Software and Systems Modeling
https://doi.org/10.1007/s10270-024-01158-0

SPECIAL SECTION PAPER

Active model learning of stochastic reactive systems (extended version)

Edi Muškardin[1,2] · Martin Tappler[1,2] · Bernhard K. Aichernig[2] · Ingo Pill[1]

Received: 2 July 2022 / Revised: 23 January 2024 / Accepted: 8 February 2024

Abstract
Black-box systems are inherently hard to verify. Many verification techniques, like model checking, require formal models as a basis. However, such models often do not exist, or they might be outdated. Active automata learning helps to address this issue by offering to automatically infer formal models from system interactions. Hence, automata learning has been receiving much attention in the verification community in recent years. This led to various efficiency improvements, paving the way toward industrial applications. Most research, however, has been focusing on deterministic systems. In this article, we present an approach to efficiently learn models of stochastic reactive systems. Our approach adapts L^*-based learning for Markov decision processes, which we improve and extend to stochastic Mealy machines. When compared with previous work, our evaluation demonstrates that the proposed optimizations and adaptations to stochastic Mealy machines can reduce learning costs by an order of magnitude while improving the accuracy of learned models.

Keywords Active automata learning · Model mining · Probabilistic verification · Stochastic mealy machines · Markov decision processes

1 Introduction

Formal techniques are an attractive and important asset when it comes to assessing functional properties of a system or one of its components—prominent examples being tools like model checkers or approaches like model-based testing and diagnosis. They require us to provide a detailed system model though, which we often lack in practice. This might stem from using third-party components and the arising IP issues, or due to a lack of the resources needed to derive and maintain such a model and where we potentially suffer also from incomplete documentation. So, while we most certainly have a specification of a system's I/O interface at our discretion, we lack in many cases a detailed model which is a prerequisite for a variety of formal and model-based verification and debugging techniques.

Active automata learning allows us to learn such a detailed model in the form of an automaton from a black-box reactive system and enables us to exploit formal as well as model-based techniques and tools where this would be impossible otherwise. Consequently, this has been a very active research area since the inception of the field with Angluin's L^* algorithm [1] for learning deterministic finite automata (DFAs). Corresponding extensions that have paved the way to applying the concept in an industrial context range from general algorithmic improvements [2, 3] over domain-specific optimizations [4] to learning further automata variants like Mealy machines [5, 6] or timed automata [7, 8]. While in the deterministic setting executing an input from a state always produces the same output, in non-deterministic and stochastic learning a single input might produce multiple outputs. Stochastic systems produce outputs that are distributed probabilistically. Stochastic learning algorithms accordingly learn models where output distributions are

Communicated by Antonio Cerone and Frank de Boer.

✉ Martin Tappler
martin.tappler@ist.tugraz.at

Edi Muškardin
edi.muskardin@silicon-austria.com

Bernhard K. Aichernig
aichernig@ist.tugraz.at

Ingo Pill
ingo.pill@silicon-austria.com

1 TU Graz - SAL DES Lab, Silicon Austria Labs, Graz, Austria

2 Institute of Software Technology, Graz University of Technology, Graz, Austria

approximated based on the data observed during learning. In the non-deterministic setting, outputs are not sampled from probability distributions, but arise due to the inherent non-deterministic behavior of the system. In practice, non-determinism often arises from abstraction [9]. More concretely, by abstraction we consider the partitioning of concrete inputs or outputs into equivalence classes, and the non-deterministic behavior often arises due to imprecisions introduced by such abstraction. Non-deterministic behavior can also indicate the presence of unintended behavior/bugs in deterministic systems, as observed in [10].

While uncertainty is an issue we face in practice quite often, learning automata in settings with uncertainty has been receiving less attention so far. In previous work [11, 12] we presented the first L^*-based learning algorithm that allows one to actively learn Markov decision processes (MDPs) of stochastic systems. In the conference version of this manuscript, we improved upon our previous work via the following contributions: We (1) presented L^*_{SMM}, an L^*-based approach for learning stochastic Mealy machines (SMMs) along with algorithmic improvements that also apply to learning MDPs and (2) reported on a thorough experimental evaluation with implementations in our open-source library AALpy [13].

Extension of the original paper. In this manuscript, we make the following contributions in comparison to the original L^*_{SMM} paper [14]: we (a) extend all sections of the original paper, making the paper more self-contained, and (b) provide a more detailed description of the L^*_{SMM} algorithm. Additionally, (c) we extend the algorithm with an additional statistical compatibility check, which may be more suitable when learning stochastic system models under certain conditions. We (d) revise and discuss counterexample processing strategies. These algorithmic improvements demonstrate the extensibility and adaptability of L^*_{SMM} to new environments, potentially taking domain knowledge into account. For example, when we know that the system does not produce very rare outputs, we may choose to apply the newly introduced statistical compatibility check. Finally, (e) the present manuscript contains experiments with an additional benchmark based on a Bluetooth low-energy model, and (f) it provides a more detailed evaluation based on randomly generated stochastic automata and statistical model checking. Thus, we provide a more detailed evaluation, reinforcing our findings from the original paper regarding the sampling efficiency and accuracy of learned models.

As we will show, learning SMMs optimizes the learning process (i.e., the required number of interactions with a (SUL)) in comparison to existing work [12, 15]. An automatic translation from SMMs to MDPs enables the use of probabilistic model checkers like Prism [16] or Storm [17]. Thus, L^*_{SMM} can be used for learning-based verification of black-box systems.

Structure. In Sect. 2, we recapitulate preliminaries like L^*-based automata learning. We present our approach to active learning of SMMs in detail in Sect. 3, followed by a corresponding evaluation and comparison to learning MDPs in Sect. 4. After a discussion of related work in Sect. 5, we will conclude with a summary of our findings and an outlook on future work in Sect. 6.

2 Preliminaries

In this section, we will introduce our notation as well as background knowledge.

2.1 Foundations

A discrete probability distribution μ over a countable set X is a function $\mu : X \rightarrow [0, 1]$ such that $\sum_{x \in X} \mu(x) = 1$, where we refer with $Dist(X)$ to the set of all probability distributions over X. To ease probability estimations from sampled data, we focus on rational probabilities only, i.e., $\mu(x)$ takes only rational values. Furthermore, we support partial functions μ by assuming $\mu(x) = 0$ if μ is undefined for some x. The set $supp(\mu) = \{x \in X \mid \mu(x) > 0\}$ is referred to as the support of μ. We use $A(e) \in \mathbb{N}_0$ to denote the multiplicity of e in some multiset A.

For a finite set X, let $u, v, w \in X^*$ be finite sequences over X: $u = v \cdot w$ is the concatenation of v and w, where v is a prefix of u, and w is a suffix. The length of u is denoted $|u|$, ϵ denotes the empty sequence, and we lift $x \in X$ to be a sequence of length one. For $A, B \subseteq X^*$, $A \cdot B$ is the set of all concatenations of sequence pairs in A and B. $A \subseteq X^*$ is prefix/suffix-closed if A contains all prefixes/suffixes of all $s \in A$.

We consider reactive systems M that produce exactly one output $o \in O$ at a time in response to an input $i \in I$ for in- and output alphabets I and O. Thus, an interaction with M, coined a *trace*, is a finite sequence $t \in (I \cdot O)^*$, and a single interaction step consists of an input–output pair in $I \cdot O$. For convenience, we adopt the notions of prefix, suffix, and length for traces, referring to input–output pairs as atomic elements. For example, the trace set $T = \{\epsilon, a \cdot b, a \cdot b \cdot c \cdot d\}$ for $a, c \in I$ and $b, d \in O$ is prefix-closed, as T contains every input–output pair prefix of $a \cdot b \cdot c \cdot d$. The trace length $|t|$ gives the number of input–output pairs, so that, for example, $|a \cdot b \cdot c \cdot d| = 2$.

In addition to traces $t \in \mathcal{TR} = (I \cdot O)^*$, we consider *test sequences* from $\mathcal{TS} = (I \cdot O)^* \cdot I$, which test a system's response to an input i following a trace t. We also consider continuation sequences $\mathcal{CS} = (I \cdot O)^* \cdot I$, which extend traces to test sequences $\mathcal{TR} \cdot \mathcal{CS} = \mathcal{TS}$. Analogously to traces, we extend the notions of prefixes and suffixes by considering input–output pairs and trailing/leading inputs in these types of sequences. For example, $C = \{i_2 \cdot o_2 \cdot i_1, i_1\} \subseteq \mathcal{CS}$ is suffix-closed.

2.2 Stochastic system models

We learn SMMs and transform them into labeled MDPs. This has the advantage that learning SMMs is more efficient as we will show in our experiments, while the transformation enables model-based analyses with probabilistic model checkers [16, 17] that use MDPs as input format. Analogously to Mealy and Moore machines, SMMs and MDPs differ in the way outputs are produced. An SMM produces an output considering its current state and an input, whereas the output of an MDP depends only on the current state. Like in a non-stochastic setting, SMMs are potentially smaller as illustrated in Fig. 1 for a simple example.

On the top of Fig. 1, we show an MDP modeling a faulty coffee machine, and a corresponding SMM on the bottom (see Definitions 2.1, 2.2). Both models are observationally equivalent, except for the initial output produced by the MDP. For both we have inputs and probabilities as edge labels, s.t. the probabilities follow after a colon. In the MDP, the outputs are defined by the labels on the states, whereas for the SMM they are part of the edge label—separated from the input via a slash. The described coffee machine works such that it sounds a *beep* with probability one upon receiving a *coin*. When a *button* is pressed, a *coffee* is issued with a probability of 0.9, or we have *init* with the complementary probability of 0.1.

Definition 2.1 *(Markov Decision Processes)* A labeled MDP is a tuple $\mathcal{M} = \langle Q, I, O, q_0, \Delta, L\rangle$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, I and O are finite sets of input and output symbols, $\Delta : Q \times I \to Dist(Q)$ is the probabilistic transition function, and $L : Q \to O$ is the labeling function.

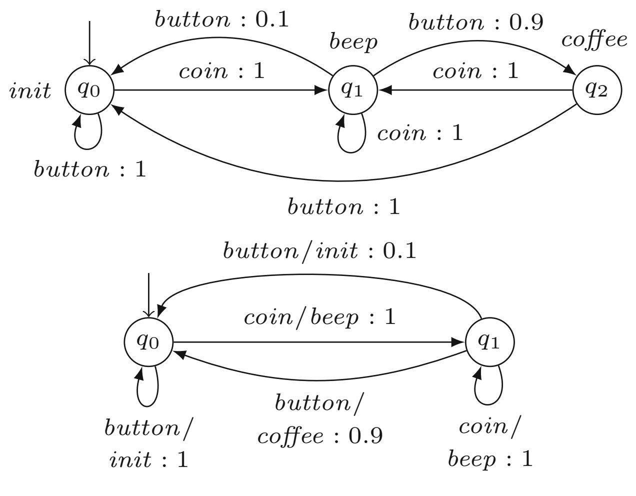

Fig. 1 An MDP model (top) and an SMM model (bottom) of a faulty coffee machine

Definition 2.2 *(Stochastic Mealy Machines)* An SMM is a tuple $\mathcal{M} = \langle Q, I, O, q_0, \delta\rangle$ where Q, q_0, I, and O are defined as for MDPs and $\delta : Q \times I \to Dist(Q \times O)$ is the probabilistic transition function.

We use $q \xrightarrow{i \cdot o} q'$ to denote $\delta(q, i)(q', o) > 0$ and extend this notation to traces t in $(I \cdot O)^*$ by $q \xrightarrow{\epsilon} q$ and $q \xrightarrow{i \cdot o \cdot t} q'$ if $\exists q'' : q \xrightarrow{i \cdot o} q'' \wedge q'' \xrightarrow{t} q'$. As is common in automata learning and testing, we consider input-enabled systems, such that δ and Δ are total and are thus defined for all $q \in Q$ and $i \in I$. Furthermore, we consider *deterministic* MDPs and SMMs, such that for every trace t, there is exactly one path producing t. More formally, we require for all $q \in Q, i \in I$ that

- $\forall q', q'' \in supp(\Delta(q, i))$: if $L(q') = L(q'')$ then $q' = q''$ (MDPs) and
- $\forall (q', o'), (q'', o'') \in supp(\delta(q, i))$: if $o' = o''$ then $q' = q''$ (SMMs).

Consequently, while an input may cause different outputs, an input–output pair cannot lead to different states. Nondeterminism results only from the environment's choice of inputs. Under these conditions, we can define an SMM output function λ by $\lambda(q, i) \in Dist(O)$ and
$\lambda(q, i) = \{o \mapsto p \mid (q', o) \mapsto p \in \delta(q, i)\}$.

These requirements enable a transformation from SMMs to MDPs similar to the one from deterministic Mealy to Moore machines. That is, we basically have to create an MDP state for every pair of (1) an SMM state s and (2) an output of one of s's incoming transitions. After creating another state with a new special label as initial state, we transfer the transitions from the SMM.

We use input–output prefix trees (IOPTs) as compact representations of a set of traces. An IOPT is a tree with edges labeled by inputs and nodes labeled by outputs. Similar trees are used in passive automata learning [18, 19].

Figure 2 shows an example of an input–output prefix tree of a faulty coffee machine shown in Fig. 1. The figure shows that we actually use IOPTs with input frequencies

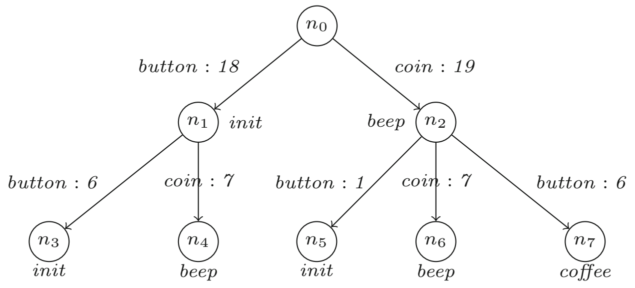

Fig. 2 An IFOPT for the faulty coffee machine

(IFOPTs) where each edge is labeled with an input symbol and a frequency value in the natural numbers. For example, the frequency of *button* in node n_2 is 7 (combined frequency from the edges labeled *button*), denoted by $freqLabel(n_2, button) = 7$. We use frequencies to define how often an input should occur in relation to other inputs. Upon receiving responses from the SUL, the algorithm updates the concrete frequency values. These frequencies continuously change during the learning process to reflect how often individual inputs should be sampled during tree query. We discuss this in more detail in Sect. 3.4.

2.3 Active automata learning

Angluin presented her L^* algorithm and introduced the minimally adequate teacher (MAT) framework in her seminal paper on active automata learning [1]. With L^* we can learn DFAs accepting regular languages and L^* has been serving as the basis for several adaptations and variations to learn further automata types, such as Mealy machines [6], non-deterministic automata [20], timed automata [21], and MDPs [12].

L^*-based algorithms in the MAT framework learn automata capturing some (regular) language L by querying a teacher for information. Generally, learners use two types of queries: (1) *input queries* for asking whether a word $u \in X^*$ is in L and (2) *equivalence queries* to check whether a hypothesized automaton accepts exactly L. Most L^*-based algorithms share furthermore a similar structure and concept of how to derive hypotheses from queried data. When learning finite-state transducers, such as Mealy machines or MDPs, an input query is substituted with a *membership query*. Whereas input queries answer the question of whether the word is in L, membership queries answer with an output obtained after executing a sequence of inputs. Figure 3 depicts the interaction between the teacher and the learning algorithm in the MAT framework.

In principle, L^* operates in rounds, such that it issues multiple membership queries in a round to gain information about L. Once the learner has sufficient information to create a hypothesis automaton (for the original L^* this is a DFA), it finishes the round by issuing an equivalence query. The teacher may now respond with *yes* for signaling that the automaton a.k.a. hypothesis accepts L, so that the learning process concludes. However, the teacher may respond also with a counterexample, i.e., a word c in the symmetric difference between L and the automaton's language. Thus, when receiving a counterexample c, the learner integrates c into its knowledge and continues via starting the next round. L^* stores information in observation tables and creates hypotheses from these tables. A table's rows are labeled by a prefix-closed set of words S and columns are labeled by a suffix-closed set of words E. The states in a hypothesis are equivalence classes in S, s.t. $s, s' \in S$ are equivalent if $s \cdot e \in L \Leftrightarrow s' \cdot e \in L$ for all $e \in E$. This equivalence relation is the Nerode relation [22], but evaluated on a finite set E. In the next section, we adapt this concept to stochastic Mealy machines.

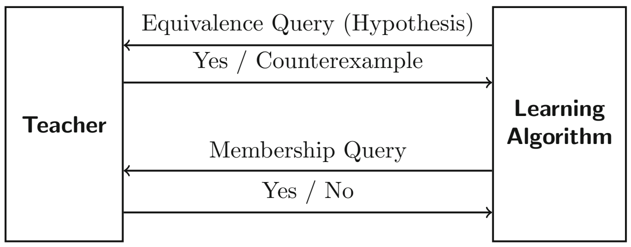

Fig. 3 The interaction between a learner and a teacher in the MAT framework

3 Method

In this section, we present L^*_{SMM}, an algorithm for learning SMMs. For this purpose, we extend, adapt, and improve our L^*-based algorithm for MDPs [11, 12]. First, we define the setting and basics for learning SMMs, such as semantics. Next, we introduce the queries that are used in the interaction between learner and teacher. After that, we discuss learning itself and present implementation details on selected aspects, such as queries, counterexample processing, and stopping. We conclude the section with a complexity analysis and a discussion of convergence.

3.1 Basics

Let us assume that $\mathcal{M} = \langle Q, I, O, q_0, \delta \rangle$ is an SMM underlying the black-box SUL, representing the knowledge we would like to learn. A learner initially only knows the available inputs and outputs, but we use the SUL behavior to formalize learning. Thus, let us define input–output semantics $[\![\mathcal{M}]\!]$ that maps traces followed by an input to the output distribution produced by the input. Formally, $[\![\mathcal{M}]\!] : (I \cdot O)^* \cdot I \to Dist(O) \cup \{\perp\}$ with $[\![\mathcal{M}]\!](t \cdot i) = \lambda(q, i)$ if there is a $q \in Q$ such that $q_0 \xrightarrow{t} q$ and $[\![\mathcal{M}]\!](t \cdot i) = \perp$ otherwise. We will query a teacher that samples SUL traces to gain information about $[\![\mathcal{M}]\!]$. For sampling, the teacher performs inputs on the SUL and observes outputs produced by the SUL, which are distributed according to $[\![\mathcal{M}]\!]$. In this way, the teacher collects multisets of SUL traces. In the following, we define trace equivalence, as an adaptation of the Nerode relation for regular languages [22], and equivalence of SMMs.

Definition 3.1 *(Trace Equivalence)* Two traces $t, t' \in (I \cdot O)^*$ are equivalent if for all continuation sequences $cs \in$

$I \cdot (O \cdot I)^*$:

$$[\![\mathcal{M}]\!](t \cdot cs) = [\![\mathcal{M}]\!](t' \cdot cs).$$

Definition 3.2 *(SMM Equivalence)* Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two SMMs over the same input and output alphabets. $\mathcal{M}_1$ and $\mathcal{M}_2$ are equivalent iff for all test sequences

$$ts \in (I \cdot O)^* \cdot I : [\![\mathcal{M}_1]\!](ts) = [\![\mathcal{M}_2]\!](ts).$$

Two traces are equivalent if they produce the same output distributions in response to the last input of every continuation sequence. Likewise, two SMMs are equivalent if they produce the same output distributions for all test sequences.

3.1.1 Statistical tests to determine trace equivalence

Since we sample traces with outputs distributed according to SUL semantics $[\![\mathcal{M}]\!]$, we cannot determine exact equivalence of output distributions as required by Definition 3.1. We instead perform statistical tests for difference between sampled output frequencies.

To formalize the problem of approximating equivalence checking between sampled frequencies, let tc_1 and tc_2 be two sequences in $(I \cdot O)^* \cdot I$ and let $\mathcal{T}$ be a multiset of traces collected from the SUL $\mathcal{M}$. We define the frequency function as $\mathbf{freq}_{\mathcal{T}}(tc) = o \mapsto \mathcal{T}(tc \cdot o)$ for $o \in O$, extend the notion of support *supp*() to frequencies, and introduce *Freq*(O) for $O \to \mathbb{N}_0$, the set of all output frequency functions. Our goal is to approximate $[\![\mathcal{M}]\!](tc_1) \neq [\![\mathcal{M}]\!](tc_2)$ by testing whether $f_1 = \mathbf{freq}_{\mathcal{T}}(tc_1)$ and $f_2 = \mathbf{freq}_{\mathcal{T}}(tc_2)$ have been sampled from different distributions. We say that the frequencies f_1 and f_2 are compatible, denoted $f_1 \approx f_2$, if they are not different. Let $n_i = \sum_{o \in O} f_i(o)$. In the special case that $n_1 = 0$ or $n_2 = 0$, we define $f_1 \approx f_2$ to hold, since then we do not have sufficient information to detect a difference.

More concretely, we implemented two difference checking strategies: (1) one based on Hoeffding bounds [23], which are also used in other stochastic automata learning algorithms [12, 19], and one based on Pearson's χ^2 test [24] for testing homogeneity of multinomial distributions.

Both strategies come with advantages and drawbacks. Using Hoeffding bounds, we view the occurrence of each output as Bernoulli-distributed random event. This means we must perform several individual tests for outputs produced in response to a single state-input pair. Hence, errors may compound. The application of Hoeffding bounds is actually motivated by its application in IoAlergia [15] and its predecessor Alergia [19]. In contrast to Alergia which learns probabilistic finite automata, we learn MDPs, where outputs follow a categorical distribution rather than a Bernoulli distribution. Hence, Pearson's χ^2 test may be a better fit. However, this test places stricter requirements on the amount of sampled data. Especially "almost deterministic" systems and systems with rare outputs, in general, may be problematic, as the χ^2 distribution may not fit well in such cases. Due to the relevance of such domain-dependent considerations, we will empirically evaluate both strategies on various systems.

Hoeffding check. For $n_1 > 0$ and $n_2 > 0$, α defining the significance level.

Frequencies f_1 and f_2 are different if $\exists o \in O$:

$$\left| \frac{f_1(o)}{n_1} - \frac{f_2(o)}{n_2} \right| > \sqrt{\frac{1}{2} \ln \frac{2}{\alpha}} \left(\frac{1}{\sqrt{n_1}} + \frac{1}{\sqrt{n_2}} \right).$$

(correct with probability $\geq (1 - \alpha)^2$ [19]) (1)

Chi-squared (χ^2). Let $O_1 = supp(f_1)$ and $O_2 = supp(f_2)$. The frequencies f_1 and f_2 are different if

$$\sum_{i=1}^{2} \sum_{o \in O_1 \cup O_2} \frac{f_1(o) - n_1 \cdot \hat{p}_o}{n_1 \cdot \hat{p}_o} \geq \chi^2_{1-\alpha, |O_1 \cup O_2| - 1} \quad (2)$$

$$\text{where } \hat{p}_o = \frac{f_1(o) + f_2(o)}{n_1 + n_2}. \quad (3)$$

There are two special cases affecting the applicability of the χ^2 test, which may occur for long sequences tc with few observations. If $O_1 \cap O_2 = \emptyset$, we apply the Hoeffding test defined above as a fallback. If $|O_1 \cup O_2| = 2$ and there is an $o \in O_1 \cup O_2$ with $f_i(o) \leq 5$, we apply the χ^2 test with the Yates correction [25].

The two different tests lead to varying learning behavior and performance. For example, if we expect mostly non-stochastic behavior from the SUL, the requirements for the standard χ^2 test may often be violated. In such cases, we apply the Yates correction, which may result in more pessimistic checks, thus requiring more samples to detect a difference. On one hand, this leads to worse performance, but on the other hand, it leads to very accurate learned models according to our experiments. The worse performance of pessimistic checks with the Yates correction stems from the fact that fewer difference verdicts means that the intermediate hypotheses have a lower number of states as more observation table rows would be compatible; see hypothesis construction below. As a result, learning might take more sampling to converge. The final models, however, may be more accurate than with the Hoeffding check simply because more data is available to estimate output distributions. Hence, the choice of difference check depends on the application domain and the time budget for learning.

3.2 Queries

The learner and teacher interact with each other via two types of queries: (1) *tree queries* and (2) *equivalence queries*. The teacher samples traces for both query types, but with different goals. Tree queries attempt to gain more accurate information about the SUL, whereas equivalence queries attempt to falsify a hypothesis SMM—as proving equivalence is not possible in a black-box setting, we target falsification.

Tree query (**tq**): Let *FT* be an IFOPT, a tree query $\mathbf{tq}(FT)$ returns a multiset of traces from $(I \cdot O)^*$, where inputs are chosen according to *FT* and outputs are sampled according to the SUL semantics $[\![\mathcal{M}]\!]$.

Equivalence query (**eq**): Let $\mathcal{H}$ be a hypothesis SMM, an equivalence query $\mathbf{eq}(\mathcal{H})$ returns a pair $(r, \mathcal{T}_{\text{cex}})$ where r is the query result in $\{yes\} \cup (I \cdot O)^* \cdot I$ and $\mathcal{T}_{\text{cex}}$ is a multiset of traces sampled for the query.

For a tree query **tq**, the learner creates an IFOPT *FT* and asks the teacher to sample paths from *FT*, while selecting inputs i with a probability proportional to the frequency value of i in *FT*. The result of a **tq** is a multiset of sampled SUL traces. For an equivalence query, the learner forms a hypothesis SMM $\mathcal{H}$ and asks if $\mathcal{H}$ is equivalent to the SUL $\mathcal{M}$. The teacher responds either with *yes* or with a counterexample to equivalence in $(I \cdot O)^* \cdot I$. Additionally, the teacher returns a multiset of traces that have been sampled to perform the query.

3.3 Learner

3.3.1 Data structures

The learner uses two main data structures, a multiset $\mathcal{T}$ of sampled traces and an observation table, a triple $\langle S, E, T \rangle$. Based on $\mathbf{freq}_{\mathcal{T}}(tc)$, the output frequencies observed so far, S, E, and T are defined by:

- $S \subseteq (I \cdot O)^*$ is a prefix-closed set of traces,
- $E \subseteq (I \cdot O)^* \cdot I$ with $I \subseteq E$ is a suffix-closed set of continuations, and
- $T : (S \cup Lt(S)) \cdot E \to Freq(O)$ with $Lt(S) = \{s \cdot i \cdot o \mid s \in S, i \in I, o \in O : \mathbf{freq}_{\mathcal{T}}(s \cdot i)(o) > 0\}$ and $T(s \cdot e) = \mathbf{freq}_{\mathcal{T}}(s \cdot e)$ stores output frequencies.

An observation table can be represented as a two-dimensional table with rows labeled by *short* traces in S and by *long* traces in $Lt(S)$, columns labeled by continuation sequences in E, and with cell content given by T. Table 1 shows an observation table from a learning run of L^*_{SMM} on the coffee machine shown in Fig. 1.

As is commonly done in L^*-based learning, we create hypotheses as follows. We partition S based on the row content given by functions $row(s) : E \to Freq(O)$ with $row(s)(e) = T(s \cdot e)$ and create a state for every block

Algorithm 1 Creation of compatibility classes

```
1: for all $s \in S$ do
2:     $\text{rank}(s) \leftarrow \sum_{i \in I} \sum_{o \in O} \widehat{T}(s \cdot i)(o)$
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 $\text{rank}(m)$
8:     $R \leftarrow R \cup \{r\}$
9:     $cg(r) \leftarrow \{s \in unpartitioned \wedge \mathbf{compatible}_E(s, r)\}$
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
```

in the partition. We further create transitions for input–output pairs $i \cdot o$ between blocks b and b' by determining the block b containing an $s \in S$ and the block b' containing its input–output extension $s \cdot i \cdot o$, if it exists. The long traces $Lt(S)$ ensure that traces can be created for all observed input–output pairs. To partition S, we extend the notion of frequency compatibility to rows and adapt a technique introduced in previous work [12], as compatibility is not an equivalence relation. We say that two rows labeled by traces s and s' are compatible if all their cells are compatible, i.e., $\forall e \in E : row(s)(e) \approx row(s')(e)$. We also say that the traces s and s' are compatible. In Table 1, the first, third, fourth, and fifth rows contain different frequencies, but by normalizing we can see that they model exactly the same (empirical) probability distributions. Hence, they would be pairwise compatible and thus be in the same block of a partition of S and correspond to the same state in the hypothesis derived from Table 1. The situation for the second and last row is a bit different. Their respective first cells contain the same outputs, but the empirical probability distributions estimated from them are different. However, the difference is small enough that we would deem the cells, and thus the rows, compatible according to both statistical tests. As an example where the tests disagree, suppose that the cells contain $\{coffee\colon 147, init\colon 7\}$ and $\{coffee\colon 65, init\colon 16\}$, respectively, i.e., frequencies values would be swapped. Using an $\alpha = 0.05$ for both, the Hoeffding check would deem the cells compatible, whereas the χ^2-squared tests would deem them incompatible. The reason is that the former looks at outputs individually, whereas the latter looks at the whole cell at once, thus noticing a difference. Hence, the Hoeffding check would be more aggressive, differentiating fewer rows and thus creating a model with fewer states.

We face the difficulty that compatibility is not an equivalence relation, as transitivity does not hold in general. A row may be compatible with multiple other rows that are not necessarily pairwise compatible. To tackle this challenge, we create compatibility classes $cg(r)$ that partition S, like in our

Table 1 Observation table for the faulty coffee machine shown in Fig. 1

		button	*coin*
S	ϵ	$\{init: 247\}$	$\{beep: 414\}$
	$coin \cdot beep$	$\{coffee: 147, init: 16\}$	$\{beep: 134\}$
$Lt(S)$	$button \cdot init$	$\{init: 69\}$	$\{beep: 82\}$
	$coin \cdot beep \cdot button \cdot coffee$	$\{init: 64\}$	$\{beep: 53\}$
	$coin \cdot beep \cdot button \cdot init$	$\{init: 9\}$	$\{beep: 6\}$
	$coin \cdot beep \cdot coin \cdot beep$	$\{coffee: 65, init: 7\}$	$\{beep: 61\}$

Algorithm 2 Ensuring closedness and consistency of an observation table

```
function MakeClosedAndConsistent($\langle S, E, T\rangle$)
    if $\langle S, E, T\rangle$ is not closed then
        $l \leftarrow l' \in Lt(()S)$ such that $\forall s \in S : row(s) \neq row(l')$
        $S \leftarrow S \cup \{l\}$
    else if $\langle S, E, T\rangle$ is not consistent then
        for all $s_1, s_2 \in S$ such that eqRow$_E(s_1, s_2)$ do
            for all $i \in I, o \in O$ do
                if $T(s_1 \cdot i)(o) > 0$ and $\neg$eqRow$_E(s_1 \cdot i \cdot o, s_2 \cdot i \cdot o)$ then
                    $e \leftarrow e' \in E$ such that $T(s_1 \cdot i \cdot o \cdot e') \neq T(s_2 \cdot i \cdot o \cdot e')$
                    $E \leftarrow E \cup \{i \cdot o \cdot e\}$
                end if
            end for
        end for
    end if
    return $\langle S, E, T\rangle$                ▷ Closed and consistent obs. table
end function
```

previous work [12]. Each compatibility class $cg(r) \subseteq S$ has a unique representative r in the set of representatives $R \subseteq S$ and every trace t in $cg(r)$ is compatible to r. The function $rep(t) = r$ returns the unique representative r for short and long traces t. We create compatibility classes by iteratively selecting a new representative r from the unpartitioned part of S and greedily adding other unpartitioned traces to $cg(r)$. The selection of a new representative r is based on how often r was observed during sampling, which ensures that $\epsilon \in R$ can be used as the initial hypothesis state. Algorithm 1 shows the compatibility classes computation process.

3.3.2 Hypothesis generation

To create a hypothesis from an observation table $\langle S, E, T\rangle$, the table must be closed and consistent. Adapting the standard notion of closedness and consistency [1, 12], we say that an observation table is *closed* if for all $l \in Lt(S)$ there is an $r \in R$ such that r and l are compatible. Closedness ensures that we can create transitions for all inputs in all states.

An observation table is *consistent* if for all pairs of compatible short traces $s, s' \in S$ and all input–output pairs $i \cdot o \in I \cdot O$: either (1) $s \cdot i \cdot o$ and $s' \cdot i \cdot o$ are compatible or (2) $T(s \cdot i)(o) = 0$ or $T(s' \cdot i)(o) = 0$. Consistency requires that the extensions of compatible traces are also compatible. Put differently, if s leads to the same hypothesis state as s', then its extension $s \cdot i \cdot o$ should lead to the same state as $s' \cdot i \cdot o$. This corresponds to the determinism requirement of SMMs.

When an observation table is not closed, there is an $l \in Lt(S)$ such that there is no compatible $r \in R$. We can make an observation table closed by adding such an l violating closedness to the set of short traces S and recalculating the set R. When an observation table is not consistent, there exists a pair of compatible traces s, s', an input–output pair $i \cdot o$, and a column sequence e such that $s \cdot i \cdot o \cdot e \not\approx s' \cdot i \cdot o \cdot e$. In such a case, we add $i \cdot o \cdot e$ to E. The iterated application of these updates—adding traces to S and sequences to E—eventually establishes closedness and consistency. Algorithm 2 depicts the process of making the table closed and consistent. In contrast to deterministic learning, this does not require resampling as compatibility is defined for any amount of samples.

Given a closed and consistent observation table $\langle S, E, T\rangle$ with representatives R, we derive a hypothesis SMM $\text{hyp}(S, E, T) = \langle Q_h, I, O \cup \{undef\}, q_{0h}, \delta_h\rangle$ via:

- $Q_h = R \cup \{q_{undef}\}$ and $q_{0h} = \epsilon$
- For $q \in R$ and $i \in I$, if $\sum_{o \in O} T(q \cdot i)(o) = 0$:

  $$\delta_h(q, i)((q_{undef}, undef)) = 1$$

  Otherwise:

  $$\delta_h(q, i) = \mu \text{ where for } o \in O \text{ if } T(q \cdot i)(o) > 0\text{:}$$
  $$q' = rep(q \cdot i \cdot o) \text{ and } \mu((q', o)) = \frac{T(q \cdot i)(o)}{\sum_{o' \in O} T(q \cdot i)(o')}$$

- If q_{undef} is reachable, then for $i \in I$:

  $$\delta_h(q_{undef}, i)((q_{undef}, undef)) = 1$$

We create a state for every representative with $\epsilon \in R$ being the initial state. Transitions from $q \in R$ lead to the representatives of the input–output extensions $q \cdot i \cdot o$, where transition probabilities are estimated from T. If we have no observations for an input, we create transitions to a sink state q_{undef}.

3.3.3 Learning algorithm

Algorithm 3 implements the stochastic L^* algorithm for SMMs, adapting L^* for MDPs [12]. In Line 1, we initial-

Algorithm 3 The main algorithm implementing L^*_{SMM}

```
Input: input alphabet $I$, teacher capable of answering tq and eq
Output: final learned model $\mathcal{H}$
1: $S \leftarrow \{\epsilon\}, E \leftarrow I, T \leftarrow \{\}, \mathcal{T} \leftarrow \{\epsilon\}$   ▷ initialize observation table
   and samples $\mathcal{T}$
2: $round \leftarrow 0$
3: repeat
4:     $round \leftarrow round + 1$
5:     $FTree \leftarrow$ CREATEIFOPT($\langle S, E, T \rangle$)
6:     $\mathcal{T} \leftarrow \mathcal{T} \uplus$ tq($FTree$)
7:     for all $s \in S \cup Lt(S), e \in E$ do
8:         $T(s \cdot e) \leftarrow$ freq$_{\mathcal{T}}(s \cdot e)$   ▷ update observation table
9:     end for
10:    while $\langle S, E, T \rangle$ not closed or not consistent do
11:        $\langle S, E, T \rangle \leftarrow$ MAKECLOSEDANDCONSISTENT($\langle S, E, T \rangle$)
12:    end while
13:    $\mathcal{H} \leftarrow$ hyp($S, E, T$)   ▷ create hypothesis
14:    $\langle S, E, T \rangle \leftarrow$ TRIM($\langle S, E, T \rangle, \mathcal{H}$)   ▷ remove cells not needed
15:    $(r, \mathcal{T}_{cex}) \leftarrow$ eq($\mathcal{H}$)   ▷ Check hypothesis $\mathcal{H}$ against SUL $\mathcal{M}$
16:    $\mathcal{T} \leftarrow \mathcal{T} \uplus \mathcal{T}_{cex}$
17:    if $r \neq yes$ then   ▷ we found a counterexample
18:        $\langle S, E, T \rangle \leftarrow$ PROCESSCEX($r, \langle S, E, T \rangle$)
19:    end if
20: until STOP($\langle S, E, T \rangle, \mathcal{H}, round$)
21: return $\mathcal{H}$   ▷ output final hypothesis
```

ize the learning data structures. The main loop starts with a tree query in lines 5 and 6. After updating the learner's data structures, we make the observation table closed and consistent (Line 11) and form a hypothesis $\mathcal{H}$ (Line 13). Given $\mathcal{H}$, we remove table rows and columns that are not needed for hypothesis generation. Line 14 basically removes rows that carry the same information as other rows and cells that do not distinguish rows. For more details, we refer to our previous work [12].

In Line 15, we perform an equivalence query. If it returns a counterexample r, we process it by updating the observation table with information derived from r. L^*-based learning [1, 6] commonly stops once an equivalence query returns *yes*, but we continue learning until the stopping criterion in Line 20 is fulfilled. The reason is that in stochastic learning, we may not be able to find a counterexample given an inaccurate hypothesis that could be improved by additional tree queries. Therefore, we employ a stopping criterion that takes hypothesis generation into account. Once we stop, we return the final hypothesis.

3.4 Implementation

We implemented Algorithm 3 in AALPY, an open-source automata learning library.[1] AALPY supports learning of both MDPs and SMMs, which we compare empirically in Sect. 4. In the following, we discuss selected aspects of the implementation with a focus on improvements over the original algorithm for MDPs [12], for example, stopping and resampling by the tree query.

3.4.1 SUL interface

The teacher assumes an application-specific SUL interface comprising two operations **step** and **reset**. They facilitate sampling, where at each point, the SUL is in a current state q_c. The **reset** operation resets q_c to q_0. The **step** operation takes an input i as parameter, executes it on the SUL, and returns the SUL output. More concretely, it samples a state-output pair (q, o) according to the distribution $\delta(q, i)$ of the SUL, sets $q_c = q$, and returns o.

3.4.2 Equivalence queries and counterexample processing

The teacher performs two steps in equivalence queries. The first step is checking compatibility between already sampled traces (multiset $\mathcal{T}$ in Algorithm 3) and the hypothesis. The second, optional step samples new traces to reveal a counterexample to equivalence between hypothesis and SUL. Sampling happens only when the compatibility check does not reveal a counterexample. This ensures that we use existing samples as efficiently as possible and when there is no counterexample in $\mathcal{T}$ we try to find new counterexamples via sampling. The implementation of these steps follows our previous work [12]. We check compatibility between $\mathcal{T}$ and the hypothesis using Eq. 1, and we apply random testing for sampling. To ensure that every counterexample can be detected, every input and every trace length has a nonzero probability of being selected during testing.

A counterexample c returned from an equivalence query indicates that the observation table shall be extended in a way to ensure that upcoming hypotheses are correct w.r.t. c. Since hypotheses in active automata learning [1, 12] are generally the smallest models consistent with the queried information, the goal of counterexample processing is to reveal new states.

Due to the uncertainties found in the learning process, the counterexample processing might not reveal the additional states in the current learning round. Therefore, before each equivalence query we check whether the last returned counterexample has been successfully processed. If yes, the equivalence query continues, and if not it is reused until it fails to serve as a counterexample.

There are various ways to process counterexamples in L^*-based learning. Our implementation provides three counterexample processing strategies that are commonly applied in deterministic learning.

- *Angluin-style:* Angluin adds all prefixes of a counterexample to S [1].

[1] An interactive example illustrating the learning process is available at https://github.com/DES-Lab/AALpy/blob/master/notebooks/MDP_and_SMM_Example.ipynb.

- *Longest-prefix:* The *longest prefix* strategy by Shahbaz and Groz [6] splits c into a prefix p and a suffix e, where p is the longest prefix that is already in S. It then adds e to E.
- *Rivest and Shapire*: The processing strategy proposed by Rivest and Shapire [2] processes a counterexample and finds the shortest suffix e that reveals the difference between the SUL and the current hypothesis. The suffixes of e are added to the E set.

While *Rivest and Shapire* and alternative strategies based on the extraction of so-called distinguishing suffixes are very efficient in deterministic learning [3, 26], we found them to be (generally) less efficient in stochastic learning. This is due to the usually low amount of statistical information on counterexample suffixes. Such strategies would require repeated sampling of representative traces from R concatenated with counterexample suffixes until a distinguishing suffix can be found via Eq. 1. In contrast, the other two techniques rely solely on sampling performed in subsequent learning rounds.

We want to illustrate the potential sampling cost of a technique, such as *Rivest and Shapire*'s counterexample processing using a hypothetical example. Suppose we have a counterexample of length 16, which we would split into a prefix, a single action a, and a suffix. By replacing the prefix with one of the representative traces and concatenating it again with a and the suffix, we might end up with a trace t of length 8. In the case that there are only two outputs with equal probability, it would take on average 256 tries to sample t once. Without going into more details, it should be noted that at the same splitting point, a second trace needs to be sampled and both traces need to be sampled repeatedly for statistical significance. Additionally, a binary search on the counterexample is necessary to determine the correct splitting point. It can be seen that sampling costs would grow fast when individual traces need to be sampled. For the same reason, L^*_{SMM} applies tree queries, where one of the multiple traces gets resampled. If the system under learning is mostly deterministic, sampling a desired trace may be significantly less costly, thus increasing the efficiency of counterexample analysis. In such cases, L^*_{SMM} might further benefit from the technique presented by Howar and Steffen [27], where data structures are extended in a more efficient way in response to counterexamples.

Counterexample length is generally an important factor in the active learning of stochastic systems. Long input–output sequences are hard to sufficiently sample especially when the system under learning is stochastic. This is exacerbated by the fact that counterexample processing techniques from deterministic learning are hardly applicable, as we outlined above. To mitigate this, we sort all the test cases executed during equivalence queries according to their length before counterexample processing. This heuristic helps to identify shorter, easier-to-sample, counterexamples.

Algorithm 4 *Tree* query

```
Input: IFOPT FTree, SUL with reset and step
Output: a sampled trace t
1:  node ← root(FTree), t ← ε                                ▷ initialize
2:  reset()                                                  ▷ reset SUL
3:  loop
4:      $freqSum \leftarrow \sum_{i \in I} freqLabel(node, i)$          ▷ sum frequencies in IFOPT
5:      $inputDist \leftarrow \left\{ i \mapsto p \mid i \in I, p = \frac{freqLabel(node,i)}{freqSum} \right\}$
6:      $in \leftarrow$ CHOOSE($I$, $inputDist$)                        ▷ choose input
7:      $out \leftarrow$ step($in$)                       ▷ execute SUL and observe output
8:      $t \leftarrow t \cdot in \cdot out$                                  ▷ extend traces
9:      if $\nexists n \in nodes(FTree) : node \xrightarrow{in/out} n$ then      ▷ did we leave the tree?
10:         return trace
11:     end if
12:     $node \leftarrow n$ with $node \xrightarrow{in/out} n$                 ▷ walk down one tree level
13: end loop
```

3.4.3 Tree queries

Membership queries in L^* provide information about newly added sequences in the observation table. Tree queries have an analogous purpose. They gather more information on sequences that are in the observation table. While deterministic learning requires a single query (sample) for every sequence, uncertainties affect stochastic learning. We address this issue by sampling traces with the goal of reducing uncertainties.

Uncertainties in stochastic L^* mainly arise from the difference tests and the derived compatibility relation. As discussed in the context of hypothesis generation, this relation is not necessarily an equivalence relation for finite sample sizes. The resulting uncertainties directly affect hypothesis generation, as a trace in $S \cup Lt(S)$ may be compatible to multiple other traces that are not pairwise compatible. Hence, the target state of a transition may be ambiguous. In particular, a trace may be compatible to multiple compatibility class representatives. We devise a sampling strategy with the goal of reducing this form of ambiguity, in order to learn the correct model structure. We start from the viewpoint of the learner and then present the teacher's tree query implementation.

Given an observation table $\langle S, E, T \rangle$ and $se \in (S \cup Lt(S)) \cdot E$, we assign an uncertainty value *uncert(se)* to se as follows. Let s be the longest prefix trace of se s.t. $s \in S \cup Lt(S)$ and let the number of compatible representatives be

$$cr(s) = |\{r \in R \mid r \approx s\}| \text{ in}$$
$$uncert(se) = \max(2 \cdot (cr(s) - 1), 1).$$

The rationale behind *uncert(se)* is that every trace should be compatible with at most one representative. Since we are only interested in the compatibility of rows, we consider the longest trace that labels a row and is a prefix of se. The uncertainty grows with the number of compatible representatives cr, where the multiplication by two puts more weight on this number. We further subtract one, as every trace is compatible with at least one representative in a closed observation table. Finally, we ensure *uncert(se)* is larger than or equal to one for two reasons. We may spuriously conclude that $cr = 1$, i.e., that we are certain that is a unique compatible representative. Due to limited data, this representative may not be the right one corresponding to se. Hence, further sampling should validate if the representative for se is the correct one, which we ensure by having *uncert(se)* ≥ 1. Furthermore, we account for the estimation of transition probabilities as another source of uncertainty affecting hypothesis generation in general. Hence, even if we are certain about the representatives, i.e., the MDP structure, every trace should have a nonzero probability of being sampled. With that, we further improve the accuracy of transition probabilities through extended sampling.

Now, we can define the function `createIFOPT` in Line 5 of Algorithm 3.

1. *Trace creation.* Extend the sequences in $(S \cup Lt(S)) \cdot E$ to traces by adding a special output $leaf \notin O$ at the end of every sequence, let $Tr = \{s \cdot e \cdot leaf \mid s \in S \cup Lt(S), e \in E\}$.
2. *IOPT creation.* Create an IOPT $Tree$ from the traces in Tr.
3. *IFOPT initialization.* Create an IFOPT *FTree* from $Tree$ by initializing every input frequency with zero.
4. *Adding frequencies.* For each $se \in (S \cup Lt(S)) \cdot E$: add *uncert(se)* to the frequency of every input on the path from the root node to the last edge reached by se.

The frequency label for a given edge *ed* in *FTree* is the sum of uncertainty values *uncert(se)*, for sequences se traversing *ed* when starting from the root. Aside from the IFOPT *FTree*, the implementation of tree queries takes another parameter n_{tree} that defines the number of traces to be sampled by the teacher. We determine n_{tree} proportional to the uncertainty and observation table size via

$$n_{tree} = \left\lfloor \frac{\sum_{se \in (S \cup Lt(S)) \cdot E} uncert(se)}{2} \right\rfloor. \tag{4}$$

Roughly speaking, we take one sample for every unambiguous cell in the table and additional samples for ambiguity.

The teacher performs tree queries by sampling n_{tree} traces corresponding to directed random walks on the IFOPT *FTree* created by the learner. Algorithm 4 implements this form of sampling. It starts with an initialization in lines 1 and 2. The sample loop starts with the selection of an input *in* in lines 4 to 6, where the selection probability of *in* is proportional to the frequency assigned to *in*. Here, we use $\text{CHOOSE}(I, d)$ to sample an input in I according to a probability distribution d. Next, we execute *in* on the SUL and extend the sampled trace t with *in* and the SUL output. When there is no path in the IFOPT corresponding to t, we return t. This is guaranteed to happen when reaching a leaf, as leaves are labeled with a symbol not in the output alphabet.

3.4.4 Stopping

Similarly to tree queries, stopping takes ambiguity into account. We stop, when ambiguity decreases. For stopping, we quantify ambiguity or rather the absence thereof as the number of row traces that have a single compatible representative. This number *unambiguity* is given by:

$$unambiguity = \frac{|\{s \in S \cup Lt(S) \mid cr(s) = 1\}|}{|S \cup Lt(S)|}$$

We also used this value to decide when to stop learning in our previous work [12]. Previously, we stopped learning once *unambiguity* was greater than a fixed threshold. However, we concluded from experiments that a fixed threshold is not an ideal choice for SMM learning concerning efficiency. In these experiments, we measured hypothesis accuracy in relation to the value of *unambiguity* during the course of learning. For this purpose, we quantified accuracy as the average error made in model checking computations, as compared to model checking performed on the true model of the SUL. We observed a general trend that the hypothesis accuracy converges at the same time when *unambiguity* converges to a plateau.

Figure 4 shows plots of the *unambiguity* value, the maximum error in various model checking computations, and the corresponding average error on two examples that we discuss in more detail in Sect. 4. The x-axis displays the number of learning rounds. We can see that upon reaching an *unambiguity* value of approximately 0.8, the maximum and average errors are close to zero and stay close to zero. At this point, further learning and sampling costs resources, but does not contribute to the accuracy of the model. For this reason, we stop learning when we detect that *unambiguity* reaches a plateau, i.e., the difference between several consecutive *unambiguity* values is below a small positive ϵ.

In common with previous work [12], we do not stop when q_{undef} is reachable in the hypothesis. This state is reachable, when there is a state-action pair for which there is no information at all. Additionally, it is possible to specify a minimum number of rounds and a maximum of rounds. The

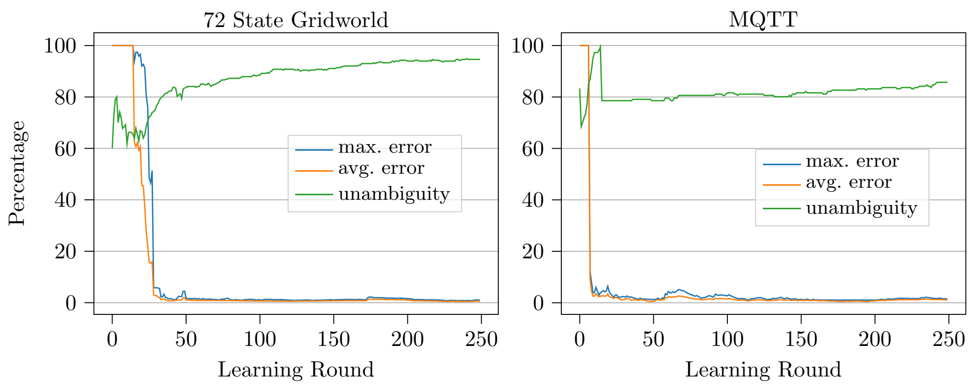

Fig. 4 Relation between number of unambiguous rows and hypothesis accuracy

early stopping criterion can also be disabled in favor of a fixed threshold.

3.5 Analysis of L^*_{SMM}

3.5.1 Complexity

In the following, we will analyze the complexity of individual operations performed by L^*_{SMM}, such as queries. For this purpose, let m be the length of the longest sampled trace, let n be the number of sampled steps, and let k be the number of different sampled traces. In the worst case, k grows linearly in n. For simplicity, we consider the set of sampled traces to be prefix-closed, because whenever we observe a trace, we observe all its prefixes as well.

Since we add only traces to S that have been sampled at least once, we can bound the length of sequences in S by m and the cardinality of S by k. We can analogously bound the length of sequences in E and the cardinality of E, as E contains suffixes of sampled traces. Making an observation table closed and consistent requires time in $O(k^3 \cdot |O|)$. Checking consistency requires iterating over all pairs of rows and checking compatibility for each cell. There are at most k^2 row pairs and at most k columns (cells in each rows) and each compatibility check requires $|O|$ computations (see Eq. 1). Additionally checking the extensions of compatible rows only adds a constant factor. Hence, the runtime is in $O(k^3 \cdot |O|)$. Fixing a consistency violation simply amounts to adding a new element to the E set. Checking closedness requires compatibility checks between every pair consisting of a long and a short row. Thus, closedness checks are in $O(k^3 \cdot |O|)$ as well. Creating a hypothesis from an observation table is in $O(k \cdot |I| \cdot |O|)$. We need to potentially create a transition for every input–output pair from every state. There are $|I| \cdot |O|$ such pairs and there are at most k states.

The IFOPT creation for tree queries takes time in $O(k \cdot m)$, as every unique trace of length at most m is added at most once. The value n_{tree} of traces sampled during a tree query (see Eq. 4) is at most k^2, but generally much lower. Hence, a tree query performs at most $k^2 \cdot m$ sampling steps.

Equivalence queries consist of two steps, checking compatibility and sampling traces to actively check for equivalence between hypothesis and SUL. The amount of sampling can be adjusted freely according to the sampling budget and accuracy requirement. The compatibility check between sampled information and hypothesis requires runtime in $O(k \cdot m \cdot |O|)$. For every sampled trace t we determine a hypothesis state and its representative trace r, which has a length of at most m. For t and r, we perform a compatibility check, which takes time linear in the number of outputs. This analysis matches our experience in that making observation tables closed and consistent takes the most time. While the analysis of sampling steps performed by tree queries provides a conservative upper bound, we observe that tree queries perform thorough sampling leading to accurate models.

We use a heuristic to decide stopping, which prevents stating a complexity bound on entire runs of L^*_{SMM}.

3.5.2 Convergence

Under mild, common assumptions on equivalence queries (every trace must have a nonzero probability to be sampled), the learned model converges in the limit, i.e., with stopping disabled, to a minimal SMM equivalent to the SUL. A proof of convergence can be adapted from [12]. In the next section, we empirically analyze the accuracy of models learned from finitely many traces.

4 Evaluation

In order to evaluate our method, we conducted a study considering six different stochastic systems that have been used also in previous work [12, 28]. More concretely, we simulated known models of stochastic reactive systems through a **reset** and a **step** operation, thus treating them as black boxes. After learning models from the simulations, we measured the accuracy of the learned models. As a measure of accuracy, we used the absolute difference between the probabilities computed by model checking properties on a learned model and the corresponding known true model. In the following, we report this form of error averaged over several properties for each system.

Systems used as a basis for learning are: a 35-state and a 72-state *Gridworld* [12], the *slot-machine* previously used in [15, 29], parts of the *MQTT* protocol encoded as a 61-state MDP, parts of the *TCP* protocol encoded as a 151-state MDP, and a *Bluetooth Low-Energy* device encoded as a 156-state MDP. For the first three systems, we used model checking properties also used in previous work [12] to enable a direct comparison. We derived the *MQTT*, *TCP*, and *Bluetooth low-energy* models from learned deterministic models [10, 30, 31] by injecting stochastic faults into the model, denoted as a "crash" state (see also [28]). More concretely, we added probabilistic behavior at selected places in the deterministic models, such that previously deterministic transitions now have a chance of leading to a "crash" state. For this purpose, we randomly selected a subset of the transitions in the deterministic models and split each of them into two transitions. One transition of each such pair produces the original, correct output and occurs with a probability $1 - p_c$, whereas the other one produces a newly introduced *crash* output and occurs with probability p_c, leading to a "crash" state. This allows us to efficiently simulate network protocols and allow us to reason about the applicability of L^*_{SMM} in real-world scenarios. Throughout the remainder of this section, we will reason about the benchmark model's underlying structure and how it affects learning performance.

To verify the accuracy of the learned models, we derived a set of reachability properties for each model. More precisely, those properties check for the bounded reachability of reaching a fault or a goal state in the models with varying bounds. Correct property values were computed on the original model and on the learned models, with the error being the absolute difference between the results. To compute the values, we used probabilistic model checking with PRISM and statistical model checking [32]. The former computes the maximum probability of satisfying the property, quantified over all policies for choosing actions. For this reason, we denote the results with $\mathbb{P}_{MAX}$. For statistical model checking, we sample actions according to the uniform distribution and estimate the probability of satisfying properties under such a random policy. Therefore, we denote the results with $\mathbb{P}_{RAND}$. Concretely, we apply a Monte Carlo simulation for the estimation with a Chernoff bound [33] with an error bound of 0.01 and a confidence of 0.995. We use statistical model checking as a means of getting an insight into the accuracy of learned models for longer traces, as maximum probabilities computed via probabilistic model checking often equal or are close to 1 for long traces.

All experiments have been performed with our implementation in the open-source library AALPY v1.2.7 running on a Dell Latitude 5410 with an Intel Core i7-10610U processor, 8 GB of RAM running Windows 10 and using Python 3.9. We configure the Hoeffding-bound-based difference check with a constant $\alpha = 0.05$ and we limit the number of traces sampled during equivalence queries by $n_{cextraces} = 150$. In comparison to previous work [12], we reduced the number of configurable parameters, while achieving robust learning performance. In addition, all benchmark models can be found on AALPY official repository.[2] Due to the stochastic nature of the experiments, we repeated every experiment, consisting of learning and model checking, 20 times and reported average results.

4.1 Comparing MDP and SMM learning

The contributions made in this manuscript are twofold. Firstly, a formalization of SMMs and adaptation of L^*_{MDP} that is able to learn them, and an overall algorithmic improvement of the original algorithm. In Sects. 4.1 to 4.5 we focus on the comparison of MDP and SMM learning with new and improved algorithm, while in Sect. 4.6 we compare L^*_{SMM} with its predecessors.

To compare the performance of MDP and SMM learning, we measured the learning time and the number of traces sampled by the teacher. While total learning time can be used to evaluate the feasibility of our approach in simulated environments, it is important to note that in real-world non-simulated environments, sampling of traces, i.e., the interaction with the SUL, is the most time-consuming aspect of active automata learning. Table 2 summarizes the results of our benchmarking study. We see that for all examples but *slot machine* SMMs are the preferred formalism.

Table 3 shows detailed results of learning MDPs and SMMs of the 72-state *Gridworld*. In this example, learning SMMs reduces the state space by about 75%, as the SUL can be modeled by a 42-state SMM. This size difference accounts for the better and faster learning of SMMs compared to MDPs. Note that on average both learned the underlying model accurately, with the maximum average error for MDP learning being 7.1% and 0.9% for SMM learning.

Figures 5 and 6 depict the effect of the property bound in the reachability analysis, both with a property-based model checking (Fig. 5) and with statistical model checking [32] (Fig. 6). These experiments show that learned models remain accurate with the increasing property bound, meaning that the learned model remains true to the system under learning even for longer, harder-to-sample, test sequences.

More concretely, Fig. 5 depicts the value of reaching the "crash" state in the *Bluetooth Low-Energy* model. We observe that the accuracy of learned models remains consistent with the increasing bound, with learned SMM being slightly more precise (interestingly, differences in absolute

[2] https://github.com/DES-Lab/AALpy.

Table 2 Comparison of MDP and SMM learning

	Learning time (s)			# Traces		
	MDP	SMM	Improvement (%)	MDP	SMM	Improvement (%)
35-State Grid	8.61	5.67	51.85	68,736	44,954	52.9
72-State Grid	40.92	29.85	37.09	199,789	113,690	75.73
MQTT	8.59	6.97	23.24	68,964	39,904	72.82
TCP	21.94	13.4	63.73	98,333	54,514	80.38
Bluetooth	17.26	6.98	60.91	94,525	35,605	165.48
Slot Machine	60.65	306.4	−80.21	456,243	622,285	−26.86

Table 3 Results for learning models of the 72-state Gridworld

	true	MDP	SMM	Error MDP	Error SMM	Improvement
# Traces	–	199,789	113,690	–	–	75.73%
# Steps	–	1,363,233	762,338	–	–	78.82%
$\mathbb{P}_{\max}(F^{\leq 14}(goal))$	0.9348	0.931	0.94	0.0038	0.0052	− 26.92%
$\mathbb{P}_{\max}(F^{\leq 12}(goal))$	0.6712	0.6943	0.681	0.0231	0.0098	135.71%
$\mathbb{P}_{\max}(\neg M\ U^{\leq 18}(goal))$	0.9743	0.9721	0.973	0.0022	0.0013	69.23%
$\mathbb{P}_{\max}(\neg S\ U^{\leq 20}(goal))$	0.1424	0.2138	0.1482	0.0714	0.0058	1131.03%

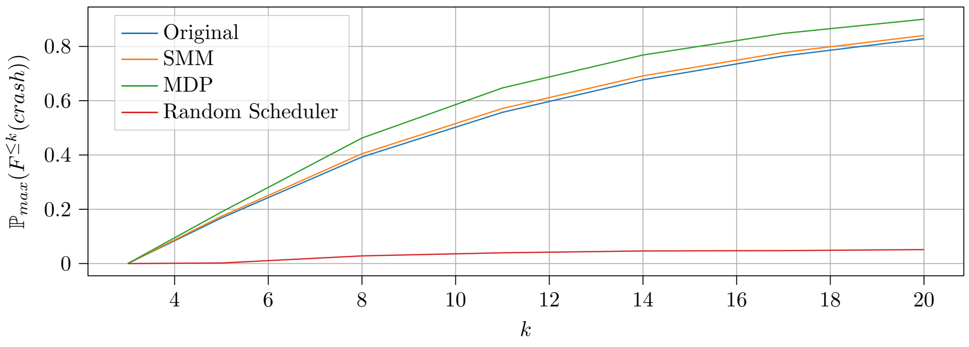

Fig. 5 Maximum probabilities of reaching a “crash” state on an original and learned models within a bounded number of steps

errors between learned MDP and SMM models are, while small, statistically significant, according to the p-value of Student’s t-test [34]) than the learned MDP while requiring significantly fewer learning queries (as shown in Table 2). Figure 6 depicts the probability of reaching a “goal” state on the 72-state Gridworld with an increasing step bound. These results are consistent with those shown in Fig. 5, in that the accuracy of learned models remains high with the increasing property bound. However, in this case learned MDP is statistically significantly more accurate (with respect to the absolute error) than learned SMM, albeit with higher learning costs.

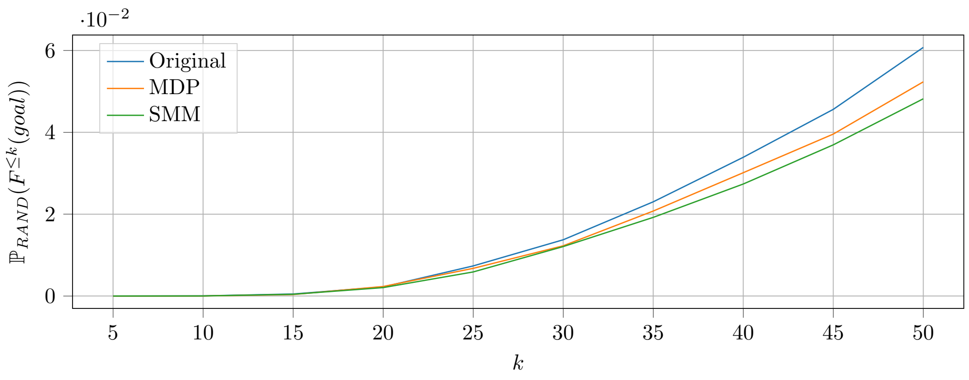

Fig. 6 Probability of reaching a “goal” state in the 72 state Gridworld within k steps. Probabilities are computed using Monte Carlo simulation with a random scheduler

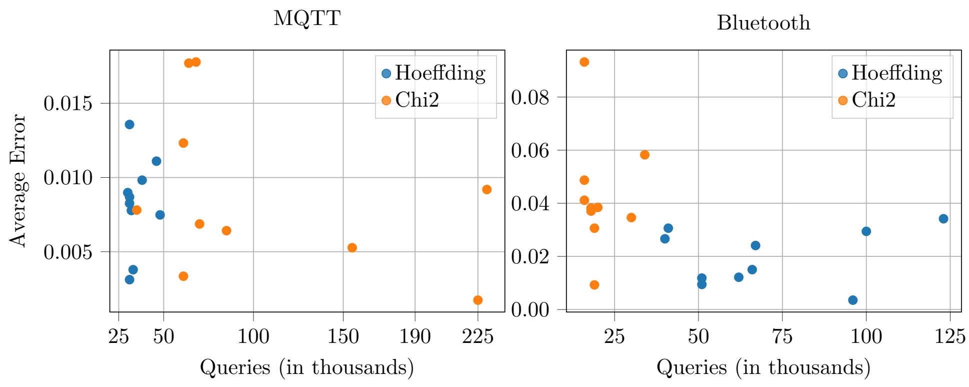

Fig. 7 Accuracy and learning costs comparison between Hoeffding and χ^2 statistical compatibility checks

4.2 Comparison of statistical compatibility checks

In Sect. 3.1.1 we outlined two statistical compatibility tests to determine whether the two traces are equivalent. We compare the effect of the different statistical compatibility checks on the overall L^*_{SMM} learning process. In both compatibility checks, α was set to 0.001, increasing the confidence in the verdicts of the compatibility checks. With all other settings being constant, we repeated the learning process multiple times for each statistical compatibility test and compared the number of traces executed during learning and the average error over all properties.

Figure 7 depicts the results of this comparison. Experiments were repeated for MQTT and Bluetooth Low-Energy models. Firstly, we observed that both statistical compatibility checks lead to accurate models, with a maximum average error being 1.5% and 9%. From the results, we make the following observations: (1) more queries do not necessarily lead to more accurate models and (2) that there is more variability in the learning results when using χ^2 compatibility checks. Hoeffding compatibility checks appear to be slightly more preferable, given that it tends to lead to a lower learning error while keeping the learning costs low.

In addition, these findings show that the learning algorithm is not dependent on the Hoeffding compatibility check, but can be used with other statistical compatibility checks as well. In future work, it would be interesting to implement and further study the effects of χ^2 and other statistical compatibility tests on other stochastic automata learning algorithms, such as IoAlergia.

4.3 Comparison of counterexample processing strategy

In Sect. 3.4.2 we outlined three counterexample processing strategies. To evaluate their effect on the learning process, we compared them on the previously described benchmark models. As stated in Sect. 3.4.2, we sort test cases executed by an equivalence query by length, so that we increase the probability of finding short counterexamples.

Results from the counterexample processing evaluation show that the selection of a counterexample processing strategy in a stochastic setting influences the sampling efficiency of the learning process. However, the selection of an appropriate counterexample processing strategy is influenced by the topology and transition probabilities of the SUL, which are unknown in a black-box setting. Therefore, we propose the usage of the *Angluin-style* counterexample processing, as it consistently performed well without major performance outliers. The other two methods, especially *Rivest and Schapire* could be more appropriate in settings where we know apriori that the majority of SUL transitions are deterministic. This is an important consideration as *Longest-prefix* and *Rivest and Shapire* add suffixes to the E set of the observation table, therefore requiring fewer, but significantly longer queries compared to *Angluin-style*, which adds prefixes of the counterexamples to the S set. Such long queries are in turn hard to sample, which usually prolongs the learning process.

More concretely, Table 4 shows the influence of the counterexample processing on the total number of traces sampled during the learning and equivalence checking process. From the experimental results, which are averaged over multiple execution runs, we observe that the *Angluin-style* counterexample processing is consistently relatively sample efficient without major outliers, while the *Longest-prefix* counterexample processing performed better on network protocol benchmarks (MQTT, TCP, Bluetooth). Although the performance of *Rivest and Shapire* is acceptable for network protocols, we observed that it might lead to an increase in total number of traces sampled for other benchmarks, mostly due to additional sampling required by *Rivest and Schapire*. Note that the accuracy of learned models was high, regardless of the counterexample processing method, implying that the selection of the counterexample processing method does not impede convergence to the true model.

Table 4 Comparison of the impact of different counterexample processing strategies on a total number of sampled traces during learning

Counterexample Processing	35-State Gridworld	72-State Gridworld	MQTT	TCP	Bluetooth	Slot Machine
Angluin-style	39,718	139,358	30,761	54,514	41,237	564,368
Longest Prefix	77,974	254,087	26,493	33,436	24,499	629,069
RS	51,802	147,297	45,091	47,519	24,188	1,083,828

Table 5 Comparison between SMM and MDP learning costs on randomly generated MDPs and SMMs.

Random stochastic mealy machine							
		Number of states					
Input alphabet size	Output alphabet size	5	10	15	20	30	50
3	7	69.51%	83.28%	52.91	58.12%	79.63%	70.97%
4	10	86.97%	83.93%	86.78%	78.31%	76.34%	82.55%
7	15	93.38%	95.18%	94.35%	94.85%	96.18%	87.32%
Random Markov decision process							
		Number of States					
Input alphabet size	Output alphabet size	5	10	15	20	30	50
3	7	−11.76%	−19.58%	−22.6%	−59.59%	−−43.86%	−73.25%
4	10	18.06%	19.78%	37.88%	−29.27%	−17.11%	−20.54%
7	15	−37.27%	−44.41%	−65.86%	−45.11%	−19.78%	−0.09%

Cell values show the difference between total learning costs, expressed as a difference between the number of queries posed by SMM and MDP learning. Positive cell values indicate that SMM learning was more cost-efficient than MDP learning for a specific experiment, and vice versa

4.4 Comparison on randomly generated automata

To further examine the difference between the learning processes of MDP and SMM learning, we performed a set of experiments on randomly generated automata. Namely, we randomly generated both MDPs and SMMs and evaluated whether the type of the system under learning affects the learning process. Randomly generated automata were scaled along three dimensions: the number of states of the automata, input alphabet size, and output alphabet size. For each parameter configuration, we generated a random MDP and a random SMM.

Table 5 summarizes the comparison of MDP and SMM learning on randomly generated automata. The first result block compares the SMM and MDP learning of randomly generated SMMs. We observe that SMM learning consistently outperforms MDP learning if the system under learning has an SMM-like structure. For example, a cell value of *95.18%* (for randomly generated 10-state SMM with the input alphabet size of 7 and output alphabet size of 15) indicates that the SMM learning required *95.18%* fewer queries than the MDP learning of the same random SMM. More concretely, SMM learning required only 19k input–output traces, while MDP learning required 398k input–output traces, making SMM learning 20 times more efficient for this particular example. On the other hand, as seen in the lower half of Table 5, if the randomly generated automaton is an MDP, SMM learning required up to 73% more traces.

These findings indicate that the structure of the system under learning can significantly affect the learning process and required costs. Given that the L^*_{SMM} has no insight into the system-under-learning structure, a designer has to make a qualitative decision about whether to choose MDP or SMM learning. Based on our experience, we postulate that SMM learning might be preferable for most real-world systems due to SMM's more compact formalism.

4.5 Convergence

To evaluate learning speed, we disabled stopping and checked how long it takes for the learned hypothesis to reach a certain accuracy. For this purpose, we learned until creating a hypothesis with a model checking error of at most 2% for all properties defined for the respective example. Figure 8 summarizes the results of these experiments. Notice in Fig. 4 that the error stays close to zero after a certain number of rounds. Hence, these experiments serve to estimate how long it takes to converge to an accurate model.

All examples but *slot-machine* show noticeable improvements in both learning time and more importantly required traces when comparing MDP and SMM learning. The largest improvements between MDP and SMM learning are

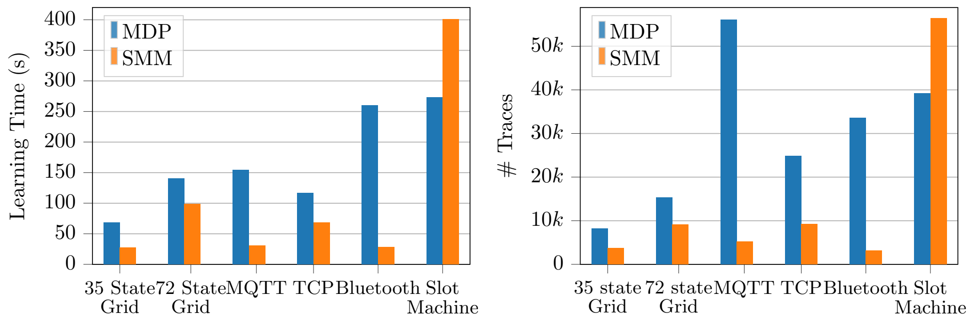

Fig. 8 Learning time in seconds (left) and number of traces (right) needed to reach at most 2% error for all properties

noticeable for *MQTT*, *TCP* and *Bluetooth* models. In those examples, models can be more compactly encoded as SMMs than as MDPs, which leads to a significant decrease of learning costs when compared to MDP learning. On the other hand, the structure of the *slot-machine* favors MDP learning. Learning a *slot-machine* MDP, required 25% fewer traces than learning an SMM. We argue that the good performance of MDP learning results from the fact that the minimal slot-machine MDP is only slightly larger than the minimal SMM and that many MDP states can be distinguished solely based on their output label without statistical tests. In general, when the output alphabet size is large, then MDP learning may be more efficient. In the extreme case, where every state is labeled with a unique output, the model structure is already given and learning amounts to estimating transition probabilities.

4.6 Comparison with related work

To put the improvements presented in this paper in context, we will compare the results found in this paper with two stochastic automata learning algorithms: L^*_{MDP} [12] and IOALERGIA [15, 29]. The algorithm presented in [12] serves as a basis for L^*_{SMM} , and we can directly compare results from overlapping benchmark models (First Grid, Second Grid, Slot Machine). For other benchmark models, we have obtained new results with AALPY, which implements both the original version of the L^*_{MDP} and IOALERGIA. It is important to note that by "original" L^*_{MDP} we consider the version of the algorithm presented in [12], without any of the optimizations presented in this paper, and when we were comparing SMM and MDP learning in the previous sections, we used the version of the algorithm presented in this manuscript.

Table 6 shows the results for the three learning algorithms on all benchmark models. Following the comparison method performed in [12], IOALERGIA was given the same number of random traces as L^*_{MDP} required throughout the learning. From the results shown in Table 6, we conclude that L^*_{SMM} requires significantly fewer queries than the original L^*_{MDP} , while maintaining high accuracy. Interestingly, in a few examples, it even has marginally better accuracy than L^*_{MDP} , but with almost an order of magnitude fewer traces. However, it is important to note that the accuracy differences between both algorithms are quite small, in some cases negligible. Compared to both algorithms, IOALERGIA has worse accuracy, even with a high number of provided traces.

The difference in the number of traces required by L^*_{SMM} and original L^*_{MDP} to learn accurate models can be attributed to several optimizations presented in this paper. Firstly, we introduced an additional stopping criterion in Sect. 3.4.4, which stops learning when we observe that additional sampling does not improve the ambiguity of the observation table. Secondly, we have replaced the completeness query and the frequency query with the tree query (Sect. 3.4.3), which makes the algorithm automatically select the number of queries it asks during each learning round. Finally, counterexample processing techniques and our strategy to invest testing resources in finding the shortest counterexamples make the algorithm more efficient, as outlined in Sect. 3.4.2.

5 Related work

In the following, we discuss passive and active approaches to learn stochastic and non-deterministic models.

5.1 Passive learning of stochastic systems

Active learning samples data by actively querying the system under learning, whereas passive learning assumes a preexisting sample of system traces. Two early, notable approaches for identifying stochastic regular languages from traces are ALERGIA [19] and `rlips` [35].

Originally, ALERGIA was used to construct Markov chains that approximate the patterns found in the provided data-set. ALERGIA has been extended to MDPs by Mao et al.[15, 29] and dubbed IOALERGIA. Like IOALERGIA, we compute transition probabilities based on observed output frequencies. Another similarity of L^*_{SMM} and IOALERGIA is the usage of IOFPT and the usage of statistical tests to help differentiate

Table 6 Comparison of L^*_{SMM} with original L^*_{MDP} and IoALERGIA

Algorithm	L^*_{SMM}		Classic L^*_{MDP}		IoALERGIA	
	# Traces	Mean Error	# Traces	Mean Error	# Traces	Mean Error
35-State Grid	44,954	0.017	391, 530	0.0113	391, 530	0.667
72-State Grid	113,690	0.04	515,950	0.069	515,950	0.163
MQTT	39,904	0.015	300,500	0.015	300,500	0.018
TCP	54,514	0.005	279,423	0.004	279,423	0.19
Bluetooth	35,605	0.011	317,446	0.016	317,446	0.10
Slot Machine	622,285	0.011	1,567,487	0.015	1,567,487	0.35

states. Active extensions of IoALERGIA can be found in [36] and [28], where the latter targets learning-based verification w.r.t reachability objectives.

Casacuberta and Vidal proposed the `GIATI` algorithm [37]. `GIATI` is given a training corpus of source-target pairs of sentences algorithm from which, using statistical alignment methods, it produces a set of conventional strings from which a stochastic rational grammar (e.g., an n-gram) is inferred. Such a grammar is then translated to a finite-state transducer.

5.2 Active learning of stochastic systems

Our work builds upon [12], an active approach to learning of MDPs. As discussed in the previous sections, we substantially improve the approach for learning MDPs itself and adapt it to learning SMMs. Our approach shares similarities with active learning of observable non-deterministic finite-state machines (ONFSMs) [38]. The observation table in [38] is similar to the one used in L^*_{SMM}, in a way that cells of the observation table contain all outputs that are observed once an input–output sequence is executed. However, while [38] requires all possible outputs to be observed after executing a query ("all-weather assumption") to build observation tables, we do not make this assumption by relying on statistical tests. In the test-based ONFSM learning, implemented in [13], all-weather assumption is relaxed and all cells in the observation table are populated by executing desired input–output sequences. Test-based learning of ONSFM requires significantly more queries, as longer traces might have a small probability of being reached. L^*_{SMM} circumvents this limitation by differentiating states not only on their future behavior, found in the E-set of the observation table, but also by performing statistical tests.

Another L^*-based learning approach for ONFSMs has been proposed by Pferscher and Aichernig [39], which is specifically well-suited to learning non-deterministic behavior resulting from abstraction. Their approach has successfully been applied to efficiently learn abstract model capturing the interaction between several deterministic MQTT clients and a broker, while greatly reducing total system interaction time during learning compared to the standard deterministic learning.

Volpato and Tretmans presented L^*-based approach for learning of non-deterministic input–output transition systems [40]. Their approach inspired the addition of the q_{undef} state, that indicates the need for further sampling in the presence of low/incomplete information.

Bacci et al. [41] presented another interesting approach that combines passive and active learning of MDPs. Unlike previous active approaches, it is not L^*-based. It adapts the Baum-Welch algorithm so that is able to learn MDPs from a set of observation. Such passive approach learns more accurate models than IoALERGIA, albeit with higher runtime complexity. Authors also examined the active variation, which introduces model-based sampling strategies based on an intermediate hypothesis. An activated version of their algorithm learns less accurate MDPs than L^*_{SMM}, but with fewer traces. In the future work, we will consider a combination of both approaches, that is a combination of L^*_{SMM} and adapted Baum-Welch algorithm, with the aim of achieving high accuracy of the learned model with even fewer samples.

6 Conclusion

We presented L^*_{SMM}, an L^*-based algorithm for active learning of models of stochastic reactive systems. By improving previous work [12] and adapting it from learning MDPs to SMMs, we learn models more efficiently while achieving accurate results. The experimental evaluation of our implementation available in AALPY [13] shows a significant reduction in the number of required system interactions. Since interactions with the system are typically the time-consuming aspect of applications of automata learning, this number is the most important efficiency metric. In particular, we reduced the required number of system traces, i.e., sequences of interactions, by up to 8.7 times, as compared to MDP learning. Through these improvements, we hope to enable industrial applications of stochastic active automata learning.

In future work, we plan to combine the stochastic L^* algorithm with learning-based verification techniques, such as [28], and apply it in case studies with stochastic reactive systems, such as communication protocols over a lossy channel. Another promising direction for future research is the adaptation and application of stochastic learning algorithms to non-deterministic/partially observable MDPs. Based on our current research efforts, we postulate that L^*_{SMM} can learn a deterministic approximation of such automata with high accuracy, but at high costs. We continue the study of the necessary adaptations and heuristics that enable efficient learning of non-deterministic MDPs.

Acknowledgements This work has been supported by the "University SAL Labs" initiative of Silicon Austria Labs (SAL) and its Austrian partner universities for applied fundamental research for electronic based systems.

Funding Open access funding provided by Graz University of Technology.

Appendix: Learned models

In this appendix, we show two learned models for the *first grid* example, one MDP model and one SMM model to illustrate their differences. Even though they have only 19 and 35 states, respectively, they are hard to visualize. Therefore, we show only the initial part of both models in Fig. 9. It can be seen that there are several self-loop transitions in the initial SMM state. They produce the *Wall* output, for instance, signaling that the agent navigating in the Gridworld cannot move *West*, but bumps into a wall when trying to move west. Modeling the same functionality in the MDP requires two states, where one is labeled *Wall*. These two states are the topmost states in the left part of Fig. 9. As another example, we can take a look at the transition labeled with the input *East* and the output *concrete* from the initial SMM state to the

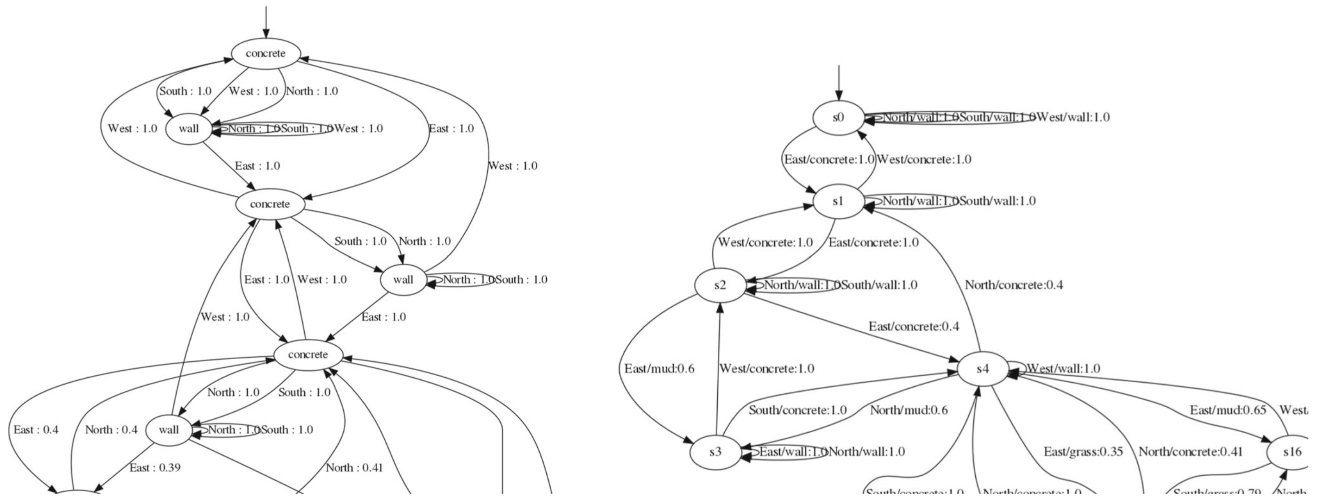

Fig. 9 Initial part of the learned MDP model of the first Gridworld example (left) and the corresponding initial part of the learned SMM model (right)

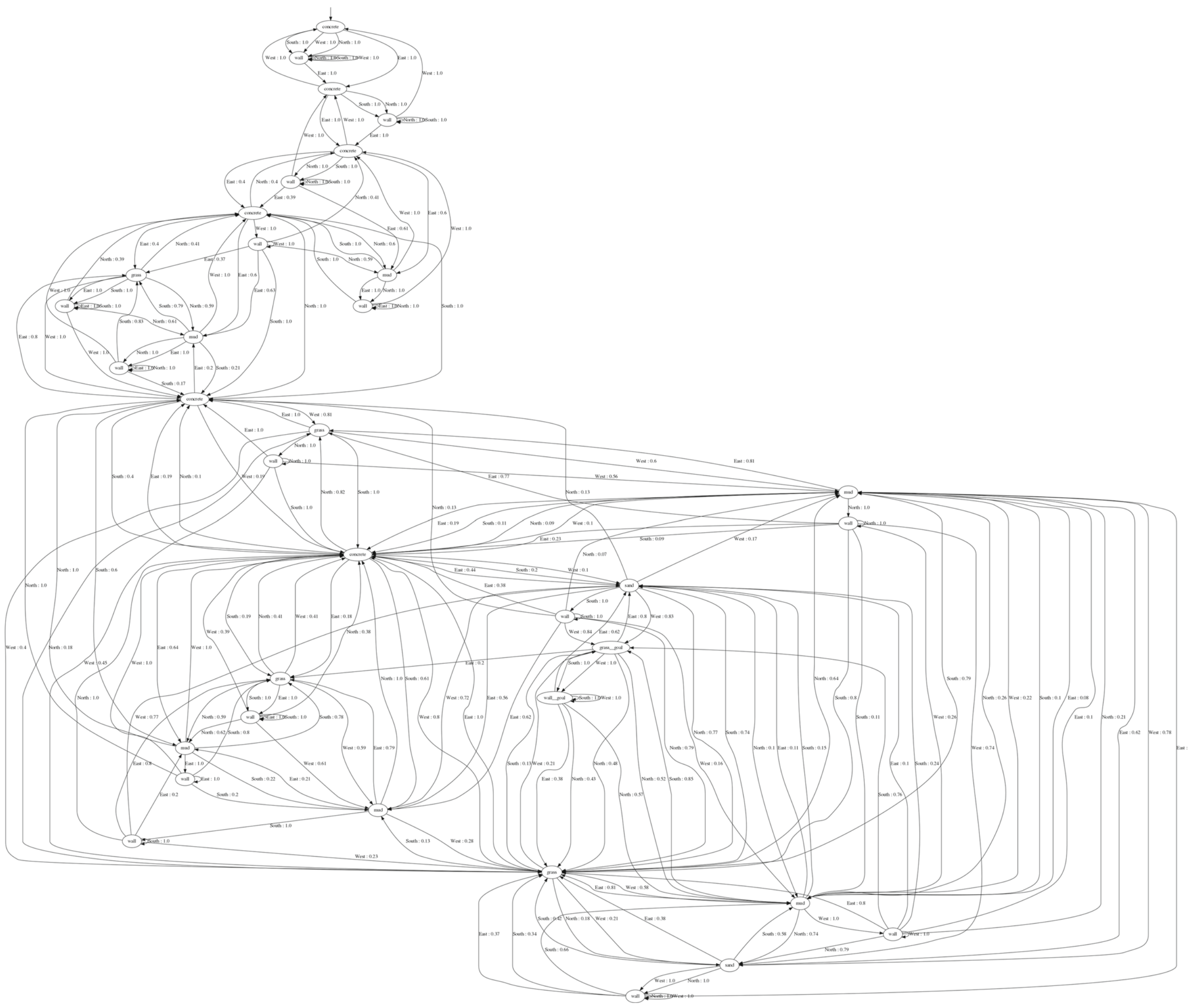

Fig. 10 Learned MDP model of the first Gridworld example

state *s1*. The transition model that by moving *East*, the agent moves onto concrete. The same transition appears twice in the MDP model; once from the initial state and once from the state that is below and left of the initial state. This explains that more samples are required to learn MDP transitions and their probabilities. In this particular case, the probabilities are actually simple to learn, as they equal to one, but in the lower parts of the figure, there are other probability values.

The complete models are shown in Figs. 10 and 11, respectively. It is apparent that the SMM model is considerably smaller and because of that also easier to analyze manually.

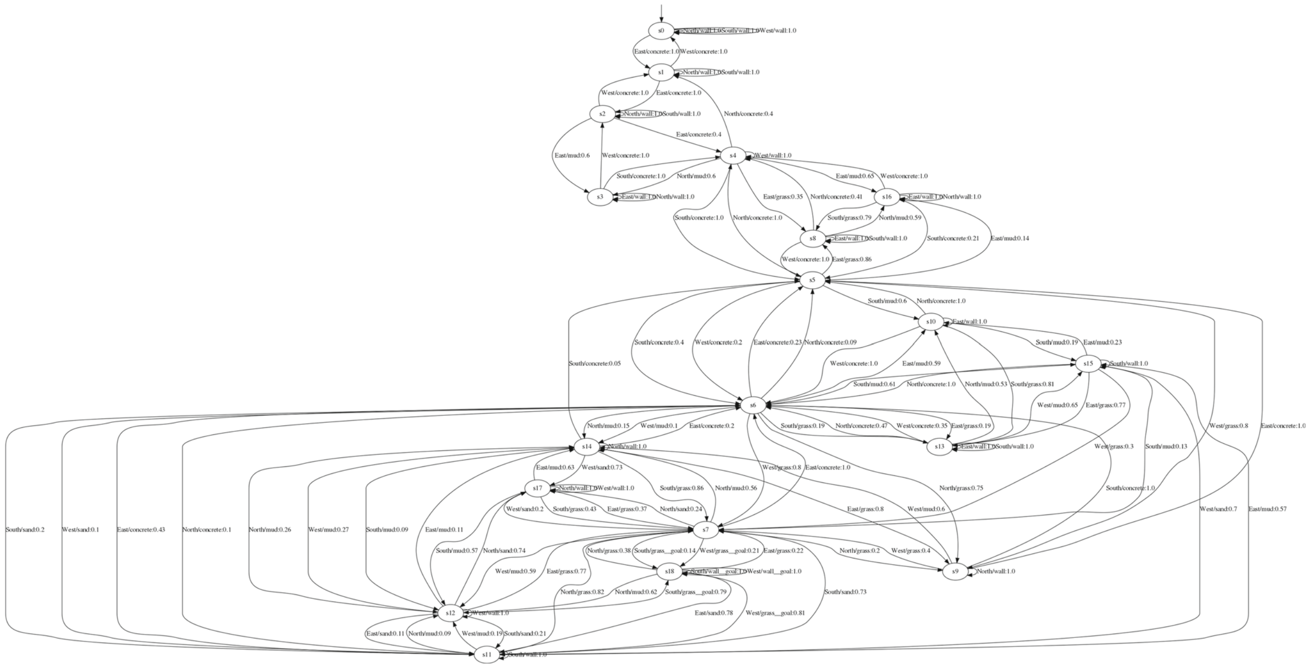

Fig. 11 Learned SMM model of the first Gridworld example

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987)
2. Rivest, R.L., Schapire, R.E.: Inference of finite automata using homing sequences. Inf. Comput. **103**(2), 299–347 (1993)
3. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: a redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds), Proceedings of the 5th International Conference on Runtime Verification, RV 2014. Lecture Notes in Computer Science, Toronto, ON, Canada, September 22-25, (2014), vol. 8734, pp. 307–322 (2014)
4. Hungar, H., Niese, O., Steffen, B.: Domain-specific optimization in automata learning. In: Proceedings of the 15th International Conference on Computer Aided Verification, CAV 2003. LNCS, Boulder, CO, USA, July 8-12, 2003, vol. 2725, pp. 315–327 (2003)
5. Margaria, T., Niese, O., Raffelt, H., Steffen, B.: Efficient test-based model generation for legacy reactive systems. In: 9th IEEE International High-Level Design Validation and Test Workshop 2004, Sonoma Valley, CA, USA, November 10-12, 2004, pp. 95–100 (2004)
6. Shahbaz, M., Groz, R.: Inferring Mealy machines. In: Proceedings of the 2nd World Congress on FM 2009: Formal Methods. LNCS, Eindhoven, The Netherlands, November 2-6, 2009, vol. 5850, pp. 207–222 (2009)
7. Aichernig, B.K., Pferscher, A., Tappler, M.: From passive to active: Learning timed automata efficiently. In: Proceedings of the 12th International Symposium on NASA Formal Methods, NFM 2020. LNCS, Moffett Field, CA, USA, May 11-15, 2020, vol. 12229, pp. 1–19 (2020)
8. Tappler, M., Aichernig, B.K., Lorber, F.: Timed automata learning via SMT solving. In: Deshmukh, J.V., Havelund, K., Perez, I. (eds), Proceedings of the 14th International Symposium on NASA Formal Methods, NFM 2022. Lecture Notes in Computer Science, Pasadena, CA, USA, May 24-27, 2022, vol. 13260, pp. 489–507 (2022)
9. Aarts, F., Heidarian, F., Kuppens, H., Olsen, P., Vaandrager, F.W.: Automata learning through counterexample guided abstraction refinement. In: Giannakopoulou, D., Méry, D. (eds), Proceedings of the 18th International Symposium on FM 2012: Formal Methods. Lecture Notes in Computer Science, Paris, France, August 27-31, 2012, vol. 7436, pp. 10–27 (2012)
10. Tappler, M., Aichernig, B.K., Bloem, R.: 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, pp. 276–287 (2017)
11. Tappler, M., Aichernig, B.K., Bacci, G., Eichlseder, M., Larsen, K.G.: L^*-based learning of Markov decision processes. In: Formal Methods—The Next 30 Years—Third World Congress, FM 2019, Porto, Portugal, October 7-11, 2019, Proceedings, volume 11800 of LNCS, pp. 651–669 (2019)
12. Tappler, M., Aichernig, B.K., Bacci, G., Eichlseder, M., Larsen, K.G.: L*-based learning of Markov decision processes (extended version). Form. Asp. Comput. **33**(4–5), 575–615 (2021)
13. Muškardin, E., Aichernig, B., Pill, I., Pferscher, A., Tappler, M.: AALpy: an active automata learning library. Innov. Syst. Softw. Eng., pp. 1–10, 03 (2022). The implementation of AALpy is available online at https://github.com/DES-Lab/AALpy
14. Tappler, M., Muskardin, E., Aichernig, B.K., Pill, I.: Active model learning of stochastic reactive systems. In: Calinescu, R., Pasareanu, C.S. (eds), Proceedings of the 19th International Conference on Software Engineering and Formal Methods, SEFM 2021. Lecture Notes in Computer Science, Virtual Event, December 6-10, 2021, vol. 13085, pp. 481–500 (2021)
15. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning deterministic probabilistic automata from a model checking perspective. Mach. Learn. **105**(2), 255–299 (2016)
16. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Proceedings of the 23rd International Conference on Computer Aided Verification, CAV 2011. LNCS, Snowbird, UT, USA, July 14-20, 2011, vol. 6806, pp. 585–591 (2011)

17. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A storm is coming: a modern probabilistic model checker. In: Proceedings of the 29th International Conference on Computer Aided Verification (CAV 2017): Part II.LNCS, Heidelberg, Germany, July 24-28, 2017, vol. 10427, pp. 592–600 (2017)
18. de la Higuera, C.: Grammatical Inference: Learning Automata and Grammars. Cambridge University Press, New York (2010)
19. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: Carrasco, R.C., Oncina, J. (eds), Proceedings of the 2nd International Colloquium on Grammatical Inference and Applications (ICGI-94). Lecture Notes in Computer Science, Alicante, Spain, September 21-23, 1994, vol. 862, pp. 139–152 (1994)
20. El-Fakih, K., Groz, R., Irfan, M., Shahbaz, M.: Learning finite state models of observable nondeterministic systems in a testing context. 01 (2010)
21. Grinchtein, O., Jonsson, B., Leucker, M.: Learning of event-recording automata. Theor. Comput. Sci. **411**(47), 4029–4054 (2010)
22. Nerode, A.: Linear automaton transformations. Proc. Am. Math. Soc. **9**(4), 541–544 (1958)
23. Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301), 13–30 (1963)
24. Karl Pearson, F.R.S.: X on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. Lond. Edinb. Dublin Philos. Mag. J. Sci. **50**(302), 157–175 (1900)
25. Yates, F.: Contingency tables involving small numbers and the $\chi - 2$ test. Suppl. J. R. Stat. Soc. **1**(2), 217–235 (1934)
26. Aichernig, B.K., Tappler, M., Wallner, F.: Benchmarking combinations of learning and testing algorithms for active automata learning. In: Proceedings of the 14th International Conference on Tests and Proofs, TAP@STAF 2020. LNCS, Bergen, Norway, June 22-23, 2020, vol. 12165, pp. 3–22 (2020)
27. Howar, F., Steffen, B.: Active automata learning as black-box search and lazy partition refinement. In: Jansen, N., Stoelinga, M., van den Bos, P. (eds), A Journey from Process Algebra via Timed Automata to Model Learning—Essays Dedicated to Frits Vaandrager on the Occasion of His 60th Birthday. Lecture Notes in Computer Science, vol. 13560 , pp. 321–338. Springer (2022)
28. Aichernig, B.K., Tappler, M.: Probabilistic black-box reachability checking (extended version). Formal Methods in System Design (2019)
29. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning Markov decision processes for model checking. In: Proceedings Quantities in Formal Methods, QFM 2012. EPTCS, Paris, France, 28 August 2012, vol. 103 , pp. 49–63 (2012)
30. Fiterau-Brostean, P., Janssen, R., Vaandrager, F.W.: Combining model learning and model checking to analyze TCP implementations. In: Proceedings of the 28th International Conference on Computer Aided Verification (CAV 2016): Part II. LNCS, Toronto, ON, Canada, July 17-23, 2016, vol 9780, pp. 454–471 (2016)
31. Pferscher, A., Aichernig, B.K.: Fingerprinting Bluetooth low energy devices via active automata learning. In: Huisman, M., Pasareanu, C.S., Zhan, N. (eds), Proceedings of the 24th International Symposium on Formal Methods, FM 2021, Virtual Event. Lecture Notes in Computer Science, November 20-26, 2021, vol. 13047, pp. 524–542 (2021)
32. Larsen, K.G., Legay, A.: Statistical model checking: past, present, and future. In: Margaria, T., Steffen, B. (eds), Proceedings of the 7th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques (ISoLA 2016): Part I. Lecture Notes in Computer Science, Imperial, Corfu, Greece, October 10-14, 2016, vol. 9952, pp. 3–15 (2016)
33. Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. Ann. Inst. Stat. Math. **10**(1), 29–35 (1959)
34. STUDENT: The probable error of a mean. Biometrika **6**(1), 1–25 (1908)
35. Carrasco, R.C., Oncina, J.: Learning deterministic regular grammars from stochastic samples in polynomial time. RAIRO Theor. Inform. Appl. (RAIRO: ITA) **33**(1), 1–20 (1999)
36. Chen, Y., Nielsen, T.D.: 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. Vol. 2, pp. 289–294 (2012)
37. Casacuberta, F., Vidal, E.: Machine translation with inferred stochastic finite-state transducers. Comput. Linguist. **30**(2), 205–225 (2004)
38. El-Fakih, K., Groz, R., Irfan, M.N., Shahbaz, M.: Learning finite state models of observable nondeterministic systems in a testing context. In: ICTSS 2010, pp. 97–102 (2010)
39. Pferscher, A., Aichernig, B.K.: Learning abstracted nondeterministic finite state machines. In: Casola, V., De Benedictis, A., Rak, M. (eds), Proceedings of the 32nd IFIP WG 6.1 International Conference on Testing Software and Systems, ICTSS 2020, Naples, Italy, December 9-11, 2020, volume 12543 of LNCS, pp. 52–69 (2020)
40. Volpato, M., Tretmans, J.: Approximate active learning of nondeterministic input output transition systems. ECEASST (2015). https://doi.org/10.14279/tuj.eceasst.72.1008
41. Bacci, G., Ingólfsdóttir, A., Larsen, K.G., Reynouard, R.: Active learning of Markov Decision Processes using Baum–Welch algorithm. In: Arif Wani, M., Sethi, I.K., Shi, W., Qu, G., Raicu, D.S., Jin, R., (eds), 20th IEEE International Conference on Machine Learning and Applications, ICMLA 2021, Pasadena, CA, USA, December 13-16, 2021, pp. 1203–1208 (2021)

Edi Muškardin is a PhD student at Dependable Embedded Systems Lab (DES Lab), a collaboration between Silicon Austria Labs and Graz University of Technology. His work focuses on the development of novel automata learning methods and their applications in various domains. More concretely, he explores the applicability of automata learning within the field of machine learning. His primary focus is on addressing the question: "In what areas can automata learning contribute to machine learning, and conversely, where can machine learning enhance automata learning?". He is a principal developer and maintainer of AALpy, an automata learning library.

Martin Tappler is a postdoctoral researcher at the Institute of Software Technology at Graz University of Technology. He obtained a PhD in computer science in 2019 at Graz University of Technology on the topic of automata learning. Subsequently, he worked as a postdoctoral researcher at the Schaffhausen Institute of Technology, Switzerland. His research interests include model-based testing, automata learning, and verification of learning-enabled systems.

Bernhard K. Aichernig is a tenured associate professor (ao. Univ.-Prof.) at Graz University of Technology, Austria. With his research group he investigates the foundations of software engineering for realizing dependable computer-based systems. Bernhard is an expert in formal methods and testing. His research covers a variety of areas combining falsification, verification, and abstraction techniques. Current topics include the Internet of Things, test-case generation, automata learning, and statistical model checking. Since 2020, he leads the TU Graz - SAL Dependable Embedded Systems Lab (DES Lab) with fundamental research in zero-bug software and dependable AI. He participated in five European projects. From 2004-2016 Bernhard served as a board member of Formal Methods Europe, the association that organizes the Formal Methods symposia. From 2002 to 2006 he had a faculty position at the United Nations University in Macao S.A.R., China. Bernhard holds a habilitation in Practical Computer Science and Formal Methods, a doctorate, and a diploma engineer degree from Graz University of Technology.

Ingo Pill is a staff scientist with the Silicon Austria Labs (SAL) in Graz. Before joining SAL in 2020, he was a senior scientist with the Software Engineering & Artificial Intelligence group at the Institute for Software Technology, TU Graz where he completed his PhD in Formal Verification in 2008. His research interests lie in symbolic AI and formal methods, with a special focus on diagnosis and its integration with verification and control techniques in order to facilitate resilient systems. Ingo is a recognized expert in model-based diagnosis and reasoning, is a member of AAAI, ACM and ASAI/GAI, reviews for top journals and conferences, and has been a regular (senior) program committee member of leading AI venues like AAAI, IJCAI or ECAI where he received several awards for his efforts. He has been active in organizing workshops and conferences, e.g., as co-chair of the International Workshop on Principles of Diagnosis (DX) (2014, 2017, 2023) for which he also chairs the recently founded steering committee, as co-chair of the Dagstuhl Seminar 24031 “Fusing Causality, Reasoning and Learning for Fault Management and Diagnosis) in 2024, and for further conferences and workshops in the area of AI. He co-supervised several master and PhD students and has been chair or member of MSc/PhD examination boards (e.g, TU Graz, ANU or University of Trento). Ingo is one of three members of the SAL doctoral college core team who co-authored the multi-million MSCA-Cofund program CRYSTALLINE (funding of 2.4M”) to start in 2024.