n(s) = \mathbf{odq}(s)$ for $s \in (S_n \cup Lt(S_n)) \cdot E$. Then, $T_n(r \cdot i)(o) > 0 \Leftrightarrow \widehat{T}_n(r \cdot i)(o) > 0$ for $r \in R$, $i \in I$, $o \in O$ after finitely many n.

Proof First, we show for prefix-closed R (Lemma 6.2) that $R \subseteq \overline{S}$, if $\mathbf{compatible}_E(s, s') \Leftrightarrow \mathbf{eqRow}_E(s, s')$. $\overline{S}$ contains all traces corresponding to simple paths of $can(M)$, therefore we show by contradiction that no $r \in R$ forms a cycle in $can(M)$.

Assume that $r \in R$ forms a cycle in $can(M)$, i.e. it visits states multiple times. We can split r into three parts $r = r_p \cdot r_c \cdot r_s$, where $r_p \in \mathcal{TR}$ such that r_p and $r_p \cdot r_c$ reach the same state, r_c is non-empty, and $r_s \in (I \times O)^*$ is the longest suffix such that starting from $\delta^*(r_p)$, r_s visits every state of $can(M)$ at most once. As R is prefix-closed, R includes r_p and $r_p \cdot r_c$ as well. The traces r_p and $r_p \cdot r_c$ reach the same state in $can(M)$, thus we have $r_p \equiv_M r_p \cdot r_c$ which implies $\mathbf{eqRow}_E(r_p, r_p \cdot r_c)$ and $\mathbf{compatible}_E(r_p, r_p \cdot r_c)$.

By Algorithm 3 all $r \in R$ are pairwise not compatible with respect to $\mathbf{compatible}_E$, thus Algorithm 3 ensures $\neg\, \mathbf{compatible}_E(r_p, r_p \cdot r_c)$ for $r_p \in R$ and $r_p \cdot r_c \in R$. This leads to a contradiction, therefore our assumption is false and we can deduce that no r visits a state of $can(M)$ more than once. As a consequence, it holds that $R \subseteq \overline{S}$.

Hence, every observable $r_l = r \cdot i \cdot o$ for $r \in R$, $i \in I$ and $o \in O$ is in $\overline{L}$, as $\overline{L}$ includes all observable extensions of $\overline{S}$. By Lemma 6.1, we will sample r_l eventually, i.e. $\widehat{T}_n(r \cdot i)(o) > 0$ and therefore $T_n(r \cdot i)(o) > 0 \Leftrightarrow \widehat{T}_n(r \cdot i)(o) > 0$ after finitely many n. □

Lemma 6.4 The chaos state q_{chaos} is not reachable in $\mathcal{H}_n$, except for finitely many n.

Proof We add a transition from state $q = \langle last(r), row(r) \rangle$ with input i to q_{chaos} if $\mathbf{cq}(r \cdot i) = \mathbf{false}$. As we consider $n_c = 1$, $\mathbf{cq}(r \cdot i) = \mathbf{true}$ if there is an o such that $\widehat{T}_n(r \cdot i)(o) > 0$. Lemma 6.3 states that $\widehat{T}_n(r \cdot i)(o) > 0$ for any observable $r \cdot i \cdot o$ after finitely many n. Thus, Lemma 6.3 implies $\mathbf{cq}(r \cdot i) = \mathbf{true}$ for all $r \in R$ and $i \in I$, therefore the chaos is unreachable in $\mathcal{H}_n$, except for finitely many n. □

Corollary 6.1 Let $\langle S_n, E_n, T_n \rangle$ be the exact observation table corresponding to the sampling-based observation table $\langle S_n, E_n, \widehat{T}_n \rangle$, i.e. $T_n(s) = \mathbf{odq}(s)$ for $s \in (S_n \cup Lt(S_n)) \cdot E$. Then there exists a finite N_{struct} such that the exact hypothesis $\text{hyp}(S_n, E_n, T_n)$ has the same structure as $\mathcal{H}_n$ for $n > N_{\text{struct}}$.

By combining Theorem 6.2, Lemma 6.3 and Lemma 6.4, it follows that, after finitely many n, hypotheses created in the sampling-based setting have the same structure as in the exact setting.

6.4. Equivalence queries

Theorem 6.3 (Convergence of equivalence queries). Given α_n such that $\sum_n \alpha_n n < \infty$, an observation table $\langle S_n, E_n, \widehat{T}_n \rangle$ and a hypothesis $\mathcal{H}_n$, then with probability one, $\mathcal{H}_n$ has the same structure as $\mathcal{M}$ or we find a counterexample to equivalence, except for finitely many n.

In the following, we introduce lemmas to prove Theorem 6.3, but first we recap relevant details of equivalence queries. According to Corollary 6.1, there is an N_{struct} such that $\mathcal{H}_n$ has the same structure as in the exact setting and $\mathbf{compatible}_E(s, s') \Leftrightarrow \mathbf{eqRow}_E(s, s')$ for $n > N_{\text{struct}}$. Therefore, we assume $n > N_{\text{struct}}$ for the following discussion of counterexample search through the implemented equivalence queries $\mathbf{eq}$. Let H_n be the semantics of $\mathcal{H}_n$. Recall that we apply two strategies for checking equivalence:

1. Random testing with a uniformly randomised scheduler ($p_{\text{rand}} = 1$): this form of testing can find traces $s \cdot o$, with $s \in \mathcal{TS}$ and $o \in O$, such that $H(s)(o) = 0$ and $M(s)(o) > 0$. While this form of search is coarse, we store all sampled traces in $\mathcal{S}_n$ that is used by our second counterexample search strategy for performing a fine-grained analysis.
2. Checking conformance with $\mathcal{S}_n$: for all observed test sequences, we statistically check for differences between output distributions in $\mathcal{H}_n$ and distributions estimated from $\mathcal{S}_n$ through applying $\mathit{diff}_{\mathbf{fq}}$. Applying that strategy finds counterexample sequences $s \in \mathcal{TS}$ such that $M(s) \neq \bot$ (as s must have been observed) and approximately $M(s) \neq H(s)$.

Case 1. In the case that the hypothesis $\mathcal{H}_n$ and $\mathcal{M}$ have the same structure and $n > N_{\text{struct}}$, such that $\mathbf{eqRow}_E(s, s') \Leftrightarrow \mathbf{compatible}_E(s, s')$, we may still find counterexamples that are spurious due to inaccuracies. Therefore, we will show that adding a prefix-closed set of traces to the set of short traces S_n does not change the hypothesis structure, as this is performed by Algorithm 4 in response to counterexamples returned by $\mathbf{eq}$.

Lemma 6.5 If $\mathcal{H}_n$ has the same structure as $\mathcal{M}$ and $n > N_{\text{struct}}$, then adding a prefix-closed set of observable traces S_t to S_n will neither introduce closedness-violations nor inconsistencies, hence $\langle S_n \cup S_t, E_n, \widehat{T}_n \rangle$ is closed and consistent. Consequently, the hypothesis structure does not change, thus $\mathcal{H}_n$ has the same structure as $\text{hyp}(S_n \cup S_t, E_n, \widehat{T}_n)$.

Proof Let t be a trace in S_t and $q_t = \delta^*_{\text{h}}(t)$ be the hypothesis state reached by t, which exists because $\mathcal{H}_n$ has the same structure as $\mathcal{M}$. Let $t_s \in S_n$ be a short trace also reaching q_t. Since $\mathcal{M}$ and $\mathcal{H}_n$ have the same structure, t and t_s also reach the same state of $\mathcal{M}$, therefore $t \equiv_M t_s$ (by reaching the same state both traces lead to the same future behaviour), which implies $\mathbf{eqRow}_E(t, t_s)$. With $n > N_{\text{struct}}$, we have $\mathbf{compatible}_E(t, t_s)$. By the same reasoning, we have $\mathbf{compatible}_E(t \cdot i \cdot o, t_s \cdot i \cdot o)$ for any $i \in I$, $o \in O$ with $M(t \cdot i)(o) > 0$; which is the condition for consistency of observation tables, therefore adding t to S_n leaves the observation tables consistent.

Furthermore because $\langle S_n, E_n, \widehat{T}_n \rangle$ is closed, there exists a $t'_s \in S_n$, with $\mathbf{compatible}_E(t_s \cdot i \cdot o, t'_s)$. Since $\mathbf{compatible}_E(t \cdot i \cdot o, t_s \cdot i \cdot o)$ and because $\mathbf{compatible}_E$ is transitive for $n > N_{\text{struct}}$, we have $\mathbf{compatible}_E(t \cdot i \cdot o, t'_s)$. Hence, adding t as to S_n does not violate closedness, because for each observable extension of t, there exists a compatible short trace t'_s. □

Case 2. If $\mathcal{H}_n$ does not have the same structure as $\mathcal{M}$ and $n > N_{\text{struct}}$, then $\mathcal{H}_n$ has fewer states than $\mathcal{M}$. This follows from Lemma 4.6 given that $\mathcal{H}$ is consistent with $\widehat{T}_n$ and $\mathbf{compatible}_E(s, s') \Leftrightarrow \mathbf{eqRow}_E(s, s')$. Since $\mathcal{M}$ is minimal with respect to the number of states, $\mathcal{H}_n$ and $\mathcal{M}$ are not equivalent, thus a counterexample to observation equivalence exists and we are guaranteed to find any such counterexample after finitely many samples.

Lemma 6.6 If $\mathbf{compatible}_E(s, s') \Leftrightarrow \mathbf{eqRow}_E(s, s')$ for traces s and s' in S_n, then the hypothesis $\mathcal{H}_n$ derived from $\langle S_n, E_n, \widehat{T}_n\rangle$ has the minimal number of states among all MDPs consistent with $\widehat{T}_n$ with respect to $\mathit{diff}_{\widehat{T}_n}$.

Proof Recall that for a given observation table $\langle S, E, T\rangle$, the exact learning algorithm $L^*_{\text{MDP}^e}$ derives the smallest hypothesis consistent with T. By Corollary 6.1, $\mathcal{H}_n$ has the same structure as the smallest MDP consistent with T. As $\mathit{diff}_{\widehat{T}_n}$ does not produce spurious results for $n > N_{\text{struct}}$ (Theorem 6.2), $\mathcal{H}_n$ has the same structure as the smallest MDP consistent with $\widehat{T}_n$ with respect to $\mathit{diff}_{\widehat{T}_n}$, therefore the number of states of $\mathcal{H}_n$ is minimal. □

Lemma 6.7 Let n_q be the number of states of $\mathcal{M}$, $C = \bigcup_{i=0}^{n_q^2+1}(O \times I)^i$ and $C^{\text{obs}} = \{c \mid c \in C \colon M(c) \neq \bot\}$. For any other MDP $\mathcal{M}'$ with at most n_q states and semantics M', we have $\forall\, c \in C^{\text{obs}} : M(c) = M'(c)$ iff $\mathcal{M} \equiv_{\text{od}} \mathcal{M}'$.

Hence, there is a finite set C^{obs} of sequences with lengths bounded by $n_q^2 + 1$ such that by testing all sequences in C^{obs}, we can check equivalence with certainty.

Proof Let $\mathcal{M}$ and $\mathcal{M}'$ with states Q and Q' be defined as above, i.e. $\mid Q \mid = n_q$ and $\mid Q' \mid \leq n_q$, and let reachQSeq$(t) \in (Q \times Q')^*$ be the sequence of state pairs visited along a trace t by $\mathcal{M}$ and $\mathcal{M}'$, respectively. $\mathcal{M} \equiv_{\text{od}} \mathcal{M}'$ iff for all $t \in \mathcal{TR}$ and $i \in I$, we have $M(t \cdot i) = M'(t \cdot i)$. If the length of $t \cdot i$ is at most $n_q^2 + 1$, then $t \cdot i \in C$. Otherwise, reachQSeq(t) contains duplicated state pairs, because $\mid Q \times Q' \mid \leq n_q^2$. For t longer than n_q^2, we can remove loops on $Q \times Q'$ from t to determine a trace t' of length at most n_q^2 such that reachQSeq$(t)[\mid t \mid]$ = reachQSeq$(t')[\mid t' \mid]$, i.e. such that t and t' reach the same state pair. Since t reaches the same state as t' in $\mathcal{M}$ and in $\mathcal{M}'$, we have $M(t \cdot i) = M(t' \cdot i)$ and $M'(t \cdot i) = M'(t' \cdot i)$, thus $M(t \cdot i) = M'(t \cdot i) \Leftrightarrow M(t' \cdot i) = M'(t' \cdot i)$. Consequently for all $t \cdot i \in \mathcal{TR} \cdot I$: either $t \cdot i \in C$, or there is a $t' \cdot i \in C$ leading to an equivalent check between $\mathcal{M}$ and $\mathcal{M}'$.

We further restrict C to C^{obs} by considering only observable test sequences in C. This restriction is justified by Lemma 4.1. In summary:

$$\begin{aligned}\mathcal{M} \equiv_{\text{od}} \mathcal{M}' &\Leftrightarrow \forall\, c \in \mathcal{TS}\colon M(c) = M'(c)\\ &\Leftrightarrow \forall\, c \in C\colon M(c) = M'(c)\\ &\Leftrightarrow \forall\, c \in C^{\text{obs}}\colon M(c) = M'(c)\end{aligned}$$

□

Lemma 6.8 Under the randomised testing strategy with $p_{\text{rand}} = 1$ and $p_{\text{stop}} < 1$, all c in C^{obs} have non-zero probability to be observed.

Proof Due to $p_{\text{rand}} = 1$ and $p_{\text{stop}} < 1$ we apply uniformly randomised inputs during testing and each test has a length that is distributed dependent on p_{stop}. Let $c = o_0 \cdot i_1 \cdot o_1 \cdots o_{n-1} \cdot i_n$ be a sequence in C^{obs} with $c[\leq k]$ being its prefix of length k, then the probability p_c of observing c is (note that we may observe c as a prefix of another sequence):

$$p_c = \frac{1}{\mid I \mid^n} M(c[\leq 1])(o_1) \cdot M(c[\leq 2])(o_2) \cdots M(c[\leq n-1])(o_{n-1}) \cdot (1 - p_{\text{stop}})^{n-1}$$

By definition of C^{obs}, we have $M(c[\leq j])(o_j) > 0$ for all indexes j and c in C^{obs}, therefore $p_c > 0$. $\square$

In every round of L^*_{MDP}, we check for conformance between $\mathcal{S}_n$ and the hypothesis $\mathcal{H}_n$ and return a counterexample, if we detect a difference via $\mathit{diff}_{\mathbf{fq}}$. Since we apply $\mathit{diff}_{\mathbf{fq}}$, we follow a similar reasoning as for the convergence of hypothesis construction. Here, we approximate $M(c) \neq H(c)$ for $c \in \mathcal{TS}$ by $\mathit{diff}_{\mathbf{fq}}(t \cdot i, r \cdot i)$, where $c = t \cdot i$ for a trace t, an input i, and $r \in R$ given by the hypothesis state $\langle \mathit{last}(r), \mathit{row}(r) \rangle$ reached by t.

Lemma 6.9 Given α_n such that $\sum_n \alpha_n n < \infty$, then with probability one $M(c) \neq H(c) \Leftrightarrow \mathit{diff}_{\mathbf{fq}}(t \cdot i, r \cdot i)$ for $c = t \cdot i \in C^{\text{obs}}$ and r as defined above, except for finitely many n.

Proof We use the identity $H(t \cdot i) = H(r \cdot i)$ for traces t and r and inputs i, which holds because t and r reach the same state in the hypothesis $\mathcal{H}$. Applying that, we test for $M(t \cdot i) \neq H(t \cdot i)$ by testing $M(t \cdot i) \neq H(r \cdot i)$ via $\mathit{diff}_{\mathbf{fq}}(t \cdot i, r \cdot i)$. We perform $| O |$ tests for each unique observed sequence c, therefore we apply at most $n \cdot | O |$ tests. Let B_n be the event that any of these tests is wrong, that is, $M(t \cdot i) \neq H(r \cdot i) \not\Leftrightarrow \mathit{diff}_{\mathbf{fq}}(t \cdot i, r \cdot i)$ for at least one observed $c = t \cdot i$. Due to the confidence level greater than $(1 - \alpha_n)^2$ of the tests, the probability $p(B_n)$ of B_n is bounded by $p(B_n) \leq n \cdot | O | \cdot (1 - (1 - \alpha_n)^2) \leq 2 \cdot n \cdot | O | \cdot \alpha_n$. By choosing α_n such that $\sum_n \alpha_n n < \infty$, we can apply the Borel-Cantelli lemma as in the proof of Theorem 6.2. Hence, B_n happens only finitely often, thus there is an N_1 such that for all $n > N_1$ we have $M(t \cdot i) \neq H(r \cdot i) \Leftrightarrow \mathit{diff}_{\mathbf{fq}}(t \cdot i, r \cdot i)$ for all observed $c = t \cdot i \in C^{\text{obs}}$. Furthermore, the probability of observing any c of the finite set C^{obs} during testing is greater than zero (Lemma 6.8), thus there is a finite N_2 such that $\mathcal{S}_n$ contains all $c \in C^{\text{obs}}$ for $n > N_2$. Consequently, there is an N_{cex}, such that Lemma 6.9 holds for all $n > N_{\text{cex}}$. $\square$

Lemma 6.6 states that hypotheses $\mathcal{H}_n$ are minimal after finitely many n, thus all potential counterexamples are in C^{obs} (Lemma 6.7). From Lemma 6.9, it follows that we will identify a counterexample in C^{obs} if one exists. Combining that with Lemma 6.5 concludes the proof of Theorem 6.3.

6.5. Putting everything together

We have established that after finitely many n, the sampling-based hypothesis $\mathcal{H}_n$ has the same structure as in the exact setting (Corollary 6.1). Therefore, certain properties of the exact learning algorithm $L^*_{\text{MDP}^e}$ hold for the sampling-based L^*_{MDP} as well. The derived hypotheses are therefore minimal, i.e. they have at most as many states as $\mathcal{M}$. As with $L^*_{\text{MDP}^e}$, adding a non-spurious counterexample to the trace set S_n introduces at least one state in the derived hypotheses. Furthermore, we have shown that equivalence queries return non-spurious counterexamples, except for finitely many n (Theorem 6.3). Consequently, after finite n we arrive at a hypothesis $\mathcal{H}_n$ with the same structure as $\mathcal{M}$. We derive transition probabilities by computing empirical means, thus by the law of large numbers these estimated probabilities converge to the true probabilities. Hence, we learn a hypothesis $\mathcal{H}_n$ isomorphic to the canonical MDP $\mathcal{M}$ in the limit as stated by Theorem 6.1.

More efficient parameters. So far, we discussed a particular parametrisation of L^*_{MDP}. Among others, we used uniformly random input choices for equivalence testing with $p_{\text{rand}} = 1$, and instantiated $\mathbf{cq}$ to accept samples as complete after only $n_c = 1$ observation. This simplified the proof, but is inefficient in practical experiments. Completely random testing is generally inefficient for large systems and a small n_c may lead to spurious states in intermediate hypotheses. However, the arguments based on $n_c = 1$, such as Lemma 6.3 and Lemma 6.4, are easily extended to small constant values of n_c: Since the samples are collected independently, any observation that occurs at least once after a finite number of steps also occurs at least n_c times after a finite number of steps.

7. Experiments

In active automata learning, our goal is generally to learn a model equivalent to the true model of the SUL. This changes in the stochastic setting, where we want to learn a model close to the true model, because equivalence can hardly be achieved. In this section, we evaluate the sampling-based L^*_{MDP} and compare it to the passive IoAlergia [MCJ+16], by learning several models with both techniques. In each case, we treat the known true MDP model $\mathcal{M}$ as a black box for learning and measure similarity to this model using two criteria:

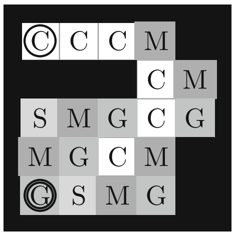

Fig. 3. The first gridworld used in our evaluation

1. *bisimilarity distance:* we compute the discounted bisimilarity distance between the true model $\mathcal{M}$ and the learned MDPs [BBLM13b, BBLM13a]. We adapted this distance measure from MDPs with rewards to labelled MDPs, by defining a distance of 1 between states with different labels.
2. *probabilistic model-checking:* we compute and compare maximal probabilities of manually defined temporal properties with all models using Prism 4.4 [KNP11]. The difference between probabilistic model-checking results for $\mathcal{M}$ and the learned MDPs serves as a similarity measure.

We performed experiments with the gridworld model used in the conference version of this article [TAB+19] and with two of the models that we used in the evaluation of our work on probabilistic black-box reachability checking [AT19b], the slot machine, and the shared coin consensus protocol. In addition to that, we performed experiments with another larger gridworld model. To account for the probabilistic nature of these experiments, we repeated each of them ten times. In the following, we report mean measurement values unless otherwise noted. All tables include mean values along with the corresponding standard deviations separated by $\pm$. Experimental results, the examined models, and the implementation of L^*_{MDP} can be found in the evaluation material [Tap20].

7.1. Measurement setup

Like Mao et al. [MCJ+16] did in one of two configurations, we configure IoAlergia with a data-dependent significance parameter for the compatibility check, by setting $\epsilon_N = \frac{10000}{N}$, where N is the total combined length of all traces used for learning. This parameter serves a purpose analogous to the α parameter for the Hoeffding bounds used by L^*_{MDP}. IoAlergia showed good performance under this configuration in the experiments discussed in this section. In contrast to that, we observed that L^*_{MDP} generally shows better performance with non-data-dependent α, therefore we set $\alpha = 0.05$ for all experiments. We sample traces for IoAlergia with a length geometrically distributed with parameter p_l and inputs chosen uniformly at random, like Mao et al. [MCJ+16]. The number of traces is chosen such that IoAlergia and L^*_{MDP} learn from approximately the same amount of data, where data is quantified by the combined length of all traces.

We implemented L^*_{MDP} and IoAlergia using Java 8. In addition to our Java implementations, we use Prism 4.4 [KNP11] for probabilistic model-checking and an adaptation of the MDPDist library [BBLM] to labelled MDPs for computing bisimilarity distances. We performed the experiments with a Lenovo Thinkpad T450 running Xubuntu Linux 18.04 with 16 GB RAM and an Intel Core i7-5600U CPU with 2.6 GHz.

7.2. Experiments with first gridworld

The basis for our first set of experiments is the gridworld that we used in the conference paper [TAB+19]. It is shown in Fig. 3. In this example, a robot moves around in a world of tiles of different terrains. The terrains are denoted by different shades of grey and uppercase letters. Generally, the robot can move into one of four directions, but it cannot move into walls, which are denoted by black borders and tiles. Moreover, it may make errors in movement depending on the target terrain. For instance, it may move south west instead of south, i.e., probabilistic errors allow to move diagonally. Each target terrain has a unique positive error probability, except for C, which has an error probability of zero, thus moving to a C tile is always successful. There is a unique starting tile with a single circle and a goal tile with a double circle. While moving, the robot can only observe the terrain on which it is standing, whether it bumped into a wall and whether it is at the goal tile. Therefore, our aim is basically to learn a map of the gridworld. The minimal true model of this gridworld has 35 different states.

Table 3. Results for learning the first gridworld example

	true model	L^*_{MDP}	IoAlergia
# outputs	–	2122580.2 ± 884024.02	2123799.9 ± 886998.79
# traces	–	354637.1 ± 163252.4	265323.6 ± 110502.47
time [s]	–	179.43 ± 158.6	17.99 ± 8.91
# states	35	35 ± 0	19.9 ± 1.73
$\delta_{0.9}$	–	0.1730 ± 0.022	0.5353 ± 0.0171
$\mathbb{P}_{\max}(F^{\leq 11}(\text{goal}))$	0.9622	0.9603 ± 0.0058	0.2110 ± 0.0489
$\mathbb{P}_{\max}(\neg\ \text{G}\ U^{\leq 14}(\text{goal}))$	0.6499	0.6498 ± 0.0142	0.1724 ± 0.0448
$\mathbb{P}_{\max}(\neg\ \text{S}\ U^{\leq 16}(\text{goal}))$	0.6913	0.7006 ± 0.0095	0.1842 ± 0.0217

For L^*_{MDP}, we set the sampling parameters to $n_{\text{resample}} = n_{\text{retest}} = 300$, $n_{\text{test}} = 50$, $p_{\text{stop}} = 0.25$ and $p_{\text{rand}} = 0.25$. As stopping parameters served $t_{\text{unamb}} = 0.99$, $r_{\min} = 500$ and $r_{\max} = 4000$. Finally, we set the parameter p_l for IoAlergia's eometric trace length distribution to 0.125.

Results. Table 3 shows the measurement results for learning the first gridworld. Active learning stopped after 1239.3 rounds on average, sampling 354637.1 traces (Row 2) with a combined number of outputs of 2122580.2 (Row 1). We see a large variation in the sampled data, as the length of the repeated experiments varied greatly. Our stopping heuristic required a varying number of rounds to detect convergence. The bisimilarity distance discounted with $\lambda = 0.9$ to the true model is 0.1730 for L^*_{MDP} and 0.5353 for IoAlergia (Row 5); thus it can be assumed that model checking the L^*_{MDP} models produces more accurate results. This is indeed true for our three evaluation queries in the last three rows. These model-checking queries ask for the maximum probability (quantified over all schedulers) of reaching the goal within a varying number of steps. The first query does not restrict the terrain visited before the goal, but the second and third require to avoid G and S, respectively. The difference between the true values and the average model-checking results for L^*_{MDP} is lower than 0.01, but the results for IoAlergia differ greatly from the true values. One reason for this is that the IoAlergia models are much smaller than the true model. The average number of states in IoAlergia models is 19.9, whereas L^*_{MDP} always learned models with 35 states, which is the exact size of the true model.

IoAlergia is faster than L^*_{MDP}, since L^*_{MDP} applies time-consuming computations during equivalence queries. However, the runtime of learning-specific computations is often negligible in practical applications. This is due to the fact that the communication with the SUL during test-case execution usually dominates the overall runtime. We, for instance, experienced that when we learned models of Message Queuing Telemetry Transport (MQTT) brokers [TAB17].

Given the smaller bisimilarity distance and the lower difference to the true probabilities computed with Prism, we conclude that L^*_{MDP} learns more accurate models.

7.3. Experiments with second gridworld

Figure 4 shows the second gridworld used in our evaluation. As before, the robot starts in the initial location in the top left corner and can only observe the different terrains. The goal location is in the bottom right corner in this example. The true minimal MDP representing this gridworld has 72 states. We configured learning as for the first gridworld, but collect more samples per round by setting $n_{\text{retest}} = n_{\text{resample}} = 1000$. Table 4 shows the measurement results including the model-checking results and the bisimilarity distances.

L^*_{MDP} sampled on average 481203.5 traces with a combined number of outputs of 3013453.6. The combined length of all traces is only about 42% higher than in the previous example, although L^*_{MDP} sampled more than three times as many traces in a single round. This is the case, because learning generally stopped sooner. It stopped on average after 521.2 rounds. We used similar model-checking queries as in the previous example that ask for the maximum probability of reaching the goal location within varying number of steps. The third and fourth query additionally specify to avoid the terrains M and the S, respectively. We can again see that the L^*_{MDP} models have the same size as the minimal true model, while the IoAlergia models are much smaller. The bisimilarity distances between the true model and the L^*_{MDP} models are much smaller than for IoAlergia as well. Also as in the first gridworld experiment, the difference to the true model-checking results is smaller for L^*_{MDP}.

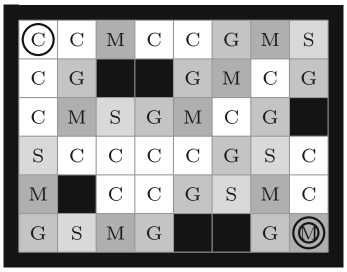

Fig. 4. The second evaluation gridworld

Table 4. Results for learning the second gridworld example

	true model	L^*_{MDP}	IoAlergia
# outputs	–	3012392.8 ± 696806.75	3013453.6 ± 697818.44
# traces	–	481203.5 ± 115931.30	376550.0 ± 87101.15
time [s]	–	176.52 ± 46.15	20.75 ± 11.81
# states	72	72.0 ± 0.0	33.1 ± 2.28
$\delta_{0.9}$	–	0.1472 ± 0.0221	0.5752 ± 0.0087
$\mathbb{P}_{\max}(F^{\leq 14}(\text{goal}))$	0.9348	0.9335 ± 0.0035	0.0229 ± 0.0040
$\mathbb{P}_{\max}(F^{\leq 12}(\text{goal}))$	0.6712	0.6708 ± 0.0062	0.0191 ± 0.0034
$\mathbb{P}_{\max}(\neg \text{ M } U^{\leq 18}(\text{goal}))$	0.9743	0.9736 ± 0.0023	0.0213 ± 0.0043
$\mathbb{P}_{\max}(\neg \text{ S } U^{\leq 20}(\text{goal}))$	0.1424	0.1541 ± 0.0010	0.0276 ± 0.0045

7.4. Shared coin consensus-protocol experiments

The following experiments are based on the shared coin consensus protocol by Aspnes and Herlihy [AH90]. We examined this protocol also in previous work [AT19b], for which we adapted a model distributed with the PRISM model checker [KNP11] that we now use again. The adaptations concern only minor changes such as adding action labels for inputs, but we also slightly changed the functionality by doing that. For the purpose of evaluating L^*_{MDP}, these changes are immaterial, though.

In our experiments, we consider the protocol configuration with the smallest state space of size 272 with two processes and the constant K set to 2. We set the learning parameters to $n_{\text{resample}} = n_{\text{retest}} = n_{\text{test}} = 50$, $p_{\text{stop}} = 0.25$ and $p_{\text{rand}} = 0.25$. We controlled stopping with $t_{\text{unamb}} = 0.99$, $r_{\text{min}} = 500$ and $r_{\text{max}} = 4000$. Finally, we set $p_l = 0.125$ for IoAlergia.

Table 5 shows the measurement results computed in the evaluation of learned models of the shared coin consensus protocol. Compared to the previous examples, we need a significantly lower sample size of 168503.9 traces containing 805132.5 outputs for convergence to be detected by our heuristic stopping criterion. This happens despite the fact that the models are much larger. A reason for this is that there is a relatively large number of outputs in this example, such that states are easier to distinguish from each other. The bisimilarity distance is in a similar range as before for L^*_{MDP}, which is again significantly smaller than IoAlergia's bisimilarity distance. The models learned by L^*_{MDP} are larger than those learned by IoAlergia, which is also in line with previous experiments. However, in this example the L^*_{MDP} models are smaller than the true model. This happens because many states are never reached during learning, as reaching them within a bounded number of steps has a very low probability; see, for instance, the fifth model-checking query. This query determines the maximum probability of finishing the protocol within less than 40 steps, but without consensus, as process p_1 chooses heads and process p_2 chooses tails. The computed probability of this event is very low.

The other model-checking queries consider the maximum probability of: (1) finishing the protocol without consensus, (2) finishing the protocol with consensus, (3) finishing the protocol without reaching an intermediate counter state of 5, (4) finishing the protocol without reaching an intermediate counter state of 4. The queries (5) to (8) bound the number of steps by less than 40, but consider the same properties as the queries (1) to (4), which are unbounded. Here, we see that the model-checking results computed with the IoAlergia model are more accurate in some cases, but L^*_{MDP} produces more accurate results overall.

Table 5. Results for learning the shared coin consensus protocol

	true model	L^*_{MDP}	IoAlergia
# outputs	–	805132.5 ± 381915.49	806229.9 ± 382185.64
# traces	–	168503.9 ± 94605.49	100642.5 ± 47739.95
time [s]	–	4156.62 ± 3586.10	1.54 ± 0.83
# states	272	164.1 ± 6.35	94.4 ± 4.86
$\delta_{0.9}$	–	0.1187 ± 0.0188	0.4276 ± 0.0565
$\mathbb{P}_{\max}(F(\text{finished} \wedge \text{p}_1\text{\_heads} \wedge \text{p}_2\text{\_tails}))$	0.1069	0.0 ± 0.0	0.0 ± 0.0
$\mathbb{P}_{\max}(F(\text{finished} \wedge \text{p}_1\text{\_tails} \wedge \text{p}_2\text{\_tails}))$	0.5556	0.6713 ± 0.0254	0.6372 ± 0.0303
$\mathbb{P}_{\max}(\text{counter} \neq 5\ U\ \text{finished})$	0.3333	0.3975 ± 0.0185	0.5332 ± 0.0084
$\mathbb{P}_{\max}(\text{counter} \neq 4\ U\ \text{finished})$	0.4286	0.5125 ± 0.0239	0.6516 ± 0.0156
$\mathbb{P}_{\max}(F^{<40}(\text{finished} \wedge \text{p}_1\text{\_heads} \wedge \text{p}_2\text{\_tails}))$	0.0017	0.0 ± 0.0	0.0 ± 0.0
$\mathbb{P}_{\max}(F^{<40}(\text{finished} \wedge \text{p}_1\text{\_tails} \wedge \text{p}_2\text{\_tails}))$	0.2668	0.3129 ± 0.0107	0.2607 ± 0.0150
$\mathbb{P}_{\max}(\text{counter} \neq 5\ U^{<40}\ \text{finished})$	0.2444	0.2983 ± 0.01501	0.4422 ± 0.0052
$\mathbb{P}_{\max}(\text{counter} \neq 4\ U^{<40}\ \text{finished})$	0.2634	0.3262 ± 0.0162	0.4937 ± 0.0088

We see an increase in runtime compared to the gridworld examples. Learning the consensus-protocol model took about 69 min on average, whereas learning the gridworld models took less than 3 min. This is caused by the larger state space, since the precomputation time for equivalence testing grows with the state space. Overall, we see a similar picture as before: L^*_{MDP} trades runtime for increased accuracy which we deem reasonable.

7.5. Slot-machine experiments

The final experiments in this section are based on a slot machine example that we also used in previous work [AT19b] and which we adapted from Mao et al. [MCJ+12, MCJ+16]. The modelled slot machine has three reels which are initially blank. There are three inputs, one for spinning each reel. Spinning causes the reels to show either bar or apple, where the probability of bar decreases with increasing number of rounds of spinning. After three rounds, a prize of 0, 2, or 10 is granted, depending on the final reel configuration. A fourth input leads with a probability of 0.5 to immediate stopping or grants two extra rounds. The minimal model of the slot machine has a state space of size 109.

We configured sampling for IoAlergia with $p_l = 0.125$ and we set the following parameters for L^*_{MDP}: $n_{\text{resample}} = n_{\text{retest}} = n_{\text{test}} = 300$, $p_{\text{stop}} = 0.25$, $p_{\text{rand}} = 0.25$, $r_{\min} = 500$ and $r_{\max} = 18000$. To demonstrate the influence of the parameter t_{unamb}, we performed experiments with $t_{\text{unamb}} = 0.9$ and with $t_{\text{unamb}} = 0.99$.

Table 6 and Table 7 show the results for $t_{\text{unamb}} = 0.9$ and $t_{\text{unamb}} = 0.99$, respectively. Configured with $t_{\text{unamb}} = 0.9$, L^*_{MDP} stopped on average after 3850.7 rounds and it stopped after 15882.9 rounds if configured with $t_{\text{unamb}} = 0.99$. However, using the latter configuration we did not detect convergence in four of ten cases, i.e., we stopped after $r_{\max} = 18000$. We see here that learning an accurate model of the slot machine requires a large amount of samples; in the case of $t_{\text{unamb}} = 0.99$, we sampled 9439328.1 traces containing 30522642.9 outputs. The amount of outputs is about 10 times as high as for the second gridworld example. However, we also see that sampling more traces clearly pays off. The L^*_{MDP} results shown in Table 7 are much better than those shown in Table 6. Notably the state space stayed almost the same. L^*_{MDP} configured with $t_{\text{unamb}} = 0.9$ learned models of approximately the correct size. In contrast, the IoAlergia models are smaller than the true model, similar to the gridworld examples.

Table 6. Results for learning the slot machine with $t_{\text{unamb}} = 0.9$

	true model	L^*_{MDP}	IoAlergia
# outputs	–	6302447.5 ± 1308561.26	6299688.8 ± 1305783.23
# traces	–	2129473.5 ± 436506.02	787805.8 ± 163570.07
time [s]	–	1883.96 ± 499.19	39.5373 ± 31.86
# states	109	107.7 ± 1.95	88.5 ± 1.27
$\delta_{0.9}$	–	0.1696 ± 0.0182	0.2849 ± 0.0050
$\mathbb{P}_{\max}(F(\text{Pr}10))$	0.3637	0.3878 ± 0.0235	0.3907 ± 0.0157
$\mathbb{P}_{\max}(F(\text{Pr}2))$	0.6442	0.6774 ± 0.0204	0.6863 ± 0.0100
$\mathbb{P}_{\max}(F(\text{Pr}0))$	1.0	1.0 ± 0.0	1.0 ± 0.0
$\mathbb{P}_{\max}(X(X(\text{bar} - \text{bar} - \text{blank})))$	0.1600	0.1606 ± 0.0009	0.1606 ± 0.0010
$\mathbb{P}_{\max}(X(X(X(\text{apple} - \text{bar} - \text{bar}))))$	0.2862	0.2870 ± 0.0014	0.2841 ± 0.0033
$\mathbb{P}_{\max}(\neg\,(F^{<10}(\text{end})))$	0.25	0.3158 ± 0.0219	0.4136 ± 0.0272
$\mathbb{P}_{\max}(X(X(X(\text{apple} - \text{apple} - \text{apple}))) \wedge (F(\text{Pr}0)))$	0.0256	0.0287 ± 0.0038	0.0127 ± 0.0018

Table 7. Results for learning the slot machine with $t_{\text{unamb}} = 0.99$

	true model	L^*_{MDP}	IoAlergia
# outputs	–	$30522642.9 \pm 4897262.21$	$30517331.2 \pm 4898907.76$
# traces	–	9439328.1 ± 1657544.92	3815330.7 ± 612157.98
time [s]	–	9038.91 ± 1772.11	1072.53 ± 1176.20
# states	109	109.6 ± 0.97	97.8 ± 0.79
$\delta_{0.9}$	–	0.0624 ± 0.0419	0.2597 ± 0.0030
$\mathbb{P}_{\max}(F(\text{Pr}10))$	0.3637	0.3700 ± 0.0030	0.4101 ± 0.0161
$\mathbb{P}_{\max}(F(\text{Pr}2))$	0.6442	0.6533 ± 0.0045	0.6987 ± 0.0057
$\mathbb{P}_{\max}(F(\text{Pr}0))$	1.0	1.0 ± 0.0	1.0 ± 0.0
$\mathbb{P}_{\max}(X(X(\text{bar} - \text{bar} - \text{blank})))$	0.1600	0.1602 ± 0.0004	0.1604 ± 0.0005
$\mathbb{P}_{\max}(X(X(X(\text{apple} - \text{bar} - \text{bar}))))$	0.2862	0.2864 ± 0.0006	0.2860 ± 0.0008
$\mathbb{P}_{\max}(\neg\,(F^{<10}(\text{end})))$	0.25	0.2749 ± 0.0245	0.4184 ± 0.0150
$\mathbb{P}_{\max}(X(X(X(\text{apple} - \text{apple} - \text{apple}))) \wedge (F(\text{Pr}0)))$	0.0256	0.0265 ± 0.0005	0.0255 ± 0.0011

We also see in both settings of t_{unamb} that L^*_{MDP} models are more accurate than IoAlergia models, with respect to bisimilarity distance and with respect to model-checking results. While the experiments with $t_{\text{unamb}} = 0.99$ required the most samples among all experiments, they also led to the lowest average bisimilarity distance.

The first three model-checking queries determine the maximum probability of reaching one of the three prizes issued by the slot machine. The next two queries consider certain reel configurations reached after exactly two and three steps, respectively. The sixth query checks the maximum probability that ending the game can be avoided for 9 steps. The final query computes the maximum probability of reaching the prize Pr0 and observing the reel configuration apple – apple – apple after exactly three steps. It is noteworthy that the average model-checking results for L^*_{MDP} configured with $t_{\text{unamb}} = 0.99$ are within a low range of approximately 0.025 of the true results.

A drawback of L^*_{MDP} compared to IoAlergia is again the learning runtime, as L^*_{MDP} with $t_{\text{unamb}} = 0.99$ required about 2.5 h while learning with IoAlergia took only about 17.9 min. However, these numbers should be put into context. In a non-simulated environment, the sampling time would be much larger than 2.5 h, such that the learning runtime becomes negligible. Consider, for instance, a scenario where the sampling of a single trace takes 20 milliseconds. The sampling time of L^*_{MDP} is about 52.4 h in that scenario which is about 21 times as long as the learning runtime.

7.6. Discussion and threats to validity

Our case studies demonstrated that L^*_{MDP} is able to achieve better accuracy than IoAlergia. The bisimilarity distances of L^*_{MDP} models to the true models were generally lower and the model-checking results were more accurate. These observations should be investigated in further case studies. It should be noted, though, that the considered systems have different characteristics, covering various model properties. The gridworld has a small state space, but it is strongly connected and the terrains lead to different probabilistic decisions. For instance, if we try to enter M there is a probability of 0.4 of entering one of the neighbouring tiles, whereas entering C is generally successful (the probability of entering other tiles instead is 0). The consensus protocol has a large state space with many different outputs and finishing the protocol takes at least 14 steps. The slot machine requires states to be distinguished based on subtle differences in probabilities, because the probability of seeing bar decreases in each round.

L^*_{MDP} has several parameters that affect performance and accuracy. We plan to investigate the influence of parameters in further experiments. For the presented experiments, we fixed most of the parameters except for n_{retest}, n_{test} and n_{resample}. However, we observed that results are robust with respect to these parameters. For instance, we increased n_{resample} from 300 for the first gridworld to 1000 for the second gridworld. Both settings led to approximately the same results, as learning simply performed fewer rounds with $n_{\text{resample}} = 1000$. Nevertheless, we will examine in further experiments if the fixed parameters are indeed appropriately chosen and if guidelines for choosing other parameters can be provided.

L^*_{MDP} and IoAlergia learn from different traces, thus the trace selection may actually be a major reason for the better accuracy of L^*_{MDP}. We examined if this is the case by learning IoAlergia models from two types of given traces: traces with uniform input selection and traces sampled during learning with L^*_{MDP}. We noticed that models learned from L^*_{MDP} traces led to less accurate results in many cases, especially in terms of bisimilarity distance. For this reason, we reported only results for models learned with IoAlergia from traces with uniformly distributed inputs.

8. Related work

In this section, we review related work in the area of automata learning with a focus on learning stochastic and non-deterministic automata.

Most sampling-based learning algorithms for stochastic systems are passive, hence they assume preexisting samples of system traces. Their roots can be found in grammar inference techniques like Alergia [CO94] and rlips [CO99], which identify stochastic regular languages. Similarly to these techniques, we also apply Hoeffding bounds [Hoe63] in L^*_{MDP}, to test for differences between samples of probability distributions.

Mao et al. [MCJ+11, MCJ+12, MCJ+16] learned stochastic automata models with the purpose of verification. More concretely, they learned models for model checking. Notably, they developed IoAlergia as an extension of Alergia to learn MDPs [MCJ+12, MCJ+16]. This learning technique basically creates a tree-shaped representation of the sampled system traces and repeatedly merges compatible nodes to create an automaton. Finally, transition probabilities are estimated from observed output frequencies. Like L^*_{MDP}, IoAlergia converges in the limit, but showed worse accuracy in our evaluation experiments in Sect. 7. Chen and Nielsen [CN12] also described active learning of MDPs, but used IoAlergia as a basis. They proposed to generate new samples by directing sampling towards uncertainties in the data as a way to reduce the number of traces required for learning. In L^*_{MDP}, we base the sampling strategy not only on the data collected so far (refine queries), but also on the current observation table and the derived hypothesis MDPs (refine and equivalence queries). With that, we do not only target uncertainties in the collected data, but we also take information about the SUL's structure into account.

Wang et al. [WSQ16] learn models using a variant of Alergia while taking properties into account with the goal of probabilistic model-checking. Nouri et al. [NRB+14] combine stochastic learning and abstraction with respect to some property, to improve the runtime of statistical model-checking (SMC). Ghezzi et al. [GPST14] presented the BEAR approach that combines inference of probabilistic models and probabilistic model-checking to analyse user behaviour in web applications. More concretely, the authors infer labelled discrete-time Markov chains extended with rewards which they analyse using Prism [KNP11].

L^*_{MDP} builds upon Angluin's L* [Ang87], therefore it shares similarities with other L*-based work such as active learning of Mealy machines [SG09, MNRS04]. Interpreting MDPs as functions from test sequences to output distributions is similar to the interpretation of Mealy machines as functions from input sequences to outputs [SHM11].

Volpato and Tretmans presented an L^*-based technique for non-deterministic input-output transition systems [VT15]. They simultaneously learn an over- and an under-approximation of the SUL with respect to the input-output conformance (**ioco**) relation [Tre96], a popular conformance relation in conformance testing. Inspired by that, L^*_{MDP} uses completeness queries and adds transitions to a chaos state in case of low/incomplete information.

Beyond that, we consider systems to behave stochastically rather than non-deterministically. While Volpato and Tretmans [VT15] leave the concrete implementation of queries unspecified, L^*_{MDP}'s implementation closely follows Sect. 5. Early work on **ioco**-based learning for non-deterministic systems has been presented by Willemse [Wil06]. Khalili and Tacchella [KT14] addressed non-determinism by presenting an L^*-based algorithm for non-deterministic Mealy machines. As Volpato and Tretmans [VT15], they assume to be able to observe all possible outputs produced in response to input sequences through the repeated application of these sequences. Both learning approaches for non-deterministic models assume a testing context, as we do. More recently, Pferscher and Aichernig proposed an L^*-based learning approach for observable non-deterministic finite state-machines (ONFSMs) [PA20], which are similar to Khalili and Tacchella's non-deterministic Mealy machines. ONFSMs are well-suited to capture non-deterministic behaviour resulting from abstraction, thus the approach enables efficient learning.

Feng et al. [FHKP11] also presented L^*-based learning for probabilistic systems. They learn assumptions in the form of probabilistic finite automata for compositional verification of probabilistic systems in the assume-guarantee framework. Their learning algorithm requires queries returning exact probabilities, hence it is not directly applicable in a sampling-based setting. The learning algorithm shares similarities with an L^*-based algorithm for learning multiplicity automata [BV96], a generalisation of deterministic automata. In order to extend the applicability of learning-based assume-guarantee reasoning for probabilistic systems, Komuravelli et al. [KPC12] presented techniques for learning labelled probabilistic transition systems from stochastic tree samples, where they rely solely on equivalence queries for active learning. Moreover, they described how their algorithms can be applied to generate assumptions for assume-guarantee reasoning.

Further query-based learning in a probabilistic setting has been presented by Tzeng [Tze92]. The author described a query-based algorithm for learning probabilistic automata and an adaptation of Angluin's L^* for learning Markov chains. In contrast to our exact learning algorithm $L^*_{\mathrm{MDP}^e}$, which relies on output distribution queries, Tzeng's algorithm for Markov chains queries the generating probabilities of strings. Castro and Gavaldà review passive learning techniques for probabilistic automata with a focus on convergence guarantees and present them in a query framework [CG16]. In contrast to MDPs, which are reactive, the learned automata are generative probabilistic automata [SdV04]. Hence, the assumed mode of interaction with the environment is different. While MDPs receive inputs (actions) and react probabilistically, generative probabilistic automata produce actions probabilistically.

9. Summary

In this article, we presented two L^*-based algorithms for learning MDPs. For our exact learning algorithm $L^*_{\mathrm{MDP}^e}$, we assume an ideal setting that allows to query information about the SUL with exact precision. Subsequently, we relaxed our assumptions by approximating exact queries through sampling SUL traces via directed testing. These traces serve to infer the structure of hypothesis MDPs, to estimate transition probabilities and to check for equivalence between the SUL and learned hypotheses. The resulting sampling-based L^*_{MDP} iteratively learns approximate MDPs which converge to the correct MDP in the large sample limit. We implemented L^*_{MDP} and evaluated it by comparing it to IoAlergia [MCJ$^+$16], a state-of-the-art passive learning algorithm for MDPs. The evaluation showed that L^*_{MDP} is able to produce more accurate models. To the best of our knowledge, L^*_{MDP} is the first L^*-based algorithm for MDPs that can be implemented via testing. Experimental results and the implementation can be found in the evaluation material for L^*_{MDP} [Tap20].

10. Conclusion

The evaluation of L^*_{MDP} showed promising results. We applied it to successfully learn models of black-box systems, such as a model of the shared coin consensus protocol. Our evaluation demonstrated that the learned models accurately capture the true system behaviour in many cases. The results of model checking the learned gridworld models, for instance, are very close to the true results.

In additon to providing a heuristic stopping criterion, we show that the models learned by L^*_{MDP} converge to the true model in the limit. Although difficult in a verification context [MCJ+16], it would be worthwhile to analyse L^*_{MDP} with respect to probably approximately correct (PAC) learnability [Val84] to provide stronger convergence guarantees. L^*_{MDP} also provides room for experimentation. Different testing techniques, such as probabilistic black-box reachability checking [AT19b], could be applied in equivalence queries.

Funding Open access funding provided by Graz University of Technology.

Acknowledgements

The work of B. Aichernig, M. Eichlseder and M. Tappler has been carried out as part of the TU Graz LEAD project "Dependable Internet of Things in Adverse Environments". The work of K. Larsen and G. Bacci has been supported by the Advanced ERC Grant no. 867096 (LASSO).

References

[AH90] Aspnes J, Herlihy M (1990) Fast randomized consensus using shared memory. J Algorithms 11(3):441–461

[AMM+18] Aichernig BK, Mostowski W, Mousavi MR, Tappler M, Taromirad M (2018) Model learning and model-based testing. In Bennaceur A, Hähnle R, Meinke K (eds) Machine learning for dynamic software analysis: potentials and limits—international Dagstuhl seminar 16172, Dagstuhl Castle, Germany, April 24–27, 2016, revised papers, volume 11026 of lecture notes in computer science. Springer, pp 74–100

[Ang87] Angluin D (1987) Learning regular sets from queries and counterexamples. Inf Comput 75(2):87–106

[AT17a] Aichernig BK, Tappler M (2017) Learning from faults: mutation testing in active automata learning. In: Barrett C, Davies M, Kahsai T (eds) NASA formal methods—9th international symposium, NFM 2017, Moffett Field, CA, USA, May 16–18, 2017, proceedings, volume 10227 of lecture notes in computer science, pp 19–34

[AT17b] Aichernig BK, Tappler M (2017) Probabilistic black-box reachability checking. In: Lahiri SK, Reger G (eds) Runtime verification—17th international conference, RV 2017, Seattle, WA, USA, September 13–16, 2017, proceedings, volume 10548 of lecture notes in computer science, pp 50–67. Springer

[AT19a] Aichernig Bernhard K, Martin T (2019) Efficient active automata learning via mutation testing. J Autom Reason 63(4):1103–1134

[AT19b] Aichernig BK, Tappler M (2019) Probabilistic black-box reachability checking (extended version). Form Methods Syst Des

[BBLM] Bacci G, Bacci G, Guldstrand LK, Mardare R. MDPDist library. http://people.cs.aau.dk/~giovbacci/tools/bisimdist.zip. Accessed on 04 Nov 2019

[BBLM13a] Bacci G, Bacci G, Larsen KG, Mardare R (2013) The BisimDist library: efficient computation of bisimilarity distances for Markovian models. In: Joshi KR, Siegle M, Stoelinga M, D'Argenio PR (eds) Quantitative evaluation of systems—10th international conference, QEST 2013, Buenos Aires, Argentina, August 27–30, 2013. Proceedings, volume 8054 of lecture notes in computer science. Springer, pp 278–281

[BBLM13b] Bacci G, Bacci G, Larsen KG, Mardare R (2013) Computing behavioral distances, compositionally. In: Chatterjee K, Sgall J (eds) Mathematical foundations of computer science 2013—38th international symposium, MFCS 2013, Klosterneuburg, Austria, August 26–30, 2013. Proceedings, volume 8087 of lecture notes in computer science. Springer, pp 74–85

[BK08] Baier C, Katoen JP (2008) Principles of model checking. MIT Press, Cambridge

[BV96] Bergadano F, Varricchio S (1996) Learning behaviors of automata from multiplicity and equivalence queries. SIAM J Comput 25(6):1268–1280

[CG16] Castro J, Gavalda R (2016) Learning probability distributions generated by finite-state machines. In: Heinz J, Sempere JM (eds) Topics in grammatical inference. Springer, Berlin, pp 113–142

[CHJS16] Cassel S, Howar F, Jonsson B, Steffen B (2016) Active learning for extended finite state machines. Formal Aspects Comput 28(2):233–263

[Cho78] Chow TS (1978) Testing software design modeled by finite-state machines. IEEE Trans Software Eng 4(3):178–187

[CN12] Chen Y, Nielsen TD (2012) Active learning of Markov decision processes for system verification. In: 11th international conference on machine learning and applications, ICMLA, Boca Raton, FL, USA, December 12–15, 2012. Volume 2. IEEE, pp 289–294

[CO94] Carrasco RC, Oncina J (1994) Learning stochastic regular grammars by means of a state merging method. In: Carrasco RC, Oncina J (eds) Grammatical inference and applications, second international colloquium, ICGI-94, Alicante, Spain, September 21–23, 1994. Proceedings, volume 862 of lecture notes in computer science. Springer, pp 139–152

[CO99] Carrasco RC, Oncina J (1999) Learning deterministic regular grammars from stochastic samples in polynomial time. Theor Inf Appl 33(1):1–20

[dlH10] de la Higuera C (2010) Grammatical inference: learning automata and grammars. Cambridge University Press, New York

[FHKP11] Feng L, Han T, Kwiatkowska MZ, Parker D (2011) Learning-based compositional verification for synchronous probabilistic systems. In: Bultan T, Hsiung P-A (eds) Automated technology for verification and analysis, 9th international symposium, ATVA 2011, Taipei, Taiwan, October 11–14, 2011. Proceedings, volume 6996 of lecture notes in computer science. Springer, pp 511–521

[FKNP11] Forejt V, Kwiatkowska MZ, Norman G, Parker D (2011) Automated verification techniques for probabilistic systems. In: Bernardo M, Issarny V (eds) Formal methods for eternal networked software systems—11th international school on formal methods for the design of computer, communication and software systems, SFM 2011, Bertinoro, Italy, June 13–18, 2011. Advanced lectures, volume 6659 of lecture notes in computer science. Springer, pp 53–113

[GPST14] C Ghezzi et al (2014) Mining behavior models from user-intensive web applications. In: Jalote P, Briand LC, van der Hoek A (eds) 36th international conference on software engineering, ICSE'14, Hyderabad, India—May 31–June 07, 2014. ACM, pp 277–287

[HNS03] Hungar H, Niese O, Steffen B (2003) Domain-specific optimization in automata learning. In: Hunt Jr. WA, Somenzi F (eds) Computer aided verification, 15th international conference, CAV 2003, Boulder, CO, USA, July 8–12, 2003. Proceedings, volume 2725 of lecture notes in computer science. Springer, pp 315–327

[Hoe63] Hoeffding W (1963) Probability inequalities for sums of bounded random variables. J Am Stat Assoc 58(301):13–30

[HS18] Howar F, Steffen B (2018) Active automata learning in practice—an annotated bibliography of the years 2011 to 2016. In: Bennaceur A, Hähnle R, Meinke K (eds) Machine learning for dynamic software analysis: potentials and limits—international Dagstuhl seminar 16172, Dagstuhl Castle, Germany, April 24–27, 2016, revised papers, volume 11026 of lecture notes in computer science. Springer, pp 123–148

[IHS14] Isberner M, Howar F, Steffen B (2014) The TTT algorithm: a redundancy-free approach to active automata learning. In: Bonakdarpour B, Smolka SA (eds) Runtime verification—5th international conference, RV 2014, Toronto, ON, Canada, September 22–25, 2014. Proceedings, volume 8734 of lecture notes in computer science. Springer, pp 307–322

[Kea98] Kearns MJ (1998) Efficient noise-tolerant learning from statistical queries. J ACM 45(6):983–1006

[KNP08] Kwiatkowska Marta Z, Gethin N, David P (2008) Analysis of a gossip protocol in PRISM. SIGMETRICS Perform Eval Rev 36(3):17–22

[KNP11] Kwiatkowska MZ, Norman G, Parker D (2011) PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan G, Qadeer S (eds) Computer aided verification—23rd international conference, CAV 2011, Snowbird, UT, USA, July 14–20, 2011. Proceedings, volume 6806 of lecture notes in computer science. Springer, pp. 585–591

[KP13] Kwiatkowska MZ, Parker D (2013) Automated verification and strategy synthesis for probabilistic systems. In: Van Hung D, Ogawa M (eds) Automated technology for verification and analysis—11th international symposium, ATVA 2013, Hanoi, Vietnam, October 15–18, 2013. Proceedings, volume 8172 of lecture notes in computer science. Springer, pp 5–22

[KPC12] Komuravelli A, Pasareanu CS, Clarke EM (2012) Learning probabilistic systems from tree samples. In: Proceedings of the 27th annual IEEE symposium on logic in computer science, LICS 2012, Dubrovnik, Croatia, June 25–28, 2012. IEEE Computer Society, pp 441–450

[KT14] Khalili A, Tacchella A (2014) Learning nondeterministic Mealy machines. In: Clark A, Kanazawa M, Yoshinaka R (eds) Proceedings of the 12th international conference on grammatical inference, ICGI 2014, Kyoto, Japan, September 17–19, 2014, volume 34 of JMLR workshop and conference proceedings. JMLR.org, pp 109–123

[MCJ$^+$11] Mao H, Chen Y, Jaeger M, Nielsen TD, Larsen KG, Nielsen B (2011) Learning probabilistic automata for model checking. In: Eighth international conference on quantitative evaluation of systems, QEST 2011, Aachen, Germany, 5–8 September, 2011. IEEE Computer Society, pp 111–120

[MCJ$^+$12] Mao H, Chen Y, Jaeger M, Nielsen TD, Larsen KG, Nielsen B (2012) Learning Markov decision processes for model checking. In: Fahrenberg U, Legay A, Thrane CR (eds) Proceedings quantities in formal methods, QFM 2012, Paris, France, 28 August 2012., volume 103 of EPTCS, pp 49–63

[MCJ$^+$16] Mao H, Chen Y, Jaeger M, Nielsen TD, Larsen KG (2016) Learning deterministic probabilistic automata from a model checking perspective. Mach Learn 105(2):255–299

[MNRS04] Margaria T, Niese O, Raffelt H, Steffen B (2004) Efficient test-based model generation for legacy reactive systems. In: Ninth IEEE international high-level design validation and test workshop 2004, Sonoma Valley, CA, USA, November 10–12, 2004. IEEE Computer Society, pp 95–100

[Ner58] Nerode A (1958) Linear automaton transformations. Proc Am Math Soc 9(4):541–544

[NRB$^+$14] Nouri A, Raman B, Bozga M, Legay A, Bensalem S (2014) Faster statistical model checking by means of abstraction and learning. In: Bonakdarpour B, Smolka SA (eds) runtime verification—5th international conference, RV 2014, Toronto, ON, Canada, September 22–25, 2014. Proceedings, volume 8734 of lecture notes in computer science. Springer, pp 340–355

[NS06] Norman G, Shmatikov V (2006) Analysis of probabilistic contract signing. J Comput Secur 14(6):561–589

[PA20] Pferscher A, Aichernig BK (2020) Learning abstracted non-deterministic finite state machines. In: Casola V, De Benedictis A, Rak M (eds) Testing Software and Systems—32nd IFIP WG 6.1 international conference, ICTSS 2020, Naples, Italy, December 9–11, 2020. Proceedings, volume 12543 of lecture notes in computer science. Springer, pp 52–69

[Put94] Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley series in probability and statistics. Wiley

[RS93] Rivest RL, Schapire RE (1993) Inference of finite automata using homing sequences. Inf Comput 103(2):299–347

[SdV04] Sokolova A, de Vink EP (2004) Probabilistic automata: system types, parallel composition and comparison. In: Baier C, Haverkort BR, Hermanns H, Katoen J-P, Siegle M (eds) Validation of stochastic systems—a guide to current research, volume 2925 of lecture notes in computer science. Springer, pp 1–43

[SG09] Shahbaz M, Groz R (2009) Inferring mealy machines. In: Cavalcanti A, Dams D (eds) FM 2009: formal methods, second world congress, Eindhoven, The Netherlands, November 2–6, 2009. Proceedings, volume 5850 of lecture notes in computer science. Springer, pp 207–222

[SHM11] Steffen B, Howar F, Merten M (2011) Introduction to active automata learning from a practical perspective. In: Bernardo M, Issarny V (eds) Formal methods for eternal networked software systems—11th international school on formal methods for the design of computer, communication and software systems, SFM 2011, Bertinoro, Italy, June 13–18, 2011. Advanced lectures, volume 6659 of lecture notes in computer science. Springer, pp 256–296

[SL95] Segala R, Lynch N (1995) Probabilistic simulations for probabilistic processes. Nord J Comput 2(2):250–273

[Sto02] Stoelinga M (2002) An introduction to probabilistic automata. Bull. EATCS 78:176–198

[TAB17] Tappler M, Aichernig BK, Bloem R (2017) Model-based testing IoT communication via active automata learning. In: 2017 IEEE international conference on software testing, verification and validation, ICST 2017, Tokyo, Japan, March 13–17, 2017. IEEE Computer Society, pp 276–287

[TAB$^+$19] Tappler M, Aichernig BK, Bacci G, Eichlseder M, Larsen KG (2019) L^*-based learning of Markov decision processes. In: ter Beek MH, McIver A, Oliveira JN (eds) Formal methods—the next 30 years—third world congress, FM 2019, Porto, Portugal, October 7–11, 2019. Proceedings, volume 11800 of lecture notes in computer science. Springer, pp 651–669

[Tap19] Tappler M (2019) Learning-based testing in networked environments in the presence of timed and stochastic behaviour. PhD thesis, Graz University of Technology

[Tap20] Tappler M (2020) Evaluation material for L^*-based learning of Markov decision processes. https://doi.org/10.6084/m9.figshare.7960928.v2. Accessed on 06 Mar 2020, updated for extended version

[Tre96] Tretmans J (1996) Test generation with inputs, outputs and repetitive quiescence. Softw Concepts Tools 17(3):103–120

[Tre08] Tretmans J (2008) Model based testing with labelled transition systems. In: Hierons RM, Bowen JP, Harman M (eds) Formal methods and testing, an outcome of the FORTEST network, revised selected papers, volume 4949 of lecture notes in computer science. Springer, pp 1–38

[Tze92] Wen-Guey T (1992) Learning probabilistic automata and Markov chains via queries. Mach Learn 8:151–166

[Vaa17] Vaandrager Frits W (2017) Model learning. Commun ACM 60(2):86–95

[Val84] Valiant LG (1984) A theory of the learnable. Commun ACM 27(11):1134–1142

[VT15] Volpato M, Tretmans J (2015) Approximate active learning of nondeterministic input output transition systems. ECEASST, 72

[Wil06] Willemse TAC (2006) Heuristics for ioco-based test-based modelling. In: Brim L, Haverkort BR, Leucker M, van de Pol J (eds) Formal methods: applications and technology, 11th international workshop, FMICS 2006 and 5th international workshop pdmc 2006, Bonn, Germany, August 26–27, and August 31, 2006, revised selected papers, volume 4346 of lecture notes in computer science. Springer, pp 132–147

[WSQ16] Wang J, Sun J, Qin S (2016) Verifying complex systems probabilistically through learning, abstraction and refinement. CoRR, abs/1610.06371

Received 17 March 2020

Accepted in revised form 30 January 2021 by Annabelle McIver, Maurice ter Beek and Cliff Jones

Published online 31 March 2021