Mach Learn
DOI 10.1007/s10994-016-5565-9

Learning deterministic probabilistic automata from a model checking perspective

Hua Mao[1] · **Yingke Chen**[1] · **Manfred Jaeger**[2] · **Thomas D. Nielsen**[2] · **Kim G. Larsen**[2] · **Brian Nielsen**[2]

Received: 1 December 2012 / Accepted: 7 April 2016

Abstract Probabilistic automata models play an important role in the formal design and analysis of hard- and software systems. In this area of applications, one is often interested in formal model-checking procedures for verifying critical system properties. Since adequate system models are often difficult to design manually, we are interested in learning models from observed system behaviors. To this end we adopt techniques for learning finite probabilistic automata, notably the ALERGIA algorithm. In this paper we show how to extend the basic algorithm to also learn automata models for both reactive and timed systems. A key question of our investigation is to what extent one can expect a learned model to be a good approximation for the kind of probabilistic properties one wants to verify by model checking. We establish theoretical convergence properties for the learning algorithm as well as for probability estimates of system properties expressed in linear time temporal logic and linear continuous stochastic logic. We empirically compare the learning algorithm with statistical model checking and demonstrate the feasibility of the approach for practical system verification.

Keywords Probabilistic model checking · Probabilistic automata learning · Linear time temporal logic

1 Introduction

Grammatical inference (GI) (Higuera 2010), also known as grammar induction or grammar learning, is concerned with learning language specifications in the form of grammars or automata from data consisting of strings over some alphabet. Starting with Angluin's seminal work (Angluin 1987), methods have been developed for learning deterministic, non-

Editors: Jeffrey Heinz, C. de la Higuera and Tim Oates.

✉ Manfred Jaeger
jaeger@cs.aau.dk

[1] College of Computer Science, Sichuan University, Chengdu 610065, China

[2] Department of Computer Science, Aalborg University, 9220 Aalborg East, Denmark

Published online: 18 May 2016

deterministic and probabilistic grammars and automata. The learning techniques in GI have been applied in many areas, such as speech recognition, software development, pattern recognition, and computational biology. In this paper we adapt the learning techniques in the GI area to learn models for model checking.

Model Checking is a verification technique for determining whether a system model complies with a specification provided in a formal language (Baier and Katoen 2008). In the simplest case, system models are given by finite non-deterministic or probabilistic automata, but model-checking techniques have also been developed for more sophisticated system models, e.g., timed automata (Laroussinie et al. 1995; Bouyer et al. 2011, 2008). Powerful software tools that are available for model checking include UPPAAL (Behrmann et al. 2011) and PRISM (Kwiatkowska et al. 2011).

Traditionally, models used in model-checking are manually constructed, either in the development phase as system designs, or for existing hard- or software systems from known specifications and documentation. This procedure can be both time-consuming and error-prone, especially for systems lacking updated and detailed documentation, such as legacy software, 3rd party components, and black-box systems. These difficulties are generally considered a hindrance for adopting otherwise powerful model checking techniques, and have led to an increased interest in methods for data-driven *model learning* (or *specification mining*) for formal verification (Ammons et al. 2002; Sen et al. 2004a; Mao et al. 2011, 2012).

In this paper we investigate methods for learning deterministic probabilistic finite automata (DPFA) from data consisting of previously observed system behaviors, i.e., sample executions. The probabilistic models considered in this paper include labeled Markov decision processes (MDPs) and continuous-time labeled Markov chains (CTMCs), where the former model class also covers labeled Markov chains (LMCs) as a special case. Labeled Markov decision processes can be used to model reactive systems, where input actions are chosen non-deterministically and the resulting output for a given input action is determined probabilistically. Nondeterminism can model the free and unpredictable choices from an environment or the concurrency between components in a system. MDPs and by extension LMCs are discrete-time models, where each transition takes a universal discrete time unit. CTMCs, on the other hand, are real-time models, where the time delays between transitions are determined probabilistically. We show how methods for learning deterministic probabilistic finite automata (DPFA) (Carrasco and Oncina 1994, 1999; Higuera 2010) can be adapted for learning the above three model classes and pose the results within a model checking context. We give consistency results for the learning algorithms, and we analyze both theoretically and experimentally how the convergence of the learned models relates to the convergence of system properties expressed in linear time logics.

We also compare the accuracy of model checking learned models with the accuracy of a statistical model checking approach, where probabilities of query properties are directly estimated from the empirical frequencies in the data. Our results here demonstrate a smoothing effect of model learning which can prevent overfitting, but may in some cases also lead to less accurate results compared to statistical model checking. Our results also indicate a significant advantage of model learning over statistical model checking for the amortized time complexity over multiple queries.

1.1 Related work

Work on learning finite automata models can first be divided into two broad categories: active learning following Angluin's $\mathbf{L}^*$ algorithm (Angluin 1987), and passive learning based on a state-merging procedure.

Active learning is based on the assumption that there exists a teacher or an oracle that answers *membership* and *equivalence* queries. Originally developed by Angluin (1987) for learning deterministic finite automata, $\mathbf{L}^*$ has been generalized in many different ways that also include extensions to learning automata models with inputs and outputs, as well as probabilistic automata: in Bollig et al. (2010), $\mathbf{L}^*$ is exploited to learn communicating finite-state machines by using a given set of positive and negative message sequence charts to answer the membership and equivalence queries. In Niese (2003), $\mathbf{L}^*$ is adapted to learn deterministic Mealy machines. This work is further extended to learn deterministic I/O automata by placing a transducer between the teacher and the Mealy machine learner (Aarts and Vaandrager 2010). In Grinchtein et al. (2005, 2006), $\mathbf{L}^*$ is adapted to learn *deterministic event-recording automata* which is a subclass of real-time automata.

To learn probabilistic automata models, modified versions of $\mathbf{L}^*$ have been proposed in which a membership query now asks for the probability of a given word in the target model (Tzeng 1992; de Higuera and Oncina 2004; Feng et al. 2011). In Komuravelli et al. (2012), $\mathbf{L}^*$ combined with a stochastic state-space partitioning algorithm makes it possible to learn nondeterministic labeled probabilistic transition systems from tree samples. Exact oracles for (classical or probabilistic) membership and equivalence queries are usually not available in practice and have to be approximated. For deterministic finite automata this has been implemented using a conformance testing sub-routine (Raffelt and Steffen 2006).

Passive learning methods that only require data consisting of observed system behaviors have been developed for probabilistic automata models (Carrasco and Oncina 1994; Ron et al. 1996). These approaches are based on iteratively merging candidate states. Different approaches differ with respect to the strategy according to which candidate states are generated, and the criteria used for deciding whether to merge states. In algorithms following the paradigm of the ALERGIA algorithm (Carrasco and Oncina 1994), first a maximal, tree-shaped automaton is constructed, and iteratively reduced by recursive merge operations. The learning paradigm introduced by Ron et al. (1996), on the other hand, starts with a minimal automaton and successively refines it by expanding existing states with new candidate states. More important than these architectural differences, however, are differences in the criteria used for state merging. The most common approach is to use a statistical test for the equivalence of the distributions defined at the nodes (Carrasco and Oncina 1994; de la Higuera and Thollard 2000). For basic probabilistic automata only tests for the equivalence of binomial distributions are required, for which the use of the Hoeffding test is usually suggested. For timed automata models, this has been extended in Sen et al. (2004a) to also test the equivalence of two exponential distributions defining the delay times at the states. Thollard et al. (2000) provide the minimum divergence inference algorithm to control state merging: two nodes should be merged if the loss of the likelihood can be compensated by the reduced complexity of the resulting model. Ron et al. (1998) base the state merging decision on the existence of a distinguishing string, i.e. a string for which the difference of probability at the two candidate states exceeds a certain threshold. The state merging algorithms have been extended to learn stochastic transducers (Oncina et al. 1993) and timed automata (Verwer 2010).

In a number of papers the convergence properties of learning algorithms have been studied. Carrasco and Oncina (1994), de la Higuera and Thollard (2000) and Sen et al. (2004a) give learning in the limit results, i.e., the unknown automaton is correctly identified in the limit of large sample sizes. Quantitative bounds on the speed of convergence in the form of PAC learnability results are given in Ron et al. (1996), Clark and Thollard (2004) and Castro and Gavaldà (2008).

The use of grammatical inference techniques for model construction in a verification context has been proposed in several papers (Cobleigh et al. 2003; Giannakopoulou and

Păsăreanu 2005; Leucker 2007; Singh et al. 2010; Feng et al. 2011). These papers focus on active learning using variants of $\mathbf{L}^*$, and only Feng et al. (2011) consider the probabilistic case.

Statistical model checking (SMC) (Sen et al. 2004b; Legay et al. 2010) or approximate model-checking (Hérault et al. 2004) has a similar objective as model learning for verification. Instead of constructing a model from sample executions, one directly checks the empirical probabilities of properties in the data. Since the sample executions can only be finite strings, this approach is limited with respect to checking probabilities for unbounded properties.

1.2 Contribution and outline

Our work follows the ALERGIA paradigm and is closely linked to previous work (Carrasco and Oncina 1994; Sen et al. 2004a). We here do not introduce any major algorithmic novelties, but give an integrated account of learning system models that can also represent input/output behaviors and time delays. The novel aspect of this paper is a theoretical and experimental analysis of the feasibility of using the learned model for formal verification of temporal logic properties. We present theoretical results that based on the convergence properties for ALERGIA-like algorithms establish the convergence also of probability estimates for system properties of interest. An extensive empirical evaluation provides insight into the workings of the algorithm and demonstrates the feasibility of the learning approach for verification applications in practice. The evaluation also includes a detailed comparison of the learning approach with statistical model checking, considering both accuracy results and the time and space complexity for performing model checking. Finally, we provide a new detailed proof of the fundamental convergence results. While generally following the lines of argument pioneered in Carrasco and Oncina (1994), de la Higuera and Thollard (2000) and Sen et al. (2004a), our new proof contains the following improvements: it is cast in a very general framework, and accommodates in a uniform manner different classes of automata models, including input/output and timed automata. It is presented in a modular form that clearly identifies separate conditions for the algorithmic structure of the state merging procedure, for the statistical tests used for state-merging decisons, and for the data-generating process. The structure of the proof thereby facilitates the application of the convergence result to new learning scenarios. Since this general convergence analysis is somewhat independent from the rest of this paper, it is placed in a self-contained "Appendix".

The paper is structured as follows: Sect. 2 presents background material. Section 3 describes the adapted ALERGIA algorithm for learning system models, and Sect. 4 analyzes the consistency and convergence properties of the learning algorithm. Section 5 provides empirical results on the behavior of the learning algorithm and demonstrates the use of the algorithm in a model checking context. The last section concludes the paper and outlines directions for future research. The "Appendix" contains our general convergence analysis. This paper is an extended version of Mao et al. (2011, 2012). Compared to these earlier conference publications, this paper significantly expands the theoretical analysis of the consistency aspects. It also includes a much more comprehensive experimental evaluation, in which the comparison against statistical model checking is added as a new dimension.

2 Preliminaries

2.1 Strings

We start by introducing the notion of strings that will be used throughout the paper.

- Given a finite alphabet Σ, we use Σ^* and Σ^ω to denote the set of all finite and infinite strings over Σ, respectively.
- Given a infinite string $s = \sigma_0\sigma_1 \ldots \in \Sigma^\omega$ starting with the symbol σ_0, $s[j \ldots] = \sigma_j\sigma_{j+1}\sigma_{j+2} \ldots$ is the suffix of s starting with the $(j+1)$st symbol σ_j and $\sigma_0\sigma_1 \ldots \sigma_j \in \Sigma^*$ is the prefix of s.
- Given an input alphabet Σ^{in} and an output alphabet Σ^{out}, an infinite I/O string is denoted as $\pi = \sigma_0\alpha_1\sigma_1 \ldots \in \Sigma^{\text{out}} \times (\Sigma^{\text{in}} \times \Sigma^{\text{out}})^\omega$, and $\sigma_0\alpha_1\sigma_1 \ldots \alpha_n\sigma_n \in \Sigma^{\text{out}} \times (\Sigma^{\text{in}} \times \Sigma^{\text{out}})^*$ is the prefix of s with $2n+1$ alternating I/O symbols.
- Given a finite string $s = \sigma_0\sigma_1 \ldots \sigma_n$, we use $\text{prefix}(s) = \{\sigma_0 \ldots \sigma_j | 0 \leq j \leq n\}$ to denote the set of all prefixes of string s. For a finite I/O string $\pi = \sigma_0\alpha_1\sigma_1 \ldots \alpha_n\sigma_n$, $\text{prefix}(\pi) = \{\sigma_0\alpha_1\sigma_1 \ldots \alpha_j\sigma_j | 0 \leq j \leq n\}$. Given a set of finite strings S, $\text{prefix}(S)$ denotes all prefixes of strings in S.
- A timed string $\rho = \sigma_0 t_0 \sigma_1 t_1 \ldots$ includes the time delay $t_i \in \mathbb{R}_{>0}$ between the observation of two consecutive symbols σ_i and σ_{i+1} in the string. Given a timed string ρ, $\rho[n] = \sigma_n$ is the $(n+1)$th symbol of ρ, $\rho[n \ldots] = \sigma_n t_n \sigma_{n+1} t_{n+1} \ldots$ is the suffix starting from the $(n+1)$th symbol, $\rho\langle n \rangle = t_n$ is the time spent between observing the symbols σ_n and σ_{n+1}, and $\rho@t$ is the suffix starting at time $t \in \mathbb{R}_{>0}$, i.e., $\rho@t = \rho[n \ldots]$, where n is the smallest index such that $\sum_{i=0}^{n} \rho\langle i \rangle \geq t$. The skeleton of ρ, denoted $\mathbb{S}(\rho)$, is the string $\sigma_0\sigma_1 \ldots \in \Sigma^\omega$.

2.2 Stochastic system models

We begin with the definition of the basic (D)MC model, which quantifies transitions with probabilities. We next extend (D)MCs to DMDPs by introducing input actions, where each input action on a state defines a probability distribution over successor states. In both DMCs and DMDPs, the time spent in each state is given by a universal discrete time unit. We lift this assumption in DCTMCs by modeling the transition times using a probabilistic model.

Definition 1 (MC) A *labeled Markov chain (MC)* is a tuple $\mathcal{M}^c = \langle Q, \Sigma^{\text{out}}, \mathbb{I}, \delta, L \rangle$, where

- Q is a finite set of states,
- Σ^{out} is a finite alphabet,
- $\mathbb{I} : Q \to [0, 1]$ is an initial probability distribution over Q such that $\sum_{q \in Q} \mathbb{I}(q) = 1$,
- $\delta : Q \times Q \to [0, 1]$ is the transition probability function such that for all $q \in Q$, $\sum_{q' \in Q} \delta(q, q') = 1$, and
- $L : Q \to \Sigma^{\text{out}}$ is a labeling function.

Definition 2 (DMC) A labeled Markov chain is *deterministic (DMC)*, if

- there exists a *start* state $q^s \in Q$ with $\mathbb{I}(q^s) = 1$, and
- for all $q \in Q$ and $\sigma \in \Sigma^{\text{out}}$: there exists at most one $q' \in Q$ with $L(q') = \sigma$ for which $\delta(q, q') > 0$.

Since the possible successor states in a DMC are uniquely labeled, we sometimes abuse notation and write $\delta(q, \sigma)$ for $\delta(q, q')$ where $L(q') = \sigma$.

Each state in the $\mathcal{M}^c$ represents a configuration of the system being modeled, and each transition represents the movement from one system configuration to another (quantified by a probability). An (infinite) *path* in $\mathcal{M}^c$ is a string of states: $h = q_0 q_1 \ldots \in Q^\omega$ where $q_i \in Q$ and $\delta(q_i, q_{i+1}) > 0$, for all $i \in \mathbb{N}$. The *trace* for h, denoted *trace*(h), is a sequence of state labels $s = \sigma_0\sigma_1 \ldots \in (\Sigma^{\text{out}})^\omega$, where $\sigma_i = L(q_i)$ for all $i \in \mathbb{N}$. Given a finite

path $h = q_0 q_1 \dots q_n$, the cylinder set of h, denoted $Cyl\,(q_0 q_1 \dots q_n)$, is defined as the set of infinite paths with the prefix h. The probability of the cylinder set is given by

$$P_{\mathcal{M}^c}(Cyl\,(q_0 q_1 \dots q_n)) = \mathbb{I}(q_0) \cdot \prod_{i=1}^{n} \delta(q_{i-1}, q_i).$$

For any trace s in a DMC, there exists at most one path h such that $trace(h) = s$, hence the definition above readily extends to cylinder sets for strings. If the MC is non-deterministic, there may exist more than one path with trace s in which case the probability of $Cyl\,(s)$ is given by

$$P_{\mathcal{M}^c}(Cyl\,(s)) = \sum_{h:trace(h)=s} P_{\mathcal{M}^c}(Cyl\,(h)).$$

The probabilities assigned to cylinder sets induce a unique probability distribution on $(\Sigma^{\text{out}})^\omega$ (equipped with the σ-algebra generated by the cylinder sets) (Baier and Katoen 2008). We denote this distribution also with $P_{\mathcal{M}^c}$. Moreover, we denote by $P_{\mathcal{M}^c,q}$ the distribution obtained by (re)defining $q \in Q$ as the unique *start* state.

Note that our definition of (D)MCs differs from other versions of probabilistic automata, such as Rabin (1963) and Segala (1996): we assume states to be labeled, whereas the more common automaton model puts the labels on the transitions. Both types of models are equivalent, but a translation of a transition-labeled automaton to a state-labeled automaton may increase the number of states by a factor of $|\,\Sigma^{\text{out}}\,|$. Despite the increase in model size, we still adopt (D)MCs as system models due to the model checking tools and algorithms already developed for this model class.

The MC is a purely probabilistic model, i.e., in a certain state, the probability of reaching a specific state in the next step is known. Deterministic labeled Markov decision processes (DMDPs) extend DMCs with non-determinism, which can be used to model reactive systems where input actions are chosen non-deterministically and the resulting output for a given input action is determined probabilistically.

Definition 3 (DMDP) A *deterministic labeled Markov decision process (DMDP)* is a tuple $\mathcal{M}^p = \langle Q, \Sigma^{\text{in}}, \Sigma^{\text{out}}, q^s, \delta, L \rangle$, where

- $Q, \mathbb{I},$ and L are the same as for DMCs,
- Σ^{in} is a finite alphabet of input actions,
- Σ^{out} is a finite alphabet of output symbols,
- the transition probability function is defined as $\delta : Q \times \Sigma^{\text{in}} \times Q \to [0, 1]$, such that for all $q \in Q$ and all $\alpha \in \Sigma^{\text{in}}$, $\sum_{q' \in Q} \delta(q, \alpha, q') = 1$, and
- for all $q \in Q$, $\alpha \in \Sigma^{\text{in}}$, and $\sigma \in \Sigma^{\text{out}}$, there exists at most one $q' \in Q$ with $L(q') = \sigma \in \Sigma^{\text{out}}$ and $\delta(q, \alpha, q') > 0$.

The last condition in the definition above together with the existence of a unique initial state q^s makes the behavior of the model deterministic conditioned on the (non-deterministically chosen) input actions. Analogously to DMCs, we will sometimes abuse notation and write $\delta(q, \alpha, \sigma)$ instead of $\delta(q, \alpha, q')$ where $L(q') = \sigma$. A path in a DMDP $\mathcal{M}^p$ is an alternating sequence of states $q_i \in Q$ and input symbols $\alpha_i \in \Sigma^{\text{in}}$, denoted as $q_0 \alpha_1 q_1 \alpha_2 q_2 \dots$. The trace of a path in a DMDP is defined analogously to the notion of trace in MCs. That is, the trace of a path $q_0 \alpha_1 q_1 \alpha_2 q_2 \dots$ is an alternating sequence of input symbols and state labels $\pi = \sigma_0 \alpha_1 \sigma_1 \alpha_2 \sigma_2 \dots \in \Sigma^{\text{out}} \times (\Sigma^{\text{in}} \times \Sigma^{\text{out}})^\omega$, where $\sigma_i = L(q_i)$. To reason about the probability of a set of paths in the DMDP, a *scheduler* (also known as an *adversary* or a *strategy*) is introduced to resolve the non-deterministic choices on the input actions.

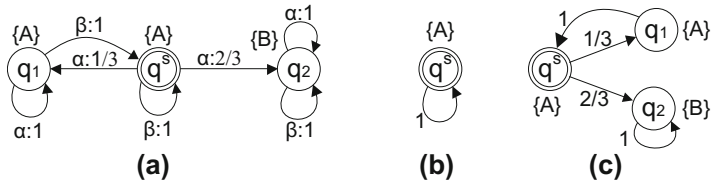

Fig. 1 **a** A DMDP $\mathcal{M}^p$. **b** The DMC $\mathcal{M}^c_{\mathfrak{S}_1}$ induced by the scheduler $\mathfrak{S}_1$. **c** The DMC $\mathcal{M}^c_{\mathfrak{S}_2}$ induced by the scheduler $\mathfrak{S}_2$

Definition 4 (Scheduler) Let $\mathcal{M}^p$ be a DMDP and Q^+ be the set of state sequences of non-zero length. A *scheduler* for $\mathcal{M}^p$ is a function $\mathfrak{S} : Q^+ \times \Sigma^{\text{in}} \to [0, 1]$ such that for all $q = q_0 q_1 \dots q_n \in Q^+$, $\sum_{\alpha \in \Sigma^{\text{in}}} \mathfrak{S}(q, \alpha) = 1$. A scheduler is said to be *deterministic* if for all $q \in Q^+$ there exists an $\alpha \in \Sigma^{\text{in}}$ for which $\mathfrak{S}(q, \alpha) = 1$.

The scheduler specifies an action for each state based on the path history for that state. It is said to be *fair* if in any state q all input actions can be chosen with non-zero probability. If a scheduler $\mathfrak{S}$ only depends on the current state we say that $\mathfrak{S}$ is *memoryless*. An $\mathcal{M}^p$ together with a scheduler $\mathfrak{S}$ induce a probability distribution defined by the cylinder set of all finite path fragments in $\mathcal{M}^p$. For a cylinder set $Cyl\,(q_0\alpha_1 q_1 \dots \alpha_n q_n)$ the probability is defined as

$$P_{\mathcal{M}^p,\mathfrak{S}}(Cyl\,(q_0\alpha_1 q_1 \dots \alpha_n q_n)) = \mathbb{I}(q_0) \cdot \prod_{i=1}^{n} \mathfrak{S}(q_0 \dots q_{i-1}, \alpha_i)\delta(q_{i-1}, \alpha_i, q_i).$$

Similarly to DMCs, the probability distribution defined above induces a probability distribution over cylinder sets of I/O strings, and hence a distribution over infinite I/O sequences.

Example 1 The graphical model of a three-state DMDP $\mathcal{M}^p$ is shown in Fig. 1a, where $\Sigma^{\text{in}} = \{\alpha, \beta\}$ and $\Sigma^{\text{out}} = \{A, B\}$. From the initial state q^s (double circled) labeled with symbol A, the actions α and β are chosen nondeterministically. Consider now the two memoryless schedulers $\mathfrak{S}_1$ and $\mathfrak{S}_2$ given by $\mathfrak{S}_1(q) = \beta$, and $\mathfrak{S}_2(q) = \alpha$ if $q = q^s$ and $\mathfrak{S}_2(q) = \beta$ otherwise. The schedulers induce the DMCs in Fig. 1b, c, where for the string $s = AAAA$ we have $P_{\mathcal{M}^c_{\mathfrak{S}_1}}(AAAA) = 1$, and $P_{\mathcal{M}^c_{\mathfrak{S}_2}}(AAAA) = 4/9$.

Both DMCs and DMDPs are discrete-time models, i.e., each transition takes a universal discrete time unit. The labeled deterministic continuous-time Markov chain (DCTMC) is a time-extension of the DMC, which models the amount of time the system stays in a specific state before making a transition to one of its successor states (Sen et al. 2004a; Chen et al. 2009).

Definition 5 (DCTMC) A *deterministic labeled continuous-time Markov chain (DCTMC)* is a tuple $\mathcal{M}^t = \langle Q, \Sigma^{\text{out}}, q^s, \delta, R, L, \rangle$, where:

- Q, Σ^{out}, q^s, δ, L are defined as for DMCs;
- $R : Q \to \mathbb{R}_{\geq 0}$ is the exit rate function.

In a DCTMC, the probability of making a transition from state q to one of its successor states q' within t time units is given by $\delta(q, q') \cdot \left(1 - e^{-R(q)\cdot t}\right)$, where $(1 - e^{-R(q)\cdot t})$ is the cumulative distribution of an exponential function with rate parameter $R(q)$.

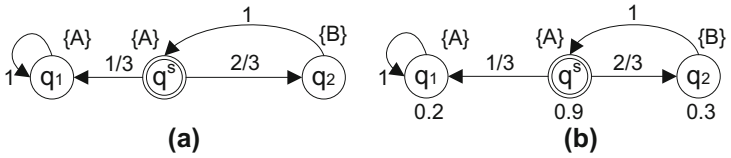

Fig. 2 **a** A DLMC $\mathcal{M}^c$ and **b** a structurally identical DCTMC $\mathcal{M}^t$ modeling the amount of time between state transitions

Example 2 Consider the DMC $\mathcal{M}^c$ and the DCTMC $\mathcal{M}^t$ shown in Fig. 2. Both models have three states with initial state q^s (double circled) and $\Sigma^{\text{out}} = \{A, B\}$. From q^s, the probability of taking one of its two transitions are $1/3$ and $2/3$, respectively. Compared with $\mathcal{M}^c$ in (a), the DCTMC $\mathcal{M}^t$ in (b) has exit-rates associated with the states, e.g., 0.9 on q^s. In $\mathcal{M}^t$, the probability of leaving the initial state and moving to state q_2 within t time units is calculated as $2/3 \cdot (1 - e^{0.9 \cdot t})$.

A *timed path* h in a DCTMC is an alternating sequence of states and time stamps $q_0 t_0 q_1 t_1 q_2 \ldots$, where $t_i \in \mathbb{R}_{>0}$ denotes the amount of time spent in state q_i before going to q_{i+1}. By adopting the notation for timed strings we let $h[n] = q_n$ and $h\langle n \rangle = t_n$.

Let $Cyl\ (q_0, I_0, \ldots, q_{k-1}, I_k, q_k)$ denote the cylinder set containing all paths with $h\langle i \rangle \in I_i$ and $h[i] = q_i$, for $i < k$. The probability of $Cyl\ (q_0, I_0, \ldots, q_{k-1}, I_k, q_k)$ is then defined inductively as follows (for $k \geq 1$) (Baier et al. 2003):

$$P_{\mathcal{M}^t}(Cyl\ (q_0, I_0, \ldots, q_{k-1}, I_k, q_k)) = P_{\mathcal{M}^t}(Cyl\ (q_0, I_0, \ldots, q_{k-1})) \cdot \delta(q_{k-1}, q_k) \cdot (e^{-R(q_{k-1}) \inf(I_k)} - e^{-R(q_{k-1}) \sup(I_k)}).$$

Following the definition of cylinder sets for DMCs, we can directly extend the definition above to probability distributions over cylinder sets for timed strings.

2.3 Specification languages

As will be detailed in Sect. 3, the proposed learning algorithms assume that data appears in the form of sequences of linearly ordered observations of the system in question. When learning system models, we therefore only look for models that preserve linear-time properties, which include safety properties (something bad will never happen) and liveness properties (something good will always happen).

Linear-time temporal logic (LTL) (Pnueli 1977) is a logical formalism used for specifying system properties from a linear time perspective. The property specified by an LTL formula does not only depend on the current state, but can also relate to future states. The basic ingredients of an LTL formula are atomic propositions (state labels $\sigma \in \Sigma^{\text{out}}$), the Boolean connectors conjunction ($\wedge$) and negation ($\neg$), and two basic temporal modalities $\bigcirc$ (*next*) and U (*until*) (Baier and Katoen 2008).

Definition 6 (LTL) Linear-time temporal logic (LTL) over Σ^{out} is defined by the following syntax

$$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \mathsf{U} \varphi_2, \text{ where } a \in \Sigma^{\text{out}}.$$

Definition 7 (LTL Semantics) Let φ be an LTL formula over Σ^{out}. For $s = \sigma_0 \sigma_1 \ldots \in (\Sigma^{\text{out}})^{\omega}$, the LTL semantics of φ are as follows:

– $s \models true$

- $s \models a$ iff $a = \sigma_0$
- $s \models \varphi_1 \wedge \varphi_2$ iff $s \models \varphi_1$ and $s \models \varphi_2$
- $s \models \neg \varphi$ iff $s \not\models \varphi$
- $s \models \bigcirc \varphi$ iff $s[1 \ldots] \models \varphi$
- $s \models \varphi_1 \mathsf{U} \varphi_2$ iff $\exists j \geq 0.\ s[j \ldots] \models \varphi_2$ and $s[i \ldots] \models \varphi_1$, for all $0 \leq i < j$

For better readability, we also use the derived temporal operators $\Box$ (*always*) and $\Diamond$ (*eventually*) given by $\Diamond\varphi = (true \mathsf{U} \varphi)$ (the model will eventually satisfy property φ) and $\Box\varphi = \neg(\Diamond\neg\varphi)$ (property φ always holds).

Model checking an MC $\mathcal{M}^c$ wrt. a LTL formula φ means to compute the total probability of the traces in $\mathcal{M}^c$ which satisfy φ, i.e., $P_{\mathcal{M}^c}(\{s \mid s \models \varphi, s \in (\Sigma^{\text{out}})^\omega\})$.

Example 3 The LTL formula $A \mathsf{U} B$ requires that a state q labeled with B will eventually be reached, and all states visited before q should all be labeled with A. For the DMC $\mathcal{M}^c$ in Fig. 2a, only paths starting with $q^s q_2$ satisfy the LTL formula. Model checking $\mathcal{M}^c$ wrt. $A \mathsf{U} B$ therefore amounts to computing the probability of all paths starting with $q^s q_2$, i.e., $P_{\mathcal{M}^c}(Cyl(q^s q_2)) = 2/3$. The LTL formula $\Diamond\Box A$, read as *eventually forever* A, requires that after a certain point only states labeled with A will be visited. Paths starting from q_1 satisfy $\Box A$ and paths *eventually* reaching q_1 satisfy $\Diamond\Box A$. Model checking $\mathcal{M}^c$ wrt. $\Diamond\Box A$ can therefore be similarly reduced to the calculation of the probability $P_{\mathcal{M}^c}(\bigcup_{i \in [0,\infty)} Cyl(q^s (q_2 q^s)^i q_1)) = 1/3 + 2/3 \cdot 1/3 + (2/3)^2 \cdot 1/3 + \cdots = 1$.

The quantitative analysis of a DMDP $\mathcal{M}^p$ against a specification φ amounts to establishing the lower and upper bounds that can be guaranteed when ranging over all possible schedulers. This corresponds to computing

$$P^{\max}_{\mathcal{M}^p}(\varphi) = \sup_{\mathfrak{S}} P_{\mathcal{M}^p,\mathfrak{S}}(\varphi) \quad \text{and} \quad P^{\min}_{\mathcal{M}^p}(\varphi) = \inf_{\mathfrak{S}} P_{\mathcal{M}^p,\mathfrak{S}}(\varphi),$$

where the infimum and supremum are taken over all possible schedulers for $\mathcal{M}^p$.

Continuous stochastic logic (CSL) (Baier et al. 2003) is a general branching-time temporal logic proposed for CTMCs that allows for a recursive combination of state and path formulas. However, as discussed in the beginning of the section, we only consider linear time properties of system models and we therefore define a linear sub-class of CSL, called sub-CSL, in which at most one temporal operator is allowed.

Definition 8 (sub-CSL) A sub-CSL formula φ is defined as follows:

$$\varphi ::= \Phi \mid \Phi_1 \mathsf{U}_I \Phi_2 \mid \Diamond_I \Phi \mid \Box_I \Phi,$$

where Φ is a propositional logic formula defined as $\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \quad a \in \Sigma^{\text{out}}$, and I is an interval in $\mathbb{Q}_{\geq 0}$.

Definition 9 (Semantics for sub-CSL) Let φ be a sub-CSL formula over Σ^{out}. The semantics of φ over a timed trace $\rho = \sigma_0 t_0 \sigma_1 t_1 \ldots$ over Σ^{out} is as follows

- $\rho \models \Phi_1 \mathsf{U}_I \Phi_2$, iff $\exists t \in I.\ (\rho@t \models \Phi_2 \wedge \forall t' < t, \rho@t' \models \Phi_1)$
- $\rho \models \Diamond_I \Phi$, iff $\exists t \in I.\ (\rho@t \models \Phi)$
- $\rho \models \Box_I \Phi$, iff $\forall t \in I.\ (\rho@t \models \Phi)$

The semantics for the Boolean connectives are defined as for LTL.

Model checking a CTMC $\mathcal{M}^t$ wrt. a sub-CSL formula φ amounts to computing the probability of the timed traces which satisfy φ, i.e., $P_{\mathcal{M}^t}(\varphi) = P_{\mathcal{M}^t}(\{\rho \mid \rho \models \varphi\})$.

Example 4 The sub-CSL formula $\varphi = A \cup_{[1.5,2.3]} B$ requires that a state q labeled with B will be reached within the time interval $[1.5, 2.3]$ and that all states visited before q are labeled with A. For instance, the path $q^s\ 1.8\ q_2$, generated by the DCTMC $\mathcal{M}^t$ in Fig. 2b, satisfies φ. Model checking $\mathcal{M}^t$ against φ amounts to calculating $P_{\mathcal{M}^t}(Cyl(q^s, [1.5, 2.3], q_2)) = 2/3 \cdot (e^{-0.9\times1.5} - e^{-0.9\times2.3}) \approx 0.0444$.

3 Learning stochastic models

In what follows we consider methods for automatically learning stochastic system models, as defined in Sect. 2, from data. The proposed algorithms are based on the ALERGIA algorithm (Carrasco and Oncina 1994; Higuera 2010) and adapted to a verification context. The ALERGIA algorithm starts with the construction of a *frequency prefix tree acceptor* (FPTA), which serves as a representation of the data. The basic idea of the learning algorithm is to approximate the generating model by *merging* together nodes in the FPTA which correspond to the same state in the generating model. Two nodes are merged after they pass a *compatibility* test based on the statistical information associated with the nodes. Both the *compatibility* test and the state *merge* are conducted recursively over all successor nodes.

In this section, we first present the original FPTA for strings, which only contain output symbols, and then extend it to handle I/O strings and timed strings. Afterwards, we discuss the general procedure of the ALERGIA algorithm. At the end, we customize the compatibility tests and merge operations for learning different types of system models.

3.1 Data representation

An FPTA T represents a set of strings S over Σ^{out} in a tree structure, where each node is labeled by a symbol $\sigma \in \Sigma^{\text{out}}$ and each path from the root to a node q_s corresponds to a string $s \in \text{prefix}(S)$. Since a string s uniquely identifies a node in T and vice versa, we will sometimes use the symbol q_s for states and s for strings interchangeably. Each node q_s is associated with a *transition frequency function* $f(q_s, \sigma)$, which encodes the number of strings with prefix $s\sigma$ in S; we define $f(s, \cdot) = \sum_{\sigma\in\Sigma^{\text{out}}} f(s, \sigma)$. The successor state of q_s given σ is denoted $succ(s, \sigma) = s\sigma$, and the set of all successor states of q_s is denoted $succs(s)$. By normalizing the transition frequency functions $f(s, \sigma)$ by $f(s, \cdot)$ we obtain the *transition probability functions* $\delta(s, \sigma)$. Figure 3a shows an FPTA constructed from observation sequences generated by the DMC in Fig. 2b. The root of the tree is labeled with the symbol A and associated with the frequencies $f(A, B) = 15$ and $f(A, A) = 7$. The frequency functions indicate that in the dataset there are 15 strings with prefix AB, 7 strings with prefix AA and there are 22 strings with prefix A, i.e., $f(A, \cdot) = 22$.

The *I/O frequency prefix tree acceptor* (IOFPTA) is an extension of the FPTA for representing a set of I/O strings S_{io}. In addition to the output symbols $\sigma \in \Sigma^{\text{out}}$ attached to the nodes, each edge is labeled with an input action $\alpha \in \Sigma^{\text{in}}$. Similar to FPTAs, a string from the root to a node q_π corresponds to an I/O string $\pi \in \text{prefix}(S_{io})$. A *transition frequency function* $f(\pi, \alpha, \sigma)$ is associated with the node q_π, to encode the number of strings with the prefix $\pi\alpha\sigma$ in S_{io}. As for FPTAs, we let $f(\pi, \alpha, \cdot) = \sum_{\sigma\in\Sigma^{\text{out}}} f(\pi, \alpha, \sigma)$.

By normalizing the transition frequency functions we obtain the *transition probability functions* $\delta(\pi, \alpha, \sigma)$ for the IOFPTA. Figure 3b shows an IOFPTA constructed from I/O strings obtained from the DMDP in Fig. 1a.

A *timed frequency prefix tree acceptor* (TFPTA) represents a set of timed strings S_t. A TFPTA is structurally identical to an FPTA and can be obtained from the skeleton of S_t.

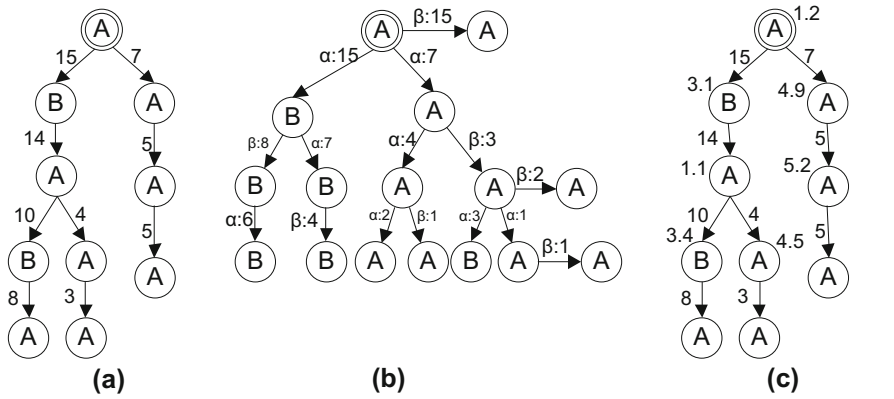

Fig. 3 Examples of frequency prefix tree acceptors

Thus, the path from the root to a node q_s in an TFPTA corresponds to a prefix of the skeleton of a timed string in S_t, i.e., $s \in \text{prefix}(\mathbb{S}(S_t))$. The transition frequency function associated with a node q_s is defined as for FPTAs by only considering the skeleton of S_t. In addition to the *transition frequency function*, each node q_s is also associated with an *average empirical exit time* $\hat{t}(s)$ (which is approximately the inverse of the exit-rate):

$$\hat{t}(s) = \frac{1}{f(s,\cdot)} \cdot \sum\nolimits_{\rho \in X} \rho\langle |s| \rangle,$$

where $X = \{\rho \mid s \in \text{prefix}(\mathbb{S}(\rho)), \rho \in S_t\}$ and $|s|$ is the number of symbols in the string s. Figure 3c illustrates an TFPTA constructed from strings sampled from the DCTMC in Fig. 2. Each node in the tree is associated with an average exit time, i.e., the time spent in the state before observing the next symbol. With the symbol A occurring 22 times as prefix of a string, we get an average exit time of 1.2 time units for the root node and the estimation of the exit rate is therefore $\frac{1}{1.2} \approx 0.83$.

3.2 ALERGIA

In this section we first sketch the main flow of the ALERGIA algorithm for learning DMCs as a modified version of the algorithm presented in Carrasco and Oncina (1994) and Higuera (2010). Afterwards we adapt the general learning algorithm to the different stochastic system models considered in this paper.

The ALERGIA algorithm is initialized by creating two identical FPTAs T and A as representations of the dataset S (line 2 of Algorithm 1). The FPTA T is kept as a data representation from which relevant statistics are retrieved during the execution of the algorithm. The FPTA A is iteratively transformed by merging nodes that have passed a statistical *compatibility* test. Observe that an FPTA (with normalized transition functions) is a DMC, and so is any model obtained by iteratively merging nodes. Similar properties hold for IOFPTAs and TFPTAs.

All compatibility tests are based on T to ensure the statistical validity of the compatibility tests that are performed. In some accounts of the ALERGIA algorithm it is suggested to join samples associated with different nodes of the original FPTA when the nodes are merged (Carrasco and Oncina 1994), and to base subsequent tests on these joined samples. While intuitively beneficial, since more data becomes available for testing, this latter approach invalidates some statistical arguments for the overall consistency of the algorithm:

if S_1 and S_2 are two sets of samples that each are drawn by independent sampling from the same distribution, then the union $S_1 \cup S_2$ no longer is a set of independent samples, if the union is performed conditional on the fact that S_1 and S_2 have passed a statistical test of compatibility. Since the assumption of independent sampling underlies all statistical tests we are using, such a join, therefore, makes a theoretical analysis of the resulting procedure very challenging. In order to maintain a strong match between the algorithmic solution, and the theoretical analysis we can provide, we generally do not join the associated samples when merging nodes. However, we have also conducted a few experiments comparing the performance of the algorithm with and without joining of the samples. It turned out that the differences in the constructed models and the runtime were only minor (cf. Sect. 5.1).

Following the terminology of Higuera (2010), Algorithm 1 maintains two sets of nodes: RED nodes, which have already been determined as representative nodes and will be included in the final output model, and BLUE nodes which are scheduled for testing. Initially, RED contains only the initial node q_A^s while BLUE contains the immediate successor nodes of the initial node. When performing the outer loop of the algorithm, the lexicographically minimal node q_b in BLUE will be chosen. If there exists a node q_r in RED which is compatible with q_b, then q_b and its successor nodes are merged to q_r and the corresponding successor nodes of q_r, respectively (line 10). If q_b is not compatible with any state in RED, it will be included in RED (line 15). At the end of each iteration, BLUE will be updated with the immediate successor nodes of RED that are not contained in RED (line 17). After merging all compatible nodes in the tree, the frequencies in A are normalized (line 18 of Algorithm 1).

In order to adapt the ALERGIA algorithm to the different model classes presented in Sect. 2, we only need to tailor the compatibility test (line 9) and the merge operator (line 10) to each specific model class. In the following section, the required model-specific compatibility tests and merge operators are presented.

Algorithm 1 ALERGIA

Input: : A set S of strings and a parameter $\epsilon > 0$.
Output: : A probabilistic model A.
1: $T \leftarrow FPTA(S)$
2: $A \leftarrow T$
3: RED $\leftarrow \{q_A^s\}$
4: BLUE $\leftarrow \{q \mid q \in succs(q_r), q_r \in \text{RED}\}$
5: **while** BLUE $\neq \emptyset$ **do**
6: $q_b \leftarrow$ lexicographically minimal $q \in$ BLUE
7: $merged \leftarrow false$
8: **for** $q_r \in$ RED & !$merged$ /* q_r in lexicographic order */ **do**
9: **if** $Compatibility(T, q_r, q_b, \epsilon)$ **then**
10: $Merge(A, q_r, q_b)$
11: $merged \leftarrow true$
12: **end if**
13: **end for**
14: **if** !$merged$ **then**
15: RED $\leftarrow$ RED $\cup \{q_b\}$
16: **end if**
17: BLUE $\leftarrow \{q \mid q \notin \text{RED}, q \in succs(q_r), q_r \in \text{RED}\}$
18: **end while**
$A \leftarrow Normalize(A)$;
19: **return** A

3.3 Local compatibility test and merge

Formally, two nodes q_r and q_b in an FPTA T are said to be ϵ-*compatible* ($\epsilon > 0$), if the following properties are satisfied:

1. $L(q_r) = L(q_b)$,
2. *LocalCompatible*(q_r, q_b, ϵ) is TRUE, and
3. the successor nodes of q_r and q_b are pair-wise ϵ-*compatible*.

Algorithm 2 *Compatibility test*

Input: : FPTA T , nodes q_r and q_b, and $\epsilon > 0$
Output: : *true* if q_r and q_b are compatible
1: **if** $L(q_r) \neq L(q_b)$ **then**
2: **return** *false*
3: **end if**
4: **if** !*LocalCompatible*(q_r, q_b, ϵ) **then**
5: **return** *false*
6: **end if**
7: **for all** $\sigma \in \Sigma^{\text{out}}$ **do**
8: $q_r' \leftarrow succ(q_r, \sigma)$
9: $q_b' \leftarrow succ(q_b, \sigma)$
10: **if** !*Compatibility*$(T, q_r', q_b', \epsilon)$ **then**
11: **return** *false*
12: **end if**
13: **end for**
14: **return** *true*

Algorithm 2 illustrates the *compatibility* test. Condition (1) requires the two nodes to have the same label (line 1). Condition (2) is model-specific and defines the local compatibility test for q_r and q_b (line 4). The last condition requires the compatibility to be recursively satisfied for every pair of successor nodes of q_r and q_b (line 10). Note that only pairs of successor nodes reached by the same output symbol (as well as the same input symbol in the IOFPTA case) are tested. For example, q_r' and q_b' are being tested only if $q_r' = succ(q_r, \sigma)$ and $q_b' = succ(q_b, \sigma)$ (in an IOFPTA, q_r' and q_b' are determined as $q_r' = succ(q_r, \alpha, \sigma)$ and $q_b' = succ(q_b, \alpha, \sigma)$).

The compatibility test depends on a parameter ϵ that controls the severity of the *LocalCompatible* tests, which are defined so that smaller values of ϵ will make *LocalCompatible* return *false* less often. In most cases, ϵ directly translates to the significance level of a statistical test that is the core of the *LocalCompatible* test.

In the following sections, we start by specifying the *local compatibility* test and *merge* procedure for FPTAs, and afterwards extend the specifications to IOFPTAs and TFPTAs. For FPTAs and IOFPTAs, the local compatibility test depends only on the local transition frequency functions, whereas for TFPTAs we also need to take the estimated exit rates into account. Analogous considerations apply for the *merge* procedure.

3.3.1 Local compatibility test and merge in FPTAs

Given two nodes q_r and q_b in an FPTA, their *local compatibility* requires that the difference between the next symbol distributions defined at two nodes is bounded. Specifically, we check for local compatibility (Line 4 in Algorithm 2) by employing the *Hoeffding* test (see

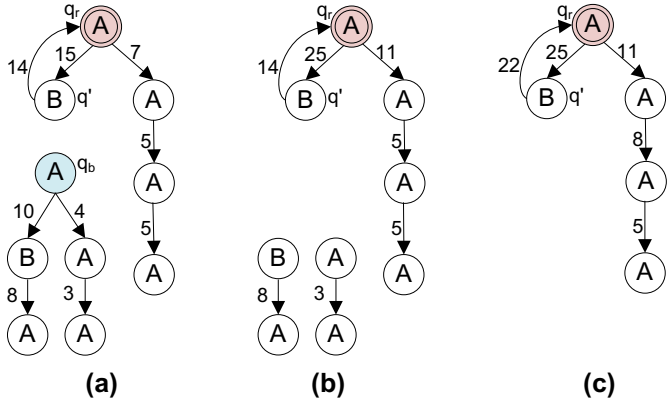

Fig. 4 Merge node q_b (shadowed) to node q_r (shadowed and double circled) in the FPTA. **a** The transition from q' to q_b is redirected to q_r. **b** Node q_b and its two outgoing transitions are folded to q_r and the frequencies are updated: $f(q_r, B) = f(q_r, B) + f(q_b, B) = 25$ and $f(q_r, C) = f(q_r, C) + f(q_b, C) = 11$. **c** The resulting FPTA obtained after recursively folding the successor nodes of q_r and q_b

Algorithm 3) realized by the call $Hoeffding(f(q_r, \sigma), f(q_r, \cdot), f(q_b, \sigma), f(q_b, \cdot), \epsilon)$, for all $\sigma \in \Sigma^{\text{out}}$. Line 4 of Algorithm 3 is a statistical test for the identity of the transition probabilities at the states q_r and q_b to their σ-successors (Carrasco and Oncina 1999). The actual statistical level of significance of this test is given by 2ϵ rather than ϵ itself. However, for the asymptotic consistency analysis that we will be concerned with in Sect. 4 and "Consistency of Alergia-style Learning" of Appendix the constant factor 2 is immaterial, and we will a little loosely refer to ϵ as the significance level of the Hoeffding compatibility test. Also observe that the feasible range of the ϵ parameter is $(0, 2]$. At $\epsilon = 2$ line 4 will always return *false*.

Algorithm 3 *Hoeffding*

Input: : $f_1, n_1, f_2, n_2, \epsilon \in (0, 2]$
Output: : *true* if f_1/n_1 and f_2/n_2 are sufficiently close
1: **if** $n_1 == 0$ or $n_2 == 0$ **then**
2: **return** *true*
3: **end if**
4: **return** $|\frac{f_1}{n_1} - \frac{f_2}{n_2}| < (\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_2}}) \cdot \sqrt{\frac{1}{2} \ln \frac{2}{\epsilon}}$

If two nodes q_r and q_b are *compatible*, q_b is merged to q_r. The *merge* procedure (line 10 of Algorithm 1) follows the same steps as described in Higuera (2010). Firstly, the (unique) transition leading to q_b from its predecessor node q' ($f^A(q', q_b) > 0$) is re-directed to q_r by setting $f^A(q', q_r) \leftarrow f^A(q', q_b)$ and $f^A(q', q_b) = 0$. Secondly, the successor nodes of q_b are recursively folded to the corresponding successor nodes of q_r and the associated frequencies are updated. The complete merge procedure is illustrated in Fig. 4.

3.3.2 Local compatibility test and merge in IOFPTAs

In an IOFPTA, the transition frequency function on node q, $f(q_r, \alpha, q')$, is also conditioned on the input action α. Thus, in order to adapt the local compatibility test to IOFPTAs, we compare the transition probability distribution defined for each input action.

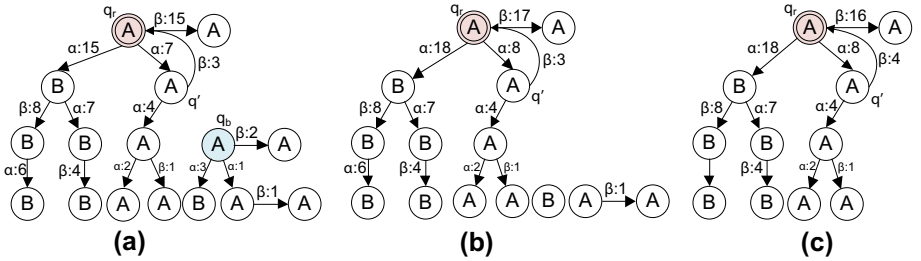

Fig. 5 Node q_b is merged to node q_r in the IOFPTA. **a** The transition from q' to q_b is redirected to q_r. **b** Successor nodes of q_b are locally folded along input actions to q_r. **c** The IOFPTA resulting from recursively folding the subtrees rooted at q_r and q_b

Specifically, given two nodes q_r and q_b, the *Hoeffding* test, realized by the procedure call $Hoeffding(f(q_r, \alpha, \sigma), f(q_r, \alpha, \cdot), f(q_b, \alpha, \sigma), f(q_b, \alpha, \cdot), \epsilon)$, is conducted for all $\sigma \in \Sigma^{\text{out}}$ conditioned on $\alpha \in \Sigma^{\text{in}}$. Similarly, the test is performed iteratively for all input actions at two given nodes.

The merge procedure for two compatible nodes in IOFPTA is similar to the one in FPTAs. An example is shown in Fig. 5. Observe that the frequencies are aggregated along the different input actions.

3.3.3 Local compatibility test and merge in TFPTAs

The nodes in a TFPTA are associated with transition frequency functions and exit-rates encoding the local transition times. We therefore define two nodes q_r and q_b in a TFPTA to be compatible if the transition probability distributions over their successor nodes as well as their exit-rates are compatible. The compatibility of transition distributions for two nodes are, as for MCs, tested by the Hoeffding test (Algorithm 3). The compatibility test of the exit rates follows the procedure described in Sen et al. (2004a), which is essentially the F-test originally introduced in Cox (1953). The test is based on the ratio $\hat{t}_r/\hat{t}_b$ of the average empirical time delays at q_r and q_b. The precise test criterion is given in Algorithm 4.

Algorithm 4 *F-test*

Input: : $\hat{t}_r, n_r, \hat{t}_b, n_b, \epsilon \in (0, 1]$
Output: : *true* if $\hat{t}_r$ and $\hat{t}_b$ are sufficiently close
1: **if** $n_r \leq 1$ or $n_b \leq 2$ **then**
2: **return** *true*
3: **end if**
4: $\mu = \frac{n_b}{n_b - 1}$
5: $\sigma = \sqrt{\frac{(n_b)^2(n_r+n_b-1)}{n_r(n_b-1)^2(n_b-2)}}$
6: $\gamma_1 = \mu - \frac{\sigma}{\sqrt{\epsilon}}, \gamma_2 = \mu + \frac{\sigma}{\sqrt{\epsilon}}$
7: **return** $\frac{\hat{t}_r}{\hat{t}_b} \in [\gamma_1, \gamma_2]$

4 Consistency and convergence analysis

In this section we investigate theoretical convergence results for ALERGIA learning. These results consist of two components: first, we establish that in the large sample limit the learning algorithm will correctly identify the structure of a data generating model (modulo equivalence of the states). This component is related to previous convergence results (Carrasco and Oncina 1999; de la Higuera and Thollard 2000; Sen et al. 2004a), and we provide the main technical results in "Consistency of Alergia-style Learning" of Appendix. The second component is to establish that identification of the structure together with convergence of the estimates for the probabilistic parameters of the models (transition probabilities and exit rates) guarantees convergence of the probabilities for model properties expressed in our formal specification languages.

Our analysis, thus, focuses on exact identification in the limit, and thereby differs from *probably approximately correct (PAC)* learning results, such as presented in Ron et al. (1996) and Clark and Thollard (2004). PAC learning results are stronger than identification in the limit results in that they provide bounds on the sample complexity required to learn a good approximation of the true model. However, a PAC learnability analysis first requires the specification of a suitable metric to measure the quality of approximation. Existing PAC learning approaches for probabilistic automata are based on a semantics for the automata as defining a probability distribution over Σ^*. In that case, the Kullback-Leibler divergence between the distributions defined by the true and the approximate model is a canonical measure of approximation error.

Being interested in the probability of LTL properties, we, on the other hand, have to see automata as defining distributions on Σ^ω. The Kullback–Leibler divergence between the distributions defined on Σ^ω is not a suitable measure for approximation quality, since it will almost always be infinite (even in the case where the approximate model is structurally identical to the true one, and differs with respect to transition probabilities only by an arbitrarily small $\epsilon > 0$). Within the verification literature, various versions of the *bisimulation distance* are a popular measure for approximate equivalence between system models (Desharnais et al. 1999; Breugel and Worrell 2005). However, it turns out that these metrics suffer from the same problem as the Kullback-Leibler distance, and fail to measure approximation quality as a smooth function of ϵ-errors in the estimates of transition probabilities. These and other candidate measures for approximate equivalence of automata defining distributions on Σ^ω are investigated in detail in Jaeger et al. (2014). A number of counterexamples and impossibility results derived in Jaeger et al. (2014) indicate that there exist fundamental obstacles to defining measures for approximation error that simultaneously satisfy the two desiderata: (a) to provide a basis on which PAC learnability results could be derived, and (b) small approximation errors between models should also entail bounds on the differences between the probabilities of LTL properties in the models (a desideratum called "LTL continuity" in Jaeger et al. (2014)).

For the analysis of the identification of the structure, we now begin by formally defining the relevant equivalence relation of states. In the following, for any automaton $\mathcal{M}$ and state q of $\mathcal{M}$, we denote with $(\mathcal{M}, q)$ the automaton obtained by (re-)defining q as the start state of $\mathcal{M}$.

Definition 10 Let $\mathcal{M}$ be a DLMC or DCTMC. States q, q' of $\mathcal{M}$ are equivalent, written $q \sim q'$, if $P_{(\mathcal{M},q)} = P_{(\mathcal{M},q')}$. States q, q' of a DMDP $\mathcal{M}$ are equivalent, if for all schedulers $\mathfrak{S}$ of $(\mathcal{M}, q)$ there exists a scheduler $\mathfrak{S}'$ of $(\mathcal{M}, q')$, such that $P_{(\mathcal{M},q),\mathfrak{S}} = P_{(\mathcal{M},q'),\mathfrak{S}'}$, and vice-versa.

When $q \sim q'$, then also $\delta(q, \alpha, \sigma) \sim \delta(q', \alpha, \sigma)$ for all $(\alpha, \sigma) \in \Sigma^{\text{in}} \times \Sigma^{\text{out}}$. Therefore, δ is also well-defined on $(Q/\sim) \times \Sigma^{\text{in}} \times (Q/\sim)$, and we thereby obtain the quotient automaton $\mathcal{M}/\sim$ whose states are the $\sim$-equivalence classes of $\mathcal{M}$.

Next, we formally define the structure of an automaton.

Definition 11 Let $\mathcal{M}$ be a DLMC, DMDP, or DCTMC. The *structure* of $\mathcal{M}$ is defined as $\widehat{\mathcal{M}} := \langle Q, \Sigma^{\text{in}}, \Sigma^{\text{out}}, q^s, \hat{\delta}, L\rangle$, where $\hat{\delta} \subseteq Q \times \Sigma^{\text{in}} \times Q$ is the transition relation defined by $(q, \alpha, q') \in \hat{\delta} \Leftrightarrow \delta(q, \alpha, q') > 0$.

For DLMCs and DCTMCs the Σ^{in} component should be regarded as vacuous in the preceding definition.

The first component of the convergence result will be the identification in the limit of $\widehat{\mathcal{M}/\sim}$. Before we can state that result, however, we have to consider the question of how training data for the learner is assumed to be generated. Since our automaton models are generative models for infinite sequences, one cannot simply assume that the training data consists of sampled runs of an automaton. All we can observe (and all that ALERGIA will accept) are finite initial segments of such runs. Thus, in the data-generation process, one has to assume that there is an external process that decides at what point the generation of a sequence is terminated. Furthermore, in the case of DMDP learning, an external scheduler is required to generate inputs. Both these external components must satisfy certain conditions, so that the generated data is rich enough to contain sufficiently many sampled transitions from all states and under all inputs. At the same time, the significance level ϵ for ALERGIA must be chosen so that certain correctness guarantees for the compatibility tests performed during the execution of the algorithm are obtained. The sampling mechanism for finite sequences and the choice of significance levels for the compatibility tests are interrelated. The details of this relationship are elaborated in "Appendix". For the present section, we only consider the case where data is generated as follows:

- The length of the observed sequence is randomly determined by a geometric distribution with parameter λ. This is equivalent to generating strings with an automaton where at each state the generating process terminates with probability λ.
- Inputs are generated by a scheduler that always chooses inputs uniformly at random.

We refer to this procedure as *geometric sampling*. It defines a probability distribution P^s on $(\Sigma^*)^\omega$, where depending on the underlying automaton, Σ is Σ^{out} (for DLMCs), $\Sigma^{\text{in}} \times \Sigma^{\text{out}}$ (for DMDPs), or $\Sigma^{\text{out}} \times \mathbb{R}_{>0}$ (for DCTMCs).

Theorem 1 *Let $\mathcal{M}$ be a DLMC or DMDP. Let $S \in (\Sigma^*)^\omega$ be generated by geometric sampling. Let $\epsilon_N = 1/N^r$ for some $r > 2$, and let $\mathcal{M}_N$ be the model learned by* ALERGIA *from the first N strings in S using significance level ϵ_N in the compatibility tests. Then*

$$P^s(\widehat{\mathcal{M}_N} = \widehat{\mathcal{M}/\sim} \text{ for almost all } N) = 1. \tag{1}$$

Let $\mathcal{M}$ be a DCTMC, and S as above. There exist values ϵ_N with $1/N \leq \epsilon_N \leq 1/\sqrt{N}$, such that for $\mathcal{M}_N$ the model learned by ALERGIA *from the first N strings in S using significance level ϵ_N:*

$$\lim_{N\to\infty} P^s(\widehat{\mathcal{M}_N} = \widehat{\mathcal{M}/\sim}) = 1. \tag{2}$$

(2) also holds when $\mathcal{M}$ is a DLMC or DMDP, and $\epsilon_N = 1/N^r$ for some $r \geq 1$.

The Theorem is a consequence of Theorem 4 and Lemmas 3 and 4 in "Consistency of Alergia-style Learning" of Appendix. The second part of the Theorem does not provide a

complete description of the required sequence of significance levels ϵ_N, because the exact ϵ_N values (obtained in the proof of Lemma 4) are defined in terms of the expected values of the size of the IOFPTA constructed from a sample of size N, and we can only bound this expected value, but do not have a closed-form expression as a function of N.

The reason we obtain somewhat stronger convergence guarantees for DLMCs and DMDPs than for DCTMCs lies in the fact that we have stronger results on the power of the Hoeffding test, than the F-test (cf. "Statistical Tests" of Appendix). It is an open problem whether almost sure convergence actually also holds for DCTMCs with the currently used F-test, or whether it could be obtained with a different, more powerful test for the compatibility of exponential distributions.

We are now ready to turn to the second component of our consistency analysis: ultimately, we are interested in whether the probabilities of properties expressed in the formal specification languages LTL and sub-CSL computed on the learned models converge to the probabilities defined by the true model. By Theorem 1 we know that the learned model will eventually have the correct structure, and the laws of large numbers also guarantee that the estimates of the transition probability and exit rate parameters will converge to the correct values. This, however, in general will not be enough to guarantee the convergence of the probabilities of complex system properties. As the following two Theorems show, however, we do obtain such a guarantee for properties expressed in LTL and sub-CSL. Since the sub-CSL case here is simpler, we consider it first.

Theorem 2 *Let $\mathcal{M}$ be a DCTMC. Let $\mathcal{M}_N$ as in Theorem 1. For all sub-CSL properties φ, and all $\delta > 0$ then:*

$$\lim_{N\to\infty} P^s(\,|\,P_{\mathcal{M}_N}(\varphi) - P_{\mathcal{M}}(\varphi)\,|> \delta) = 0.$$

Proof By Theorem 1 we have that the probability that $\mathcal{M}_N$ and $\mathcal{M}/\sim$ have different structures is negligible in the limit. Conditional on $\mathcal{M}_N$ and $\mathcal{M}/\sim$ having the same structure, we also have by the law of large numbers that the parameters of $\mathcal{M}_N$ converge to the parameters of $\mathcal{M}/\sim$. It is therefore sufficient to show that then also $P_{\mathcal{M}_N}(\varphi)$ converges to $P_{\mathcal{M}/\sim}(\varphi) = P_{\mathcal{M}}(\varphi)$.

All properties φ expressible in sub-CSL are finite-horizon in the sense that there exists a fixed time limit t, such that whether a timed trace $\rho = \sigma_0 t_0 \sigma_1 t_1 \ldots$ satisfies φ only depends on the prefix $\rho[0:k]$, where k is such that $t_0 + \cdots + t_k > t$. For a purely propositional formula Φ this is $t = 0$, and for a formula containing a temporal operator with subscript I, t is the upper bound I^u of I. The set of traces satisfying φ, therefore, can be represented as a countable disjoint union of sets of paths that are slightly generalized forms of cylinder sets. For example, the set of paths satisfying $\Phi_1 \mathsf{U}_I \Phi_2$ is the union over all paths of the form $q_0 t_0 \ldots q_{k-1} t_{k-1} q_k t_k$ where $q_0, \ldots, q_{k-1}$ satisfy Φ_1, q_k satisfies Φ_2, and $t_0 + \cdots + t_k \in I$. The probabilities of such slightly generalized cylinder sets are a continuous function of the transition probability and exit rate parameters of $\mathcal{M}_N$, and therefore the convergence of these parameters guarantees the convergence of the probabilities of the generalized cylinder sets, and thereby the convergence of the probability of φ. □

We now state the corresponding results for LTL and DMDPs, which subsumes the case of LTL and DLMCs

Theorem 3 *Let $\mathcal{M}$ be a DMDP, and $\mathcal{M}_N$ as in Theorem 1 using significance levels $\epsilon_N = 1/N^r$. If $r > 2$, then for all LTL properties φ:*

$$P^s(\lim_{N\to\infty} P^{\max}_{\mathcal{M}_N}(\varphi) = P^{\max}_{\mathcal{M}}(\varphi)) = P^s(\lim_{N\to\infty} P^{\min}_{\mathcal{M}_N}(\varphi) = P^{\min}_{\mathcal{M}}(\varphi)) = 1. \tag{3}$$

If $r \geq 1$, then for all $\delta > 0$:

$$\lim_{N\to\infty} P^s(\,|\,P^{\max}_{\mathcal{M}_N}(\varphi) - P^{\max}_{\mathcal{M}}(\varphi)\,|> \delta) = \lim_{N\to\infty} P^s(\,|\,P^{\min}_{\mathcal{M}_N}(\varphi) - P^{\min}_{\mathcal{M}}(\varphi)\,|> \delta) = 0 \quad (4)$$

The following is a slightly generalized version of the proof that was given for DLMCs in Mao et al. (2011).

Proof Using the automata-theoretic approach to verification (Vardi 1985; Courcoubetis and Yannakakis 1995; Vardi 1999; Baier and Katoen 2008, Section 10.6.4), the probabilities $P^{\max}_{\mathcal{M}_N}(\varphi)$ and $P^{\max}_{\mathcal{M}}(\varphi)$ can be identified with maximum reachability probabilities in the respective products of $\mathcal{M}_N$ and $\mathcal{M}$ with a Rabin automaton B representing ϕ. The maximum here is with respect to all possible memoryless schedulers on the product MDPs. Since $\mathcal{M}$ and $\mathcal{M}/\sim$ are equivalent with respect to LTL properties, one can consider the product of $\mathcal{M}/\sim$ with B instead, which then by Theorem 1 for the case $r > 2$ will for almost all N have the same structure as the product of $\mathcal{M}_N$ with B. Maximum reachability probabilities in the product MDPs are a continuous function of the transition probability parameters on the interior of the parameter space, i.e., for sequences of parameters $p_N \to p$ where $p \neq 0, 1$. Since $\mathcal{M}_N$ and $\mathcal{M}/\sim$ agree on all 0/1-valued parameters, and for all others the parameters of $\mathcal{M}_N$ converge to those of $\mathcal{M}/\sim$, one also obtains $P^{\max}_{\mathcal{M}_N}(\varphi) \to P^{\max}_{\mathcal{M}}(\varphi)$. The argument for $P^{\min}$ is analogous by considering minimum reachability instead of maximum reachability. The proof for the case $r \geq 1$ is identical, using the weaker convergence guarantee of Theorem 1 for this case. □

Theorem 3 makes a strictly stronger statement for the choice of significance levels $\epsilon_N = 1/N^r$ with $r > 2$. However, all statements are strictly asymptotic, and these very small ϵ_N-values may lead to significantly under-estimate the size of the generating model when learning from a given limited dataset. In practice, therefore, one may prefer the weaker guarantees obtained for $\epsilon_N = 1/N$ in exchange for a lower risk of learning an over-simplified model.

An important observation is that Theorems 2 and 3 are pointwise for each φ, and not uniform for the whole languages sub-CSL and LTL, respectively. Thus, it is not the case that in the limit we will learn a model that simultaneously approximates the probabilities of all properties ϕ to within a fixed error bound δ. In other words, the sample size N required to obtain a good approximation can be different for different ϕ. This is inevitable, due to the fact that both the languages sub-CSL and LTL contain formulas of unbounded complexity.

To illustrate this point, consider an LMC model $\mathcal{M}$ for a sequence of coin tosses: the model has two states labeled H and T, respectively, and transition probabilities of 1/2 between all the states. Let $\mathcal{M}_N$ be a learned approximation of $\mathcal{M}$. The transition probabilities in $\mathcal{M}_N$ will deviate slightly from the true values 1/2. For example, assume that the transitions in $\mathcal{M}_N$ have value $1/2 + \delta$ for the transitions leading into H, and $1/2 - \delta$ for the transitions leading into T. Then one can construct LTL formulas ϕ, such that $|\,P_{\mathcal{M}}(\phi) - P_{\mathcal{M}_N}(\phi)\,|$ is arbitrarily close to 1. To do so, observe that according to $\mathcal{M}$ the relative frequency of the symbol H in long execution traces converges to 1/2, whereas according to $\mathcal{M}_N$ it converges to $1/2 + \delta$. For any $k > 0$ we can express with an LTL formula ϕ_k that the frequency of H in the first k steps is at least $1/2 + \delta/2$ by just enumerating all sequences of length k that satisfy this condition. Then, as $k \to \infty$, $P_{\mathcal{M}}(\phi_k) \to 0$ and $P_{\mathcal{M}_N}(\phi_k) \to 1$.

5 Experiments

In order to validate the proposed algorithm we have conducted two case studies on learning stochastic system models. Since a DMC can be seen as a DMDP having only a single input

action, we only report results for DMDPs and DCTMCs. For each case study, we generated observation sequences (I/O strings and timed strings) from known system models, and compared the generating models and the learned models based on relevant system properties expressed by PLTL formulas. All experiments were performed on a standard laptop with a 2.4GHz CPU.

5.1 Experiments with MDPs

For analyzing the behavior of the learning algorithm with respect to MDPs we consider a modified version of the slot machine model given by Jansen (2002). Our model represents a slot machine with three reels that are marked with two different symbols "bar" and "apple", as well as a separate initial symbol "blank". Starting with an initial configuration in which all reels show the "blank" symbol, the player can for a given number r of rounds select and spin one of the reels. A wheel that has been spun will randomly display either "bar" or "apple", where the probability of obtaining a "bar" is 0.7 in the first round, and gradually decreases as $0.7(r-k+1)/r$ for the kth round. The player receives a reward of 10 if the final configuration of the reels shows 3 bars, and a reward of 2 if the final configuration shows 2 bars. Instead of spinning a reel, the player can also choose to push a 'stop' button. In that case, with probability 0.5 the game will end, and the player receives the prize corresponding to the current configuration of the reels. With probability 0.5, the player will earn 2 extra rounds. Thus, choosing the 'stop' option can be beneficial when the current configuration already gives a reward (but at the risk that it will change into something less favorable when instead of terminating the game is extended by 2 rounds), or when with the remaining available rounds the current configuration is unlikely to change into a reward configuration (then at the risk that the game ends immediately with the current poor configuration).

This model is formalized as a DMDP whose states are defined by the configuration of the reels, the number of spins already performed *sp* (up to the maximum of r), and a Boolean *end* variable indicating whether the game is terminated. The granting of 2 extra spins is (approximately) implemented by decreasing by 2 the *sp* counter, down to a minimum of 0 (otherwise this would lead to an infinite state space). Input actions are $spin_i$ $(i = 1, 2, 3)$ and *stop*. The output alphabet is $\Sigma^{out} = \{blank, bar, apple\}^3 \cup \{Pr0,Pr2,Pr10,end\}$. States with $sp < r$ are labeled with the symbol from $\{blank, bar, apple\}^3$ representing the current reel configuration. When the number of available spins has been exhausted, then the next input (regardless of which input is chosen) leads to a state displaying the prize won as one of $\{Pr0,Pr2,Pr10\}$. Finally, one additional input leads to a terminal state labeled with *end*. States labeled with $\{Pr0,Pr2,Pr10\}$ have an associated reward of 0, 2, and 10, respectively. We have implemented this DMDP in PRISM (Kwiatkowska et al. 2011), and experimented with two versions of the model given by $r = 3$, and $r = 5$. These models have 103 ($r = 3$) and 161 ($r = 5$) reachable states, respectively.

The model generates traces that with probability 1 are finite, in the sense that after finitely many steps the trace ends in an infinite sequence of *end* symbols. However, there is no fixed bound on the number of initial non-*end* symbols. We sample observation sequences from the models using a uniform random selection of input actions at each point. Sampling of one sequence is terminated when the *end* symbol appears. The length distribution of strings sampled in this manner is dominated by a geometric distribution with parameter $\lambda = 0.25 \cdot 0.5$ (the probability that the random scheduler chooses the *stop* input, and the game terminates on that input). The convergence in probability (2) of Theorem 1 then also is ensured under this sampling regime (the consistency properties of the Hoeffding test in relation to the expected sample string lengths as described by Definitions 20 and 21 (iii) are unaffected when the

length distribution of sampled strings is reduced; the data support condition of Definition 21 (ii) still is true for all 'relevant' states of the IOFPTA, i.e., all states that are not just copies of the unique *end* state).

In the following, we characterize the size of data sets in terms of the total number N of observation symbols, rather than the number of sequences (as a better measure of the 'raw size' of the data). For sufficiently large samples, the ratio between the number of sequences and the number of symbols is very nearly constant, so that letting $\epsilon_N = c/N$ also satisfies the conditions to obtain (4) in Theorem 3 when N is the number of symbols. In our experiments we set $c = 10.000$, because that leads to $\epsilon_{20.000} = 0.5$ for our smallest datasize $N = 20.000$. Since the use of this ϵ_N sequence only is motivated by the theoretical convergence in the limit guarantees, and these guarantees do not provide any optimality guarantees for the limited sample sizes we consider, we also consider the alternative sequence where $\epsilon_N = 0.5$, for all N. This also serves the purpose of investigating the robustness of the learning results with respect to the choice of the ϵ_N.

We evaluate the learned models based on how well they approximate properties of the generating model. We consider properties of the form $P^{\max}(\phi)$ for different LTL formulas ϕ, and use the following accuracy measure for the evaluation: when p and $\bar{p}$ are the probabilities in the true and learned models, respectively, then we use the Kullback-Leibler distance

$$KL(p, \bar{p}) = p \log \frac{p}{\bar{p}} + (1 - p) \log \frac{1 - p}{1 - \bar{p}} \tag{5}$$

to measure the error of $\bar{p}$. The error, then, depends on the ratio $p/\bar{p}$ rather than the difference $p - \bar{p}$. The inclusion of the term $(1 - p) \log \frac{1-p}{1-\bar{p}}$ evaluates the estimate of $P^{\max}(\phi)$ also as an estimate for the dual $P^{\min}(\neg\phi) = 1 - P^{\max}(\phi)$. $KL(p, \bar{p})$ is infinite when $p \neq \bar{p} \in \{0, 1\}$, i.e. when the learned value $\bar{p}$ represents an incorrect assumption of deterministic behavior. On the other hand $\bar{p} \neq p \in \{0, 1\}$, i.e., incorrectly modeling deterministic behavior as probabilistic, incurs only a finite KL error. This asymmetric view is reasonable in many situations, because estimating 0,1-values by non-extreme probabilities usually means erring on the safe side, whereas incorrectly inferring 0,1-values can lead to incorrect assumptions of critical safety properties, for example.

We compare the models learned by IOALERGIA with the models given by the initially constructed I/O frequency prefix tree acceptors (with the frequencies normalized to probabilities, so that the IOFPTA is itself a valid DMDP). These initial tree-models are just a somewhat compact representation of the original data, and model checking performed on the trees can be seen as *statistical model checking* for DMDPs. Based on the tree-model representation of the data, we can use the model checking functionality of the PRISM tool to also perform statistical model checking. However, it turned out that the PRISM model checking algorithms, which are optimized for models specified in a modular, structured way, do not perform so well on the tree models, which are given by an unstructured state-level representation. Thus, even though PRISM is known to be able to operate on models of tens of millions of states, we were only able to run PRISM on tree models of up to around 60,000 states.

Figure 6 shows how for the $r = 3$ and $r = 5$ models the number of states in the constructed IOFPTAs and learned models develops as a function of the data size. The plots are in log-log scale, with the number of data symbols (divided by 1000) on the x-axis, and the number of states of the trees and learned models on the left, respectively right, y-axes. The red lines (box symbols) show a linear growth of the IOFPTA in log-log space. These lines have a near-perfect fit with the functions $550N^{0.65}$ ($r = 3$), and $550N^{0.8}$ ($r = 5$). These fits experimentally verify the sub-linear growth of IOFPTAs, which is theoretically obtained from Lemma 2 (Appendix).

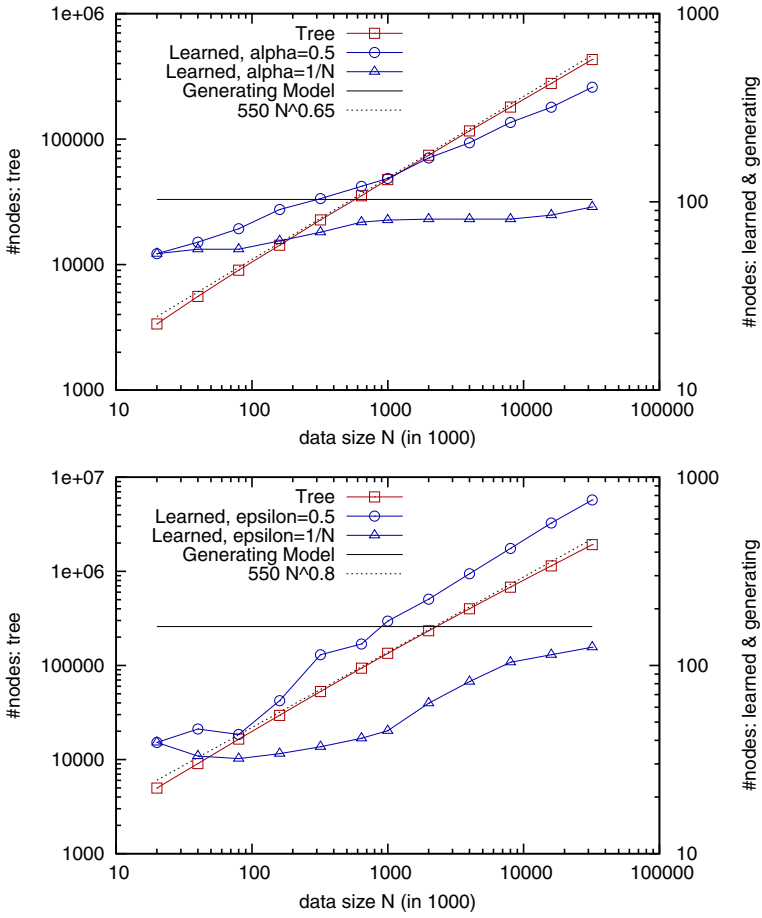

Fig. 6 Growth of tree and model size. *Top* $r = 3$, *bottom* $r = 5$

When learning with fixed $\epsilon = 0.5$, the learned model sizes also show an approximately linear behavior in log-log space, which translates to a growth of (approximately) the orders $N^{0.27}$ ($r = 3$), and $N^{0.4}$ ($r = 5$). Learning with $\epsilon_N = 10,000/N$ at first under-estimates the true model size. The models learned for the largest N values are very close in size to the generating model. However, the experimental range for N would need to be extended considerably further in order to ascertain that here we already see the asymptotic convergence to the true model.

We evaluate the accuracy of the learned model based on a test suite of 61 LTL properties. The complete list of properties is given in "MDP Test Properties" of Appendix. As mentioned above, using PRISM model checking on the IOFPTAs as a surrogate for statistical model checking does not scale to very large tree models. Therefore, the results here are limited to a maximum of $N = 1m$ for $r = 3$, and $N = 320k$ for $r = 5$ (at these tree sizes, a model-checking run for all 61 properties took several hours, vs. a few seconds for model checking the model learned from the IOFPTA).

We first consider for how many of the test properties an error $KL(p, \bar{p}) = \infty$ is obtained, i.e., the learned value $\bar{p}$ is an erroneous deterministic 0/1-value. These numbers are given in

Table 1 Number of test properties with $KL(p, \bar{p}) = \infty$

$r =$	20k	40k	80k	160k	320k	640k	1m
3	12;4	9;0	7;0	6;0	3;0	2;0	2;0
5	14;8	11;0	10;0	6;0	2;0	?;0	?;0

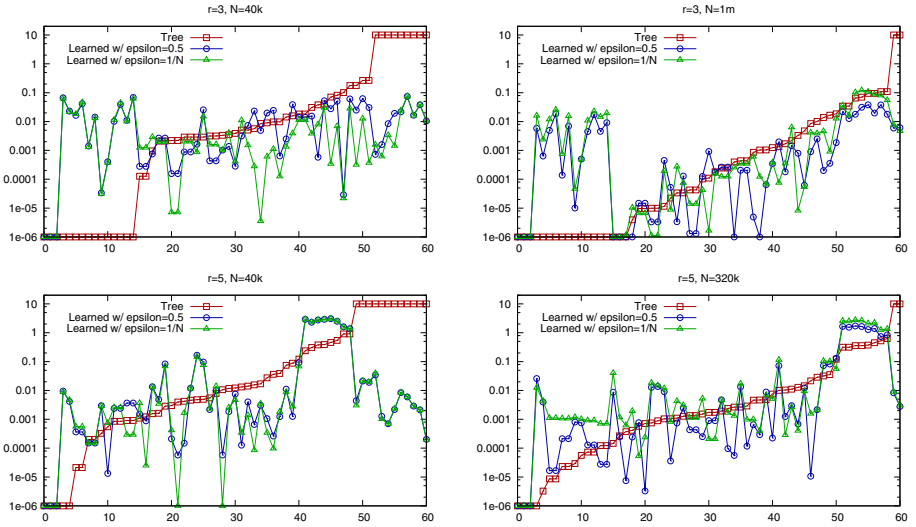

Fig. 7 KL errors

Table 1. An entry $k; l$ in this table means that for k test properties the IOFPTA gave an infinite KL-value, and for l properties this was the case for the learned model. It emerges a clear picture that the learned model is much less likely to return erroneous deterministic values. This is a natural consequence of a model-smoothing effect resulting from the state-merging process, and illustrates that model learning can alleviate overfitting problems occurring in statistical model-checking. The most problematic queries for IOFPTA-model checking were the low-probability queries 56–61, where the true probabilities are in the range 0.03–0.002, and IOFPTA-model checking returned the value 0. The values obtained from the learned models, on the other hand, approximated the true values rather well, and had KL-errors in the range 0.001–0.01.

The smoothing effect in the learned models can also have the less desirable consequence of leading to non-extreme estimates for probabilities that in the generating model are actually 0/1-valued. This was observed for property 16, which for $r = 5$ has max-probability 1 in the generating model. Here IOFPTA model checking returned the correct result, wheras the probabilities in the learned models were in the range 0.95–0.99 even for large data sizes. Similarly, some of the properties that have zero probability in the $r = 3$ model, had probabilities in the range 0.01–0.001 in the learned models.

Figure 7 illustrates the KL-errors for all 61 properties for small datasets ($N = 40k$), and the largest datasets for which model checking the IOFPTA tree was feasible ($N = 1m$ for $r = 3$, and $N = 320k$ for $r = 5$). In these plots the x-axes index the test properties. The properties are here sorted according to increasing values of the KL-errors obtained from the trees. Thus, the indexing differs from the numbering given in Table 5, and also the ordering of the properties differs in the four plots of Fig. 7. The y-axes show the KL-errors in log-scale. Infinite KL-values are represented by the value 10.0, and zero values by 10^{-6}.

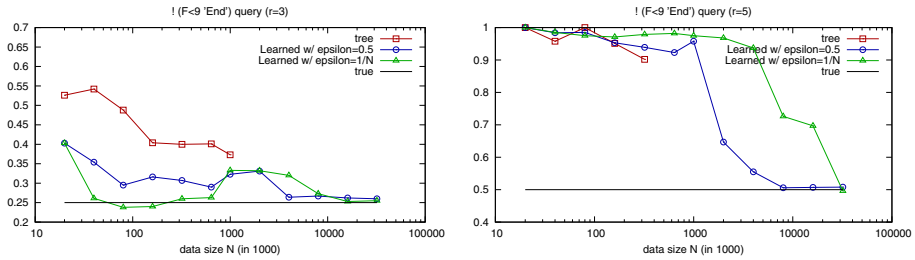

Fig. 8 Probabilities for $P^{\max}(\neg\Diamond^{<9} end)$ queries

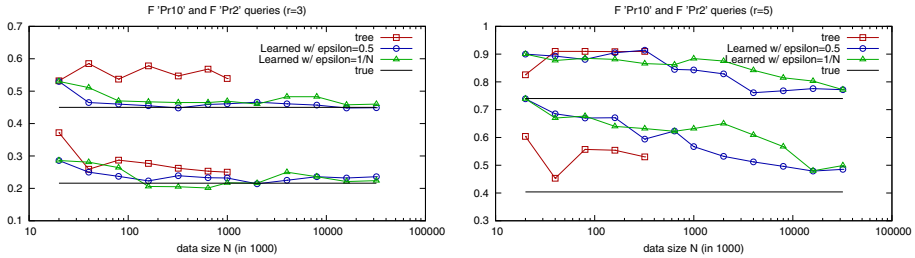

Fig. 9 Probabilities for $P^{\max}\Diamond Pr10$ and $P^{\max}\Diamond Pr2$ queries. *Plots* with the larger probability values are for $P^{\max}\Diamond Pr2$

At the right end of each plot appear the properties that gave $KL = \infty$ from IOFPTA model-checking. The errors obtained for the same properties from the learned models are in the same range as the errors for other properties. On the left ends of the plots appear properties with actual probability zero, which give zero error from the tree, but nonzero estimates, and hence nonzero errors from the learned models.

For the $r = 5$ model the properties appearing at indices 42–49 ($N = 40k$), respectively 52–59 ($N = 320k$) are properties 17–24 of Table 5, which are all of the form $P^{\max}(\neg\Diamond^{<k} end)$ for different values of k, i.e., they represent the maximum probability of the game lasting at least k steps, for various values of k. For both the tree and the learned models the estimates for these probabilities were quite inaccurate. Figure 8 on the right shows the actual probability values obtained for the $P^{\max}(\neg\Diamond^{<9} end)$ query for $r = 5$. For the datasizes $N = 40k$ and $N = 320k$ depicted in Fig. 7, the estimates are above 0.9 for all trees and models, whereas the true value is 0.5. The left plot in Fig. 8 shows the results for the same query in the $r = 3$ case.

Figure 9 shows the probabilities returned for the queries $P^{\max}\Diamond Pr10$ and $P^{\max}\Diamond Pr2$. These are queries for which the corresponding *KL*-errors lie in the middle ranges of the *KL*-errors seen in Fig. 7.

Figure 10 shows the average *KL*-errors obtained as a function of the data size. The average here is taken over all test properties excluding the properties $P^{\max}(\neg\Diamond^{<k} end)$ (whose high values would otherwise mask the development of *KL*-errors for the remaining properties). Furthermore, for each data size, only properties are included for which all models return non-infinite errors.

To obtain a more complete picture on the influence of the ϵ parameter, we also vary ϵ over the whole feasible range from 0 to 2 for the fixed data size $N = 10^6$. Figure 11 shows the sizes and average KL-errors for the learned models. The different ϵ-values we used are listed on the x-axis simply on equi-distant marks. The ϵ-values we otherwise used for $N = 10^6$

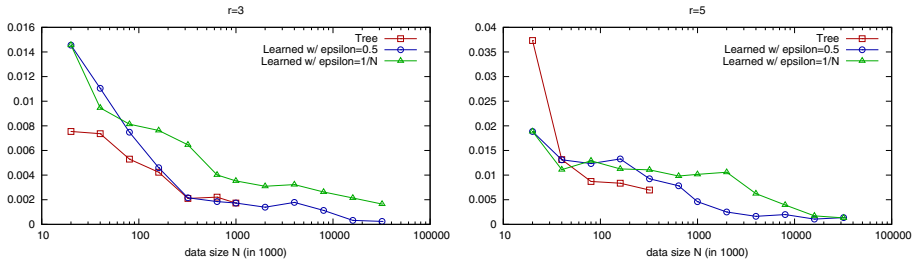

Fig. 10 Average KL errors

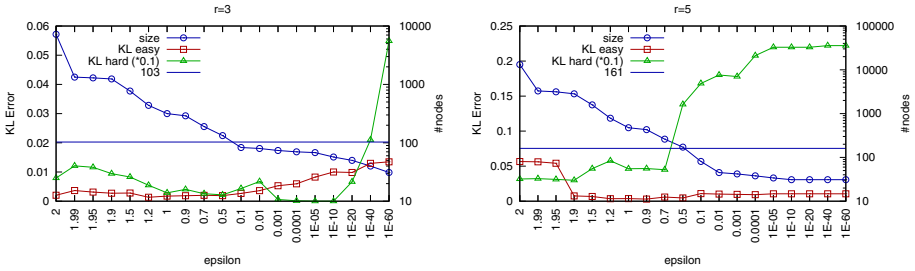

Fig. 11 Average KL error at $N = 10^6$ as function of ϵ

are 0.5 and 0.01, which both are in the middle of the range of values considered here. Even at the extreme end $\epsilon = 2$ the learned models are significantly smaller than the original IOFPTA's (which have sizes 47,564 and 134,693 for $r = 3$ and $r = 5$, respectively). This is because even though the Hoeffding test proper will always reject when $\epsilon = 2$, we still obtain positive compatibility results, and hence merges of nodes, due to the base test in line 1 of our Hoeffding compatibility test (Algorithm 3). The minimal model size is 31 nodes, corresponding to exactly one node for each output symbol. This minimal size is reached at $\epsilon = 10^{-60}$ and $\epsilon = 10^{-10}$ for $r = 3$ and $r = 5$, respectively. The average KL errors are shown in Fig. 11 separately for the "hard" test properties $P^{\max}(\neg\Diamond^{<k} end)$, and the remaining "easy" properties. Furthermore, to obtain readable plots, the KL-errors for the hard properties have been scaled by a factor of 0.1.

Figure 11 indicates that better results are obtained when ϵ is chosen so large that the size of the learned model is somewhat larger than the size of the true model. This would also have to be expected, since a model that over-estimates the true number of states can be trace-equivalent to the true model, whereas a model with fewer states than the true model usually can not. For the 'easy' test properties we obtain a fairly clear picture of optimal ϵ-values in the range 0.5–1.5, corresponding to models that are in the range of $1\times$ to $10\times$ the size of the true model. The picture for the 'hard' properties is less clear and rather different for $r = 3$, where the most accurate models are learned for a range of small ϵ-values, and $r = 5$, where the error decreases nearly monotonically as ϵ increases. Overall, the results show that IOALERGIA learning is quite robust with respect to the precise choice of the ϵ value.

Summarizing our observations, we can reach a number of conclusions: the differences in the accuracy of estimated probabilities are quite significant for different models of similar size ($r = 3$ with 103 states; $r = 5$ with 161 states), and for different queries $P^{\max}(\neg\Diamond^{<k} end)$ and $P^{\max}\Diamond PrX$ of similar syntactic form and complexity. Thus, neither the size of the true model, nor the complexity of the query alone will be good predictors for the accuracy of max-

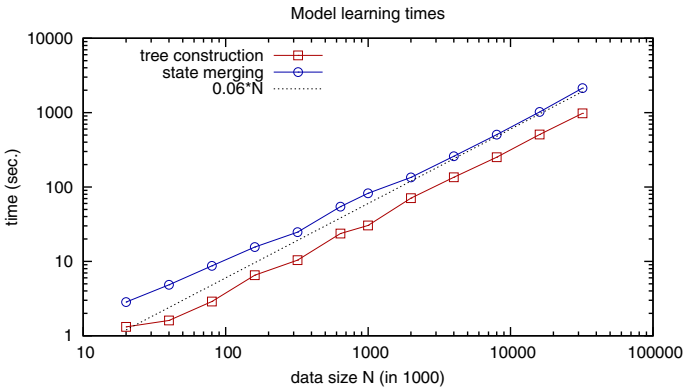

Fig. 12 Time for model learning ($r = 3$)

probability estimates obtained either by statistical model checking, or by model learning. In spite of very different convergence speeds, we observed convergence of the estimated max-probabilities to the true values for all test properties.

When comparing statistical model checking against model learning, no clear winner emerges in terms of the accuracy of estimated probabilities. The main difference lies in a smoothing effect of the learning process that eliminates extreme 0/1 empirical probabilities. This can allow the learned model to successfully generalize from the data, and return accurate estimates for low-probability properties that are not seen in the data, and for which statistical model checking returns zero probabilities. On the other hand, it can also lead to over-generalization, where true probability zero properties are given non-zero values in the learned model. Here it should be emphasized that in our experiments we have not tried to exploit another generalization capability of model learning, which is the ability to generalize from observations of finite initial trace segments to infinite behaviors. Traces in our slot machine model are finite with probability 1, and our data only contained traces of completed runs. This gives ideal conditions for statistical model checking, since empirical probabilities in the data correspond to actual model probabilities.

Comparing the results obtained from models learned with fixed $\epsilon = 0.5$, and decreasing $\epsilon = 10,000/N$ we observe in Figs. 8, 9 and 10 for smaller data sizes a slight advantage for $\epsilon = 0.5$. This is explained by Fig. 6, which shows that under the $\epsilon = 10,000/N$ regime the learned model stays smaller than the true model for the whole range of data sizes, approaching the true size only at the very end. The $\epsilon = 0.5$ models, on the other hand, soon become somewhat larger than the true model. As also indicated by Fig. 11, moderate over-approximations of the true model tend to lead to smaller KL errors.

In terms of space, model learning obviously leads to very significant savings (Fig. 6). As mentioned above, we cannot make a meaningful comparison for the time complexity of statistical model checking versus model learning, since we are using a very inefficient approach for performing the former. Figure 12 shows the computation time for IOALERGIAlearning for the case $r = 3$ and $\epsilon = 0.5$. The overall time is divided into the construction time for the IOFPTA, and the time for the IOALERGIAnode-merging process. We observe that both times are linear in the datasize. For ALERGIA, the theoretical worst-case complexity is cubic in the size of the IOFPTA, but the linear behavior we here observe is consistent with what is reported as the typical behavior of ALERGIA in practice. Moreover, we see that the times for the tree construction and the node merging phases of the learning procedure are of the same

Table 2 Accuracies of pure versus count-aggregating IOALERGIA($r = 3$)

Model	Size	Average KL-error
IOFPTA	47,564	0.012
$\epsilon = 0.5$	133	0.0048
$\epsilon = 0.5$, count aggregate	706	0.0062
$\epsilon = 0.01$	80	0.013
$\epsilon = 0.01$, count aggregate	161	0.0056

order of magnitude. Since even a highly optimized statistical model checking procedure will not be much faster than the IOFPTA construction, we can conclude that the time for model learning is of the same order of magnitude as a single run of statistical model checking, with significant savings for the amortized cost of checking multiple properties.

As discussed in Sect. 3.2, in our IOALERGIA implementation we do not aggregate frequency counts when merging nodes, and we perform the compatibility tests always based on the counts in the original IOFPTA. For comparison we also tested a version of the algorithm in which counts are aggregated. The main observation we made was that for a given ϵ-value, models learned using aggregated counts were larger than models learned without count aggregation. Thus, aggregating counts leads to more rejections in the compatibility tests. This can be explained by the fact that the Hoeffding test will always accept compatibility when the two counts n_1, n_2 are very small (cf. Algorithm 3), e.g. both are at most 2, or one is equal to 1, and the other less than 10. Since counts at the leaves of the IOFPTA (or nodes very close to the leaves) will usually have very low counts, this means that in the original IOFPTA most pairs of leaves will be tested as compatible. However, after merging the counts of two or three leaves, this will more often no longer be the case. The accuracy of models learned with count-aggregation was not higher than the accuracy of models learned without aggregation, but with ϵ-settings that lead to models of approximately equal size. Table 2 shows some detailed results for the $r = 3$ model learned from data of size $N = 1m$. For the two ϵ-values that also have been used in the previous experiments for $N = 1m$, the table shows the sizes of the learned models, with and without count aggregation. For comparison also the IOFPTA is included in the table. The average *KL*-error shown in the last column of the table is the average error over all 61 test properties (for $r = 3, N = 1m$ the errors for the difficult properties 17–24 are not such clear outliers that their inclusion in the average dominates the results). For the IOFPTA the *KL*-error is averaged over all properties except two for which the error is infinite. The table indicates that the accuracy depends more on the size of the learned model (best results being obtained when slightly over-estimating the true size) than on whether learning is with or without count aggregation.

5.2 Experiments for CTMCs

For CMTCs, we consider a case study adapted from Haverkort et al. (2000), where two sub-clusters of workstations are connected through a backbone. Each sub-cluster has N workstations, and the data from a workstation is sent to the backbone by a switch connected to the workstation's sub-cluster. The topology of the system is shown in Fig. 13. Each component in the system can break down and any broken component can be repaired. The average failure-free running time of the workstations, switches, and backbone is 2, 5, and 10 h, respectively; the average time required for repairing one of these components is 1, 2, and 4 h. There are two types of Quality of Service (QoS) associated with the system:

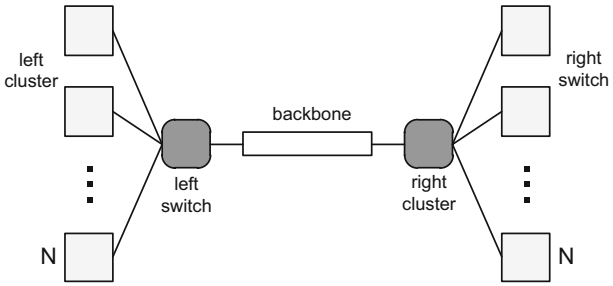

Fig. 13 The topology of a workstation cluster (Haverkort et al. 2000)

Table 3 Summary statistics of the CTMC models for the workstation cluster case study

N	4	8	10
$\|Q\|$	200	648	968
$\|Tran\|$	1240	4248	6424

– minimum: at least $3N/4$ workstations are operational and connected via switches and the backbone,
– premium: at least N workstations are operational and connected via switches and the backbone.

Note that if the premium requirement is met, then so is the minimum requirement. We specify CTMCs for this system with a varying number of workstations. The summary statistics for the models in terms of the number of states and transitions are listed in Table 3.

When generating data from the specified models, the observation sequences correspond to timed strings that alternate between observable symbols and time values. Following the sampling procedure outlined in Sect. 4, we generated observation sequences from different system configurations with 4, 8, and 10 workstations in each sub-cluster. The average length of these observation sequences is 50. We also assume that each component is operational initially. For the present case study, the most important property is the amount of time for which the minimum and premium QoS requirements are satisfied. These two properties are expressed by the sub-CSL formulas

$$P =?[\Diamond_{\leq t}\ !\text{“minimum}''] \qquad P =?[\Diamond_{\leq t}\ !\text{“premium}''],$$

where t is a real number.

For the experimental results reported below, we used $\alpha = 0.5$ for the compatibility tests employed in the learning algorithms. The choice of having a fixed α-value is based on the experimental results for the slot machine model (see Sect. 5.1), which showed that the learning algorithms are fairly robust wrt. the particular choice of α-value.

As shown in Fig. 14, the two QoS properties above are generally well approximated by the learned models although (as expected) the quality of the approximations decreases as the complexity of the generating models increases. All models are learned using 40000 symbols, and all probabilities have been computed using PRISM. For comparison, we have also included the results obtained by directly using the timed frequency prefix tree acceptors (TFPTAs) for performing model checking. As can be seen from the figure, when the prediction horizon starts to increase the properties are no longer well-approximated by the TFPTA-models. Summary information about the models learned for various data sizes and system

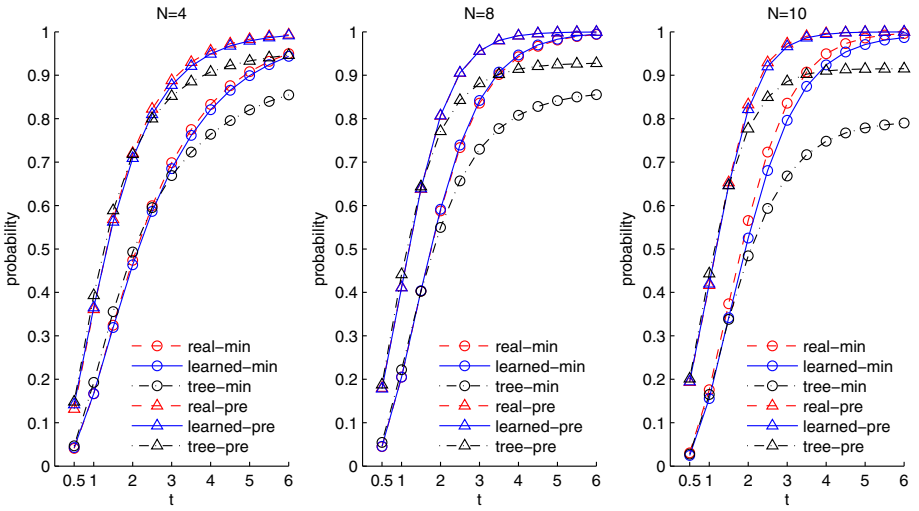

Fig. 14 The results of checking the properties $P =?[\Diamond_{\leq t}$!"minimum"] and $P =?[\Diamond_{\leq t}$!"premium"] in the learned models, timed frequency prefix tree acceptor, and the generating models with $t \in [0.5, 6]$

Table 4 Experimental results for the workstation cluster

$\lvert S\rvert(\times 10^3)$	|Seq|	|TFPTA|	Time	$\lvert Q\rvert$	D_A	D_A^T
$N = 4$						
1	15	478	0.78	125	0.0389	0.0612
4	45	1889	1.44	182	0.0294	0.0568
10	107	4659	2.67	197	0.0197	0.0447
40	402	18429	9.16	213	0.0149	0.0374
$N = 8$						
1	15	473	0.39	179	0.0398	0.0530
4	45	1878	1.31	333	0.0286	0.0700
10	107	4622	2.99	458	0.0202	0.0489
40	402	18208	11.72	578	0.0141	0.0454
$N = 10$						
1	6	493	1.00	196	0.1082	0.1786
4	30	1916	1.56	400	0.0789	0.1790
10	98	4632	4.09	584	0.0585	0.1575
40	406	18128	15.77	794	0.0531	0.1638

configurations are given in the first five columns in Table 4; $|S|$ is the number of symbols in the dataset ($\times 10^3$); |Seq| is the number of sequences in the dataset; |TFPTA| is the number of nodes in the TFPTA; *'Time'* is the learning time (in seconds), including the time for constructing the TFPTA, and $|Q|$ is the number of states in the learned model.

In addition to the two properties above, we have measured the quality of the learned models by randomly generating sets of sub-CSL formulas Φ using a stochastic context-free grammar. Each formula is restricted to a maximum length of 20. For the temporal operators we uniformly sample a time value t from $[0, 20]$ and defined the time intervals as $[0, t]$. In

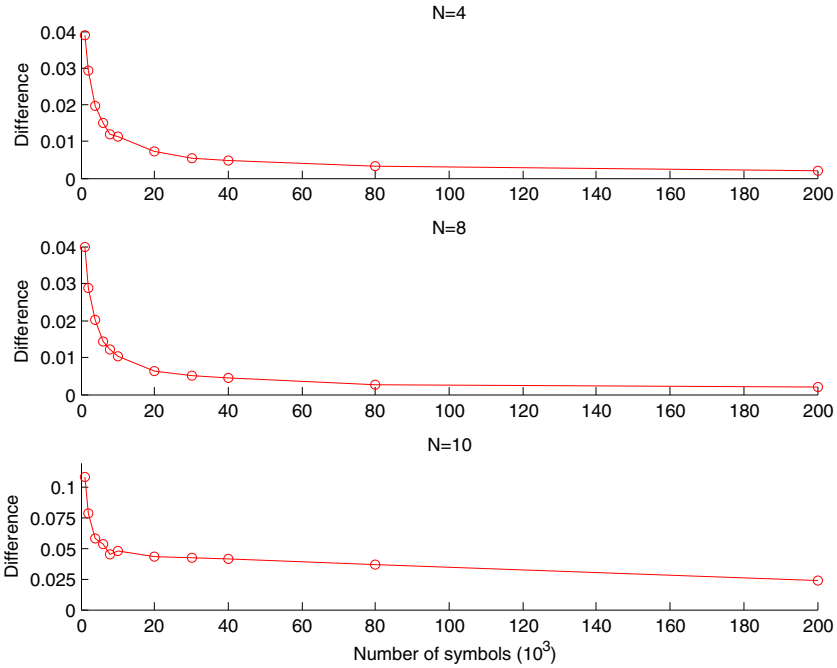

Fig. 15 The quality of learned models measured in terms of randomly generated formulas

order to avoid testing on tautologies or other formulas with little discriminative value, we constructed a baseline model B with one state for each symbol in the alphabet and with uniform transitions probabilities. For each generated formula $\varphi \in \Phi$ we then tested whether the formula was able to discriminate between the learned model A, the generating model M, and the baseline model B. If φ was not able to discriminate between these three models (i.e., $P_A(\varphi) = P_M(\varphi) = P_B(\varphi)$), then φ was removed from Φ.

We finally evaluated the learned models by comparing the mean absolute difference in probability (calculated using PRISM) over the generated formulas for the models M and A:

$$D_A = \frac{1}{|\Phi|} \sum\nolimits_{\varphi \in \Phi} |P_M(\varphi) - P_A(\varphi)|. \tag{6}$$

The mean absolute difference between M and B is calculated analogously.

The results of the experiments are listed in columns D_A and D_A^T in Table 4, where column D_A^T lists the results obtained by performing model checking using the TFPTA-model. For models with 4, 8, and 10 workstations in each sub-cluster we ended up with 677, 637, and 635 random formulas, respectively, after the elimination of non-discriminative formulas. The results are further illustrated in Fig. 15, where we also see that the difference (measured using the randomly generated formulas) between the learned model and the generating model decreases as the amount of data increases. Each data point is the mean value based on eight experiments with different randomly generated data sets. For comparison, the absolute mean difference between the baseline models and the generating models are 0.424, 0.350, and 0.293, for $N = 4$, $N = 8$, and $N = 10$, respectively.

6 Conclusion

In this paper we have proposed a framework for learning probabilistic system models based on observed system behaviors. Specifically, we have considered system models in the form of deterministic Markov decision processes and continuous time Markov chains, where the former model class includes standard deterministic Markov chains as a special case. The learning framework is presented within a model checking context and is based on an adapted version of the ALERGIA algorithm (Carrasco and Oncina 1994) for learning finite probabilistic automata models.

We have shown that in the large sample limit the learning algorithm will correctly identify the model structure as well as the probability parameters of the model. We position the learning results within a model checking context by showing that for the learned models the probabilities of model properties expressed in the formal specification languages LTL and sub-CSL will converge to the probabilities given by the true models.

The learning framework is empirically analyzed based on two use-cases covering Markov decision process and continuous time Markov chains. The use cases are analyzed wrt. the structure of the learned system models as well as relevant LTL and sub-CSL definable properties. The results show that for both model classes the learning algorithm is able to produce models that provide accurate estimates of the probabilities of the specified LTL and sub-CSL properties. The results have also been compared to the estimates obtained by statistical model checking, but with the analysis limited to properties testable by statistical model checking. Thus, we do not exploit the generalization capabilities of model learning for reasoning about unbounded system properties. The comparison shows that in terms of LTL-accuracy, there is no clear winner between the two approaches; the main differences in the results are caused by the smoothing effect of model learning. On the other hand, in terms of space and time complexity we see a significant difference in favor of model learning. For the sub-CSL properties, both the accuracy and complexity results are significantly better than those obtained by statistical model checking, in particular for sub-CSL properties defined over longer time horizons. These results are further complemented by accuracy estimates for randomly generated sub-CSL formulas, demonstrating that the learned models also provide accurate probability estimates of more general model properties.

The theoretical learning results presented in the paper focus on learning in the limit rather than on probably approximately correct (PAC) learning results. Extending the results to PAC learning would require an error measure for the model classes in question, which, in turn, would entail defining a suitable measure for probability distributions over Σ^ω. Candidate error measures have been investigated by Jaeger et al. (2014) who show that there are fundamental difficulties in defining measures that on the one hand support PAC learnability results and on the other hand satisfy natural continuity properties.

In addition to the results reported in the paper, we have conducted preliminary experiments on learning deterministic MDP approximations based on observations generated by non-deterministic system models. The results showed that the learned (deterministic) models are not sufficiently expressive to capture all relevant non-deterministic system properties. Based on these results, we wish as part of future work to consider learning methods for non-deterministic model classes. We expect, however, that the learning methods will be significantly different from the methods proposed in the current paper as, e.g., the assumption about a deterministic system behavior is key for the FPTA-based data representation.

The current paper is a significantly extended version of Mao et al. (2011) and Mao et al. (2012). We have subsequently adapted the results in Mao et al. (2012) to support active

learning scenarios, where one guides the interaction with the system under analysis in order to reduce the amount of data required for establishing an accurate system model (Chen and Nielsen 2012). Furthermore, the learning algorithm has also been extended for learning and verifying properties of systems endowed with a relational structure (Mao and Jaeger 2012). Generally, these learning results assume access to multiple observation sequences of the system in question. For systems that are hard (or even impossible) to restart, this requirement will rarely hold. In Chen et al. (2012) we have therefore considered methods for investigating system properties by learning system models based on a single observation sequence.

Appendix: Consistency of Alergia-style learning

Overview

In this appendix we give a detailed and general proof on the consistency of Alergia-like algorithms for learning finite stochastic automata. Our proof follows the same main lines of arguments as previous proofs presented in Carrasco and Oncina (1999), de la Higuera and Thollard (2000), Sen et al. (2004a). However, we extend and improve on these existing works in several ways.

First, we provide results that are formulated on the basis of a very general automaton model, and thereby provide a uniform treatment of consistency for the basic Alergia algorithm, as well as for extensions such as DLMDPs ad DCTMCs as introduced in Sect. 2. Also, the general *stochastic reactive automaton model* introduced below easily accommodates models both with non-zero termination probabilities, i.e., defining probability distributions on Σ^*, and models without termination probabilities, i.e., defining probability distributions on Σ^ω.

Second, in our proof we aim to make the statistical part of the argument more rigorous and self-contained: all previous consistency proofs – and also the one we propose in the following – depend on arguments about the error probabilities of the compatibility tests performed by the algorithm. The problem here is that the concrete tests performed depend on the structure of the specific FPTA, and thereby are dependent on the data. However, a test that would have a certain significance level if it was fixed prior to the observation of the data, may not have the same correctness guarantees if the fact that it is performed depends on the sampled data itself. A trivial example may illustrate the point: suppose we want to test the hypothesis that a coin is fair based on the empirical frequency $\bar{h}$ of heads in a sample of 100 tosses. For this we can find $p, q > 0$ with $p < q$ such that $P_{fair}(\bar{h} \notin [1/2 - p, 1/2 + q]) = P_{fair}(\bar{h} \notin [1/2 - q, 1/2 + p]) = 0.05$. Thus, reject if $\bar{h} \notin [1/2 - p, 1/2 + q]$ and reject if $\bar{h} \notin [1/2 - p, 1/2 + q]$ are both tests for the hypothesis $P(h) = 1/2$ at significance level 0.05. However, if we perform the first test whenever $\bar{h} \leq 1/2$, and the second if $\bar{h} > 1/2$, then the resulting test no longer has a 0.05 significance level. Of course, in Alergia, the execution of tests and the data sample are not connected in such an inadmissible way as in this example. In order to correctly account for this fact in the consistency proof, we largely separate the statistical argument from the concrete execution runs of the algorithm, and, in effect, always consider all the statistical tests that could be performed given some possible data sample.

The separation of the statistical from the algorithmic aspect also is part of the third goal of our consistency proof, which is to obtain a modular argument that clearly identifies three main components that lead to consistency:

- algorithmic component: conditions on the procedure by which nodes in the initial FPTA are tested for compatibility and merged. This will lead only to a very simple and loose constraint on the algorithmic procedure.
- data component: conditions on the sampling process for the data from which the automaton is learned.
- statistical component: conditions on the statistical tests used to decide node compatibility

This modular structure of the results facilitates their application to new or modified versions of existing algorithms.

Stochastic reactive automata model

We define a general stochastic automaton model of which LMDPs and CTMCs are special cases. We then also provide a general concept for deterministic stochastic automata, of which DLMDPs and DCTMCs are special cases. Our model is reactive, in the sense that it takes inputs, and its stochastic behavior is conditioned on those inputs. All probabilistic aspects of the automaton are encoded by random variables associated with each state.

Definition 12 A stochastic reactive finite automaton (SRFA)

$$\mathcal{A} = (Q, q^s, \boldsymbol{X}, \Sigma^{in}, succ, obs)$$

is given by

- A finite set of states Q containing a designated start state q^s.
- Each state $q \in Q$ is labeled with random variables $X_1^{(q)}, \ldots, X_n^{(q)}$, where each $X_i^{(q)}$ takes values in some sample space Ω_i (the same for all q), according to some parametric model Θ_i (the same for all q).
- A finite input alphabet Σ^{in}.
- A successor function $succ : Q \times \Sigma^{in} \times \prod_{i=1}^{n} \Omega_i \to Q$
- An observation function $obs : \Sigma^{in} \times \prod_{i=1}^{n} \Omega_i \to \mathcal{P}\{1, \ldots, n\}$.

We denote $\prod_{i=1}^{n} \Omega_i$ with $\boldsymbol{\Omega}$, and $(\omega_1, \ldots, \omega_n)$ with $\boldsymbol{\omega}$. $obs(\sigma, \boldsymbol{\omega})$ contains the indices of the random variables that are observed when the input is σ, and the $\boldsymbol{\omega}$ are the sampled values of $(X_1^{(q)}, \ldots, X_n^{(q)})$. We can then define the *observation space*

$$Obs := \{(\sigma, (\omega_i)_{i \in obs(\sigma, \boldsymbol{\omega})}) \mid \sigma \in \Sigma^{in}, \boldsymbol{\omega} \in \boldsymbol{\Omega})\}$$

Given an input string $\pi \in (\Sigma^{in})^{\omega}$, a SRFA defines a probability distribution over the space of state-observation sequences $(Q \times Obs)^{\omega}$ by assuming that the random variables $\boldsymbol{X}^{(q)}$ are independent at each state q, so that their joint distribution defines distributions for the successor state and the next observation.

By a slight abuse of notation, we also use $obs(\sigma, \boldsymbol{\omega})$ to denote $(\sigma, (\omega_i)_{i \in obs(\sigma, \boldsymbol{\omega})})$. We use $\boldsymbol{o}$ to denote elements of Obs.

Definition 13 A stochastic automaton is *finite-branching deterministic* (called DSRFA), if there exists an equivalence relation $\equiv$ on Obs, so that

- $\equiv$ partitions Obs into finitely many equivalence classes
- $obs(\sigma, \boldsymbol{\omega}) \equiv obs(\sigma', \boldsymbol{\omega}') \Rightarrow \forall q : succ(q, \sigma, \boldsymbol{\omega}) = succ(q, \sigma', \boldsymbol{\omega}')$

The equivalence class of $\boldsymbol{o} \in Obs$, is denoted $[\boldsymbol{o}]$, and $[Obs]$ is the set of all equivalence classes. In finite-branching deterministic automata we can also denote $succ(q, \sigma, \boldsymbol{\omega})$ as $succ(q, \boldsymbol{o})$, or $succ(q, [\boldsymbol{o}])$.

Example 5 We show how DLMDPs as described in Definition 3 can be represented as a DSRFA. We assume there also is an alphabet Σ^{out} of observable output symbols. On a given input $\sigma^i \in \Sigma^{in}$ the automaton makes a random transition to a state labeled with $\sigma^o \in \Sigma^{out}$, so that the successor state is uniquely determined by the (σ^i, σ^o) pair.

To represent this as a DSRFA, we assume that each state is labeled by random variables X_{σ^i} $(\sigma^i \in \Sigma^{in})$ with values in Σ^{out}. X_{σ^i} represents the conditional distribution over the next output symbol, given input σ^i. Given input $\sigma^i \in \Sigma^{in}$ one observes the value of the relevant variable X_{σ^i}, i.e., $obs(\sigma^i, (X_\sigma^{(q)})_{\sigma\in\Sigma^{in}}) = (\sigma^i, X_{\sigma^i}^{(q)})$. Then $succ(q, \sigma^i, (X_\sigma^{(q)})_{\sigma\in\Sigma^{in}})$ is the unique $q' \in Q$ defined by $q,\ \sigma^i,\ X_{\sigma^i}^{(q)}$. In this case, the equivalence class $[\boldsymbol{o}]$ is just the singleton $\boldsymbol{o}$.

To expand this to DCTMCs, one may add a real-valued delay variable X_T, e.g. with an exponential distribution. Assuming that the delay time is always observed, then $obs(\sigma^i, X_T, (X_\sigma^{(q)})_{\sigma\in\Sigma^{in}}) = (\sigma^i, X_T, X_{\sigma^i}^{(q)})$. Furthermore $(\sigma^i, X_T, X_{\sigma^i}^{(q)}) \equiv (\sigma^{i\prime}, X_T', X_{\sigma^i}^{(q)\prime})$ iff $\sigma^i = \sigma^{i\prime}$, and $X_{\sigma^i}^{(q)} = X_{\sigma^i}^{(q)\prime}$.

For random variables X, Y we write $X \approx Y$ if X and Y have the same distribution.

Definition 14 Two states $q, q' \in Q$ are said to be *locally compatible*, written $q \sim_l q'$, if $X_i^{(q)} \approx X_i^{(q')}$ for $i = 1, \ldots, n$. They are said to be *globally compatible*, written $q \sim q'$, if $q \sim_l q'$, and $succ(q, \bar{\boldsymbol{o}}) \sim_l succ(q', \bar{\boldsymbol{o}})$ for all $\bar{\boldsymbol{o}} \in Obs^*$.

The relation $q \sim q'$ is an equivalence relation on Q. The automaton obtained by factoring $\mathcal{A}$ over this equivalence relation is denoted $\mathcal{A}/\sim$.

Computation prefix tree

For a finite-branching deterministic automaton $\mathcal{A}$, we can define the computation prefix tree:

Definition 15 The computation prefix tree (CPT) for $\mathcal{A}$ is the infinite rooted tree in which every node v has one successor $succ(v, [\boldsymbol{o}])$ for each equivalence class $[\boldsymbol{o}] \in [Obs]$.

Each node v in the CPT can be labeled with a state $q(v) \in Q$: the root is labeled with q^s, and the $[\boldsymbol{o}]$-successor of a node labeled with q is labeled with $succ(q, \boldsymbol{o})$. For a finite computation (sequence of observations) $\bar{\boldsymbol{o}} = \boldsymbol{o}_1, \ldots, \boldsymbol{o}_k$ one inductively defines the node $v \in T$ *reached by* $\bar{\boldsymbol{o}}$: For $k = 0$ the node reached by $\bar{\boldsymbol{o}}$ is the root. For $k \geq 1$ the node reached by $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_k$ is the $[\boldsymbol{o}_k]$-successor of the node reached by $\boldsymbol{o}_1, \ldots, \boldsymbol{o}_{k-1}$. Each node v is reached by a unique observation sequence, denoted $\bar{\boldsymbol{o}}(v)$.

Without loss of generality, we from now on assume that all states in $\mathcal{A}$ are reachable by some computation, so that every state $q \in Q$ also appears as a node label in the CPT.

Definition 16 Let $\mathcal{A}$ be a DSFA with $|Q| = m$ and T its CPT. A *kernel* of T is any initial part K of T that contains for each state $q \in Q$ a node $v(q)$ labeled with q. If K is a kernel, then K^{+1} is the union of K with all $[\boldsymbol{o}]$-successors ($[\boldsymbol{o}] \in [Obs]$) of nodes in K. The *critical region* of K is the extension of K by the set of all nodes $v \in T$ reachable from K by a path of length at most m^2.

In most accounts of Alergia-like learning algorithms, it is assumed that an initial part of the CPT is constructed from the data. We take an essentially equivalent, but conceptually slightly different view, and let the data only increment empirical count variables at the nodes of the full, infinite tree. This, in particular, serves the purpose to consider sets of tests independently

from particular data samples, i.e., our analysis will be based on always considering all possible compatibility tests between nodes of the full CPT that would be performed given any sample.

Let v be a node in T. For each $i = 1, \ldots, n$ we associate with v an *empirical distribution variable* $\hat{X}_i^{(v)}$ whose values are multisets of values from Ω_i (for finite Ω_i such a multiset is just given by an integer count for each value in Ω_i).

For $j = 1, \ldots, N$ let $\bar{\boldsymbol{o}}^{(j)} = \boldsymbol{o}_1^{(j)}, \ldots, \boldsymbol{o}_{k(j)}^{(j)}$ be an observed computation of length $k(j)$. The sample $(\bar{\boldsymbol{o}}^{(j)})_j$ defines the empirical distributions at a node $v \in T$ that is reached by observation sequence $\bar{\boldsymbol{o}}(v)$ of length k as follows: the multiset $\hat{X}_i^{(v)}$ is the union of all ω_i that are observed in those $\boldsymbol{o}_{k+1}^{(j)}$ for j such that $k(j) \geq k+1$, and $\boldsymbol{o}_1^{(j)}, \ldots, \boldsymbol{o}_k^{(j)} = \bar{\boldsymbol{o}}(v)$.

State merging in the CPT

Alergia-like algorithms merge nodes of the CPT based on compatibility tests between pairs of nodes. The following definition introduces a binary relation representing the outcome of such tests.

Definition 17 A *compatibility test relation* on T is a binary symmetric and reflexive relation $\sim_t$ between the nodes of T. Furthermore, we define $v \sim_t^* v'$ iff $v \sim_t v'$, and for all $\bar{\boldsymbol{o}} \in Obs^*$: $succ(v, \bar{\boldsymbol{o}}) \sim_t succ(v', \bar{\boldsymbol{o}})$.

Based on recursively applied compatibility tests, Alergia-like algorithms actually compute the $\sim_t^*$ relation, and merge pairs of nodes v, v' (and their successors) for which $v \sim_t^* v'$. At each stage in the algorithm, an equivalence relation on T describes the equivalence classes of merged nodes. In the following, we define equivalence relations $\sim_i^{tc}$ that describe the equivalence classes of merged nodes after i iterations of the algorithm. For this we assume a fixed (but arbitrary) enumeration of the nodes of T:

$$T = v_1, v_2, v_3, \ldots$$

Definition 18 For $i = 1, 2, \ldots$ we define for $v, v' \in T$: $v \sim_i v'$ iff there exist $j, h \leq i$ and $\bar{\boldsymbol{o}} \in Obs^*$, such that $v_j \sim_t^* v_h$, $v = succ(v_j, \bar{\boldsymbol{o}})$, and $v' = succ(v_h, \bar{\boldsymbol{o}})$. Let $\sim_i^{tc}$ be the transitive closure of $\sim_i$.

The following lemma stipulates sufficient conditions on $\sim_t$ for the algorithm to terminate with the correctly identified equivalence classes of $\mathcal{A}/\sim$.

Lemma 1 *Let $k \geq 1$ be such that $K^{+1} = \{v_1, \ldots, v_k\}$ for some kernel K of T. Let C be the critical region for K. Assume that $\sim_t$ satisfies the following two conditions:*

(i) for all v, v' in C: $v \sim_t v'$ iff $q(v) \sim_l q(v')$ (correct test results on C)
(ii) for all $j, h \leq k$, and all $\bar{\boldsymbol{o}} \in \text{Obs}^*$: if $q(\text{succ}(v_j, \bar{\boldsymbol{o}})) \sim_l q(\text{succ}(v_h, \bar{\boldsymbol{o}}))$, then $\text{succ}(v_j, \bar{\boldsymbol{o}}) \sim_t \text{succ}(v_h, \bar{\boldsymbol{o}})$ (no false rejections in relevant tests)

Then for all $v, v' \in T$: $v \sim_k^{tc} v' \Leftrightarrow q(v) \sim q(v')$.

Proof First assume that $v \sim_k^{tc} v'$. It is sufficient to consider the case where $v \sim_k v'$: if in that case $q(v) \sim q(v')$, this will also be true in the general case $v \sim_k^{tc} v'$, since $\sim$ itself is a transitive relation.

Assume, then, that $v \sim_k v'$, and let $v_j, v_h \in K$ for v, v' as given by Definition 18. It is sufficient to show that $q(v_j) \sim q(v_h)$. Assume $q(v_j) \nsim q(v_h)$. Then $q(v_j) \not\sim_l q(v_h)$, or for some computation sequence $\bar{\boldsymbol{o}}$: $succ(q(v_j), \bar{\boldsymbol{o}}) \not\sim_l succ(q(v_h), \bar{\boldsymbol{o}})$. The length of $\bar{\boldsymbol{o}}$ can be

bounded by m^2, since any pair of states reachable from $q(v_j), q(v_h)$ is reachable within at most m^2 steps. Thus, $succ(v_j, \bar{\boldsymbol{o}}), succ(v_h, \bar{\boldsymbol{o}}) \in C$, and by (i), $succ(v_j, \bar{\boldsymbol{o}}) \not\sim_t succ(v_h, \bar{\boldsymbol{o}})$, so that $v_j \not\sim_t^* v_h$, a contradiction.

For the converse direction, we first note that the statement is true for $v, v' \in K^{+1}$, because then $q(v) \sim q(v')$ implies $v \sim_t^* v'$ by (i) and (ii), and therefore also $v \sim_k v'$.

For the general case we proceed as follows: we show that for every $v \in T$ there exists $v_K \in K$ with $v \sim_k^{tc} v_K$. Then, for $v, v' \in T$ with $q(v) \sim q(v')$ we obtain $v \sim_k^{tc} v_K$, $v' \sim_k^{tc} v'_K$. By the first part of the proof, then $q(v) \sim q(v_K)$, and $q(v') \sim q(v'_K)$, and hence also $q(v_K) \sim q(v'_K)$, and $v \sim_k^{tc} v'$.

Assume that there exists $v \in T \setminus K^{+1}$ for which no v_K exists. Let v be such a counterexample that is minimal in the sense that $v = succ(v_0, \boldsymbol{o})$, $v_0 \in K^{+1} \setminus K$, and $|\boldsymbol{o}|$ (i.e., the distance of v to K^{+1}) is minimal. For v_0 there exists $v_1 \in K$ with $q(v_0) \sim q(v_1)$, and therefore $v_0 \sim_t^* v_1$. Let $v' = succ(v_1, \boldsymbol{o})$. Then $v \sim_k v'$. The distance of v' to K^{+1} is less than $|\boldsymbol{o}|$, and therefore $v' \sim_k^{tc} v_K$ for some $v_K \in K$. Thus, also $v \sim_k^{tc} v_K$, a contradiction. □

Definition 18 reflects a quite high-level description of an iterative state-merging procedure that abstracts from several implementation details present in our version of the Alergia algorithm as described in Sect. 3. For instance, the definition of $\sim_i$ does not take into account that in our algorithm we test the compatibility of a node q_b (corresponding to the next node v_i considered in the enumeration of T according to Definition 18) with candidate nodes q_r (corresponding to the nodes $v_1, \ldots, v_{i-1}$ in our enumeration) in lexicographic order of q_r, and that once one such compatibility is found, no further compatibilities of q_b with other nodes q_r are tested. Due to these differences between the procedural merge strategies in concrete implementations, and the abstract merge relations $\sim_i$, it is not the case that in all cases the final equivalence classes over states computed by the algorithm coincide with the limit of $\sim_i^{tc}$ as $i \to \infty$. However, these two equivalence relations will be the same if condition (i) of Lemma 1 holds: in that case, the test relation $\sim_t$ is guaranteed to be an equivalence relation on C, and implementation details that influence which representatives of an equivalence class are used for compatibility testing do not affect the outcome.

The only necessary procedural aspect we have to require of an implementation in order to guarantee that under the conditions of Lemma 1 the computed equivalence relation coincides with $\sim_k^{tc}$ is that nodes of the CPT are processed in a fixed order, which is not influenced by the data sample.

We will now investigate conditions under which it is ensured that $\sim_t$ will satisfy the conditions of Lemma 1 if $\sim_t$ is defined by statistical tests of the relation $\sim_l$. This will be a purely statistical question without any reference to algorithmic procedures.

Statistical tests

We assume that the relation $\sim_t$ is defined by statistical tests $\sim_{t,i}$ for the local equivalences $X_i^{(q(v))} \approx X_i^{(q(v'))}$ as $\sim_t = \cap_{i=1}^n \sim_{t,i}$.

According to the terminology and notation introduced in Definition 12, a random variable X has a distribution on a state space Ω characterized by a parameter $\theta \in \Theta$. In the following, we denote this distribution by P_θ. By a slight abuse of notation, we also use P_θ to denote the distributions induced on Ω^N ($N \geq 1$) and Ω^∞ by independent random sampling from P_θ. Furthermore, $P_{\theta_1 \times \theta_2}$ denotes the sampling distribution for two independent samples according to P_{θ_1} and P_{θ_2}, respectively. For an infinite sample sequence $\boldsymbol{\omega} \in \Omega^\infty$ we denote by $\boldsymbol{\omega}(N)$ the initial sequence of N samples.

Definition 19 A *two-sample test* for equivalence for the parametric family $\{P_\theta \mid \theta \in \Theta\}$ is a mapping

$$\mathbb{R}^{>0} \times \bigcup_{N\in\mathbb{N}} \Omega^N \times \bigcup_{N\in\mathbb{N}} \Omega^N \to \{accept, reject\}$$

such that for all $\theta \in \Theta$:

$$P_{\theta\times\theta}(\{(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2) \in \Omega^{N_1} \times \Omega^{N_2} \mid T(\epsilon, \boldsymbol{\omega}_1, \boldsymbol{\omega}_2) = reject\}) < \epsilon. \tag{7}$$

Furthermore, we require that for all $\epsilon > 0$

$$T(\epsilon, \boldsymbol{\omega}_1, \boldsymbol{\omega}_2) = accept \tag{8}$$

if $|\boldsymbol{\omega}_1| = 0$ or $|\boldsymbol{\omega}_2| = 0$.

In the following we use $\Omega(f(N))$ and $O(f(N))$ in the usual complexity-theoretic sense to denote the classes of functions that grow at least, respectively at most, as fast as $f(N)$. Note, in particular, that Ω now appears with two distinct meanings: as a function class, and as a sample space.

Definition 20 Let $h : \mathbb{N} \to \mathbb{R}$ be non-decreasing. A two-sample test T is *strongly h-consistent*, if there exists a sequence $(\epsilon_N)_N$ with

(i-a) $\sum_N h(N)\epsilon_N < \infty$
(ii-a) for all $\theta_1, \theta_2 \in \Theta, \theta_1 \neq \theta_2$, and for all $g_1, g_2 \in \Omega(N)$:

$$P_{\theta_1\times\theta_2}(\{(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2) \in \Omega^\infty \times \Omega^\infty \mid T(\epsilon_N, \boldsymbol{\omega}_1(g_1(N)), \boldsymbol{\omega}_2(g_2(N))) = accept \text{ for infinitely many } N\}) = 0. \tag{9}$$

T is called *weakly h-consistent*, if instead of (i-a) and (ii-a) only the following holds

(i-b) for all $\delta > 0$, there exists $N_0 \in \mathbb{N}$, such that for all $N \geq N_0$: $h(N)\epsilon_N \leq \delta$.
(ii-b) for all $\theta_1, \theta_2 \in \Theta, \theta_1 \neq \theta_2$, for all $g_1, g_2 \in \Omega(N)$:

$$\lim_N P_{\theta_1\times\theta_2}(\{(\boldsymbol{\omega}_1, \boldsymbol{\omega}_2) \in \Omega^\infty \times \Omega^\infty \mid T(\epsilon_N, \boldsymbol{\omega}_1(g_1(N)), \boldsymbol{\omega}_2(g_2(N))) = accept) = 0.$$

The following definitions introduces the conditions we have to impose on data generation procedures to ensure consistency.

Definition 21 Let $\bar{\boldsymbol{o}}^\infty = \bar{\boldsymbol{o}}^{(1)}, \bar{\boldsymbol{o}}^{(2)}, \ldots, \bar{\boldsymbol{o}}^{(N)}, \ldots$ be an infinite sequence of finite observation sequences, where each $\bar{\boldsymbol{o}}^{(N)}$ is independently sampled from some sampling distribution P^s_N.[1] We denote with P^s the sampling distribution for $\bar{\boldsymbol{o}}^\infty$, and with $\hat{X}^{(v)}_{i,N}$ the empirical distribution defined as in Sect. 1 from the first N elements of $\bar{\boldsymbol{o}}^\infty$. Let $h : \mathbb{N} \to \mathbb{R}$ as in Definition 20. We say that P^s is *h-admissible* if

(i) for all v, N, i: the elements of $\hat{X}^{(v)}_{i,N}$ are an iid sample from $P(X_i^{q(v)})$.
(ii) for all v, i: $P^s(|\hat{X}^{(v)}_{i,N}| = \Omega(N)) = 1$ (at least linear increase of sample sizes for all empirical node distributions)
(iii) $E(|\{v \mid \exists i : |\hat{X}^{(v)}_{i,N}| > 0\}|) = O(h(N))$ (in expectation, the increase of the number of nodes with non-empty samples is at most $h(N)$).

[1] Note that the $\bar{\boldsymbol{o}}^{(N)}$ can not be assumed to be identically distributed, since the input sequences will be different in different samples.

Theorem 4 *Let P^s be an h-admissible sample distribution. For $i = 1, \ldots, n$ let T_i be an h-consistent two-sample test for equivalence for the parametric family $\{P_\theta \mid \theta \in \Theta_i\}$ with associated sequence $(\epsilon_{i,N})$.*

For $N > 1$, $i \in \{1, \ldots, n\}$ we define

$$v \sim_{t,i}^{(N)} v' \ :\Leftrightarrow\ T(\epsilon_{i,N}, \hat{X}_{i,N}^{(v)}, \hat{X}_{i,N}^{(v')}) = \text{accept} \tag{10}$$

Furthermore, define

$$v \sim_{t}^{(N)} v' \ :\Leftrightarrow\ \forall i : \ v \sim_{t,i}^{(N)} v'.$$

If T is strongly h-consistent, then

$$P^s(\sim_t^{(N)} \text{ almost always satisfies (i) and (ii) in Lemma 1}) = 1. \tag{11}$$

If T is weakly h-consistent, then for all $\delta > 0$ exists $N_0 \in \mathbb{N}$, such that for all $N \geq N_0$:

$$P^s(\sim_t^{(N)} \text{ satisfies (i) and (ii) in Lemma 1}) \geq 1 - \delta. \tag{12}$$

Proof We first observe that we may assume that all T_i satisfy Definition 20 with the same sequence ϵ_N, because replacing $\epsilon_{i,N}$ with $\epsilon_N := \max_i \epsilon_{i,N}$ preserves the validity of conditions (i) and (ii) in Definition 20.

We now first show that $\sim_t^{(N)}$ a.a. satisfies (i). Let $v, v' \in C$, and assume, first, that $q(v) \sim_l q(v')$. Then $\theta_i = \theta_i'$ for all i, and thus

$$P^s(T(\epsilon_N, \hat{X}_{i,N}^{(v)}, \hat{X}_{i,N}^{(v')}) = \textit{reject}) \leq \epsilon_N.$$

If (i-a) holds, then by the Borel-Cantelli lemma (here only using $\sum_N \epsilon_N < \infty$)

$$P^s(v \sim_{t,i}^{(N)} v' \text{ a.a.}) = 1$$

for all i, and therefore also

$$P^s(v \sim_{t}^{(N)} v' \text{ a.a.}) = 1. \tag{13}$$

If (i-b) holds, then for a given δ and sufficiently large N:

$$P^s(v \sim_{t,i}^{(N)} v') \geq 1 - \delta/(4n \mid C \mid^2),$$

and therefore

$$P^s(\text{for all } v, v' \in C : \ q(v) \sim_l q(v') \Rightarrow v \sim_t v') > 1 - \delta/4. \tag{14}$$

Conversely, assume $q(v) \not\sim_l q(v')$. Let $i \in \{1, \ldots, n\}$ be such that $\theta_i \neq \theta_i'$. Assume that T is strongly consistent. Then, by Definition 20 (ii-a) and Definition 21 (ii):

$$P^s(T(\epsilon_N, \hat{X}_{i,N}^{(v)}, \hat{X}_{i,N}^{(v')}) = \textit{reject} \text{ a.a.}) = 1,$$

and therefore

$$P^s(v \not\sim_t^{(N)} v' \ a.a.) = 1. \tag{15}$$

(13) and (15) together imply that $P^s(\sim_t^{(N)}$ almost always satisfies (i) in Lemma 1$) = 1$.

If T is only weakly consistent, then by Definition 20 (ii-b) and Definition 21 (ii), for all δ and all sufficiently large N:

$$P^s(T(\epsilon_N, \hat{X}_{i,N}^{(v)}, \hat{X}_{i,N}^{(v')}) = \textit{reject}) \leq \delta/(4n \mid C \mid^2),$$

and therefore

$$P^s(\text{for all } v, v' \in C : q(v) \sim_l q(v') \Leftarrow v \sim_t v') > 1 - \delta/4. \tag{16}$$

(14) and (16) together imply that $P^s(\sim_t^{(N)}$ satisfies (i) in Lemma 1) $\geq 1 - \delta/2$.

We now turn to showing condition (ii) of Lemma 1. For this we consider the probability that (ii) does not hold for $\sim_t^{(N)}$. In the following, when we write a union or summation over pairs v, v' this is always shorthand for union or summation over the set

$$\{v, v' \mid q(v) \sim_l q(v'), \exists v_j, v_h \in K, \bar{\boldsymbol{o}} \in Obs^* : v = succ(v_j, \bar{\boldsymbol{o}}), v' = succ(v_h, \bar{\boldsymbol{o}})\}$$

Using (7) and (8) we can write:

$$\begin{aligned}
P^s(\cup_{v,v'}\{v \not\sim_t^{(N)} v'\}) &\leq \sum_{v,v'}\sum_{i=1}^{n} P^s(v \not\sim_{t,i}^{(N)} v') \\
&= \sum_{v,v'}\sum_{i=1}^{n} P^s(v \not\sim_{t,i}^{(N)} v', |\hat{X}_{i,N}^{(v)}| > 0) \\
&= \sum_{v,v'}\sum_{i=1}^{n} P^s(v \not\sim_{t,i}^{(N)} v' \mid |\hat{X}_{i,N}^{(v)}| > 0) P^s(|\hat{X}_{i,N}^{(v)}| > 0) \\
&\leq \epsilon_N \sum_{v,v'}\sum_{i=1}^{n} P^s(|\hat{X}_{i,N}^{(v)}| > 0)
\end{aligned} \tag{17}$$

For any given v there exist at most $|K|$ different v' for which the pair v, v' is included in the sum. Also writing $P^s(|\hat{X}_{i,N}^{(v)}| > 0\})$ as $E(\mathbf{1}_{|\hat{X}_{i,N}^{(v)}|>0})$ with $\mathbf{1}_e$ the indicator function of event e, we can therefore further bound (17):

$$\begin{aligned}
&\leq \epsilon_N |K| \sum_{v\in T}\sum_{i=1}^{n} P^s(|\hat{X}_{i,N}^{(v)}| > 0) \\
&= \epsilon_N |K| \sum_{v\in T}\sum_{i=1}^{n} E(\mathbf{1}_{|\hat{X}_{i,N}^{(v)}|>0}) = \epsilon_N |K| E\left(\sum_{v\in T}\sum_{i=1}^{n} \mathbf{1}_{|\hat{X}_{i,N}^{(v)}|>0}\right) \\
&\leq \epsilon_N |K| n E\left(\sum_{v\in T} \mathbf{1}_{\exists i:|\hat{X}_{i,N}^{(v)}|>0}\right) = O(h(N)\epsilon_N),
\end{aligned} \tag{18}$$

where the last equality is due to Definition 21 (iii).

If Definition 20 (i-a) holds, it follows with the Borel-Cantelli Lemma that

$$P^s(\sim_t^{(N)} \text{ infinitely often violates Lemma 1 (ii)}) = 0.$$

□

If Definition 20 (i-b) holds, then for sufficiently large N

$$P^s(\sim_t^{(N)} \text{ violates Lemma 1 (ii)}) \leq \delta/2. \tag{19}$$

We conclude this section by showing that the Hoeffding test for the equivalence of binomial distributions, and the F-test for the equivalence for exponential distributions are strongly and weakly h-consistent, respectively, for

$$h_\lambda^{geo}(N) := E(|\{v \mid \exists i : |\hat{X}_{i,N}^{(v)}| > 0\}|),$$

where the expectation is with respect to the sampling procedure described in Sect. 4, i.e. with a geometric distribution with parameter λ for the length of the sample sequences $\bar{\boldsymbol{o}}^{(j)}$.

Lemma 2 $\lim_N h_\lambda^{geo}(N)/N = 0$

Proof Let $V_N := |\{v \mid \exists i : |\hat{X}_{i,N}^{(v)}| > 0\}|$ and $V_N^+ := V_N - V_{N-1}$, i.e., V_N^+ is the number of nodes $v \in T$ that are reached for the first time in the Nth sample. Then $E(V_N) = \sum_{k=1}^N E(V_k^+)$, and the lemma can be proven by showing that $E(V_k^+) \to 0$ as $k \to \infty$. We can write

$$E(V_k^+) = E(V_k^+ \mid V_k^+ > 0) P^s(V_k^+ > 0).$$

For all k: $E(V_k^+ \mid V_k^+ > 0) = (1-\lambda)/\lambda$. This is because the geometric distribution represents a memoryless sampling procedure for the length of an observation sequence $\bar{\boldsymbol{o}}$, so that conditional on $\bar{\boldsymbol{o}}$ having reached a first new node v, the expected length of the remaining string is still the prior expectation $(1-\lambda)/\lambda$. It is thus sufficient to show that $P^s(V_k^+ > 0) \to 0$ for $k \to \infty$. For this let $A_{l,k}$ be the event that all nodes $v \in T$ at depth $\leq l$ are included in V_k. Then, because of Definition 21 (ii), we have that for all fixed l: $P^s(A_{l,k}) \to 1$ for $k \to \infty$. Thus, for all l and all $\delta > 0$ there exists k_0 such that for all $k \geq k_0$: $P^s(V_k^+ > 0) \leq P^s(V_k^+ > 0 \mid A_{l,k}) + \delta$. With $P^s(V_k^+ > 0 \mid A_{l,k}) \leq (1-\lambda)^l$ then $P^s(V_k^+ > 0) \to 0$ follows. □

Lemma 3 *The Hoeffding test defined by Algorithm 3 is strongly h_λ^{geo}-consistent.*

Proof We first note that the Hoeffding test is indeed a two-sample test in the sense of Definition 19 (Carrasco and Oncina 1999). To show strong consistency, let $\epsilon_N := 1/N^r$ for some $r > 2$. Then (i-a) of Definition 20 is satisfied, because according to Lemma 2 $h_\lambda^{geo}(N)\epsilon_N < 1/N^{r-1}$ in the limit $N \to \infty$.

To show (ii-a), let $\theta_1 > \theta_2$ be parameters of the binomial distribution, and $g_1, g_2 \in \Omega(N)$. In this case, $\boldsymbol{\omega}_i(g_i(N))$ are samples from $\Omega = \{0, 1\}$ of size $g_i(N)$. Let f_i denote the number of occurrences of 1 in $\boldsymbol{\omega}_i(g_i(N))$, and $T(\epsilon_N, \boldsymbol{\omega}_1(g_1(N)), \boldsymbol{\omega}_2(g_2(N))) = \mathit{accept}$ iff

$$|f_1/g_1(N) - f_2/g_2(N)| < (\sqrt{1/g_1(N)} + \sqrt{1/g_2(N)})\sqrt{1/2 \ln(2/\epsilon_N)}. \tag{20}$$

By the strong law of large numbers, $P_{\theta_1 \times \theta_2}(\lim_{N\to\infty} |f_1/g_1(N) - f_2/g_2(N)| \to \theta_1 - \theta_2) = 1$. The right-hand side of (20) is of the order $O(\sqrt{\ln N/N})$, and, thus, goes to zero as $N \to \infty$. It follows that with probability 1, (20) only holds for finitely many N.

We note that similarly we obtain that the Hoeffding test is weakly h_λ^{geo}-consistent for sequences $\epsilon_N := 1/N^r$ with $r > 1$. □

Lemma 4 *The F-test defined by Algorithm 4 is weakly h_λ^{geo}-consistent.*

Proof For the F-test, the data $\boldsymbol{\omega}_i(g_i(N))$ consists of samples from $\Omega = \mathbb{R}$ following exponential distributions with parameters θ_i. Let $g_1, g_2 \in \Omega(N)$. In the following, we denote $N_i := g_i(N)$ $(i = 1, 2)$.

Let $\hat{\theta}_i := \sum_{l=1}^{N_i} \omega_l / N_i$. Then $(\hat{\theta}_1/\hat{\theta}_2)(\theta_2/\theta_1)$ (approximately) follows an $F(2N_1, 2N_2)$-distribution with mean $\mu = \frac{N_2}{N_2 - 1}$ and standard deviation

$$\sigma = \sqrt{\frac{N_2^2(N_1 + N_2 - 1)}{N_1(N_2 - 1)^2(N_2 - 2)}} \tag{21}$$

(Cox 1953; Gehan and Thomas 1969), and

$$P_{\theta_1\times\theta_2}\left(\hat{\theta}_1/\hat{\theta}_2 \in \left[\mu - \frac{\sigma}{\sqrt{\epsilon_N}}, \mu + \frac{\sigma}{\sqrt{\epsilon_N}}\right]\right)$$
$$= P_{F(2N_1,2N_2)}\left(\left[(\mu - \frac{\sigma}{\sqrt{\epsilon_N}})\frac{\theta_1}{\theta_2}, (\mu + \frac{\sigma}{\sqrt{\epsilon_N}})\frac{\theta_1}{\theta_2}\right]\right). \quad (22)$$

The F-test is constructed by an application of Chebyshev's inequality for the $F(2N_1, 2N_2)$-distribution, and thereby is seen to be a two-sample test in the sense of Definition 19.

To show weak consistency, let $\epsilon_N = 1/\sqrt{N h_\lambda^{geo}(N)}$. With Lemma 2 then $h_\lambda^{geo}(N)\epsilon_N = \sqrt{h_\lambda^{geo}(N)/N} \to 0$, so that Definition 20 (i-b) is satisfied.

Now assume $\theta_1 \neq \theta_2$. With Lemma 2 we obtain $\sigma/\sqrt{\epsilon_N} = O((h_\lambda^{geo}(N)/N)^{1/4}) \to 0$. With $\mu \to 1$ this means that the interval $[(\mu - \frac{\sigma}{\sqrt{\epsilon_N}})\frac{\theta_2}{\theta_1}, (\mu + \frac{\sigma}{\sqrt{\epsilon_N}})\frac{\theta_2}{\theta_1}]$ is bounded away from 1 as $N \to \infty$, and that the right-hand side of (22) goes to zero. □

Appendix 2: MDP test properties

See Table 5.

Table 5 LTL test properties used in the Experiments of Sect. 5.1

Query		Answer	
		$r = 3$	$r = 5$
1.	Pmax=? [F "Pr10"]	2.16$E-$001	4.04$E-$001
2.	Pmax=? [F "Pr2"]	4.48$E-$001	7.40$E-$001
3.	Pmax=? [F "Pr0"]	1.00	1.00
4.	Pmax=? [X (X ("100"))]	5.33$E-$001	4.58$E-$001
5.	Pmax=? [X (X ("200"))]	4.90$E-$001	5.60$E-$001
6.	Pmax=? [X (X ("110"))]	1.60$E-$001	1.32$E-$001
7.	Pmax=? [X (X ("120"))]	3.73$E-$001	3.08$E-$001
8.	Pmax=? [X (X ("220"))]	3.27$E-$001	3.92$E-$001
9.	Pmax=? [X (X (X ("111")))]	1.23$E-$001	7.66$E-$002
10.	Pmax=? [X (X (X ("122")))]	2.50$E-$001	2.27$E-$001
11.	Pmax=? [X (X (X ("112")))]	2.86$E-$001	1.79$E-$001
12.	Pmax=? [X (X (X ("222")))]	7.62$E-$002	1.65$E-$001
13.	Pmax=? [X (X (X ("Pr0")))]	5.00$E-$001	5.00$E-$001
14.	Pmax=? [X (X (X ("Pr2")))]	0.00	0.00
15.	Pmax=? [X (X (X ("Pr10")))]	0.00	0.00
16.	Pmax=? [! (F<7 ("End"))]	5.00$E-$001	1.00
17.	Pmax=? [! (F<8 ("End"))]	2.50$E-$001	5.00$E-$001
18.	Pmax=? [! (F<9 ("End"))]	2.50$E-$001	5.00$E-$001
19.	Pmax=? [! (F<10 ("End"))]	1.25$E-$001	2.50$E-$001
20.	Pmax=? [! (F<11 ("End"))]	1.25$E-$001	2.50$E-$001
21.	Pmax=? [! (F<12 ("End"))]	6.25$E-$002	1.25$E-$001

Table 5 continued

Query		Answer	
		$r = 3$	$r = 5$
22.	Pmax=? [! (F<13 (“End”))]	$6.25E{-}002$	$1.25E{-}001$
23.	Pmax=? [! (F<14 (“End”))]	$3.13E{-}002$	$6.25E{-}002$
24.	Pmax=? [! (F<15 (“End”))]	$3.13E{-}002$	$6.25E{-}002$
25.	Pmax=? [X (X (“100”)) & (F “Pr10”)]	$5.25E{-}002$	$9.18E{-}002$
26.	Pmax=? [X (X (“100”)) & (F “Pr2”)]	$1.28E{-}001$	$1.92E{-}001$
27.	Pmax=? [X (X (“100”)) & (F “Pr0”)]	$5.33E{-}001$	$4.58E{-}001$
28.	Pmax=? [X (X (“200”)) & (F “Pr10”)]	$2.02E{-}001$	$3.05E{-}001$
29.	Pmax=? [X (X (“200”)) & (F “Pr2”)]	$3.22E{-}001$	$4.43E{-}001$
30.	Pmax=? [X (X (“200”)) & (F “Pr0”)]	$4.90E{-}001$	$5.60E{-}001$
31.	Pmax=? [X (X (“110”)) & (F “Pr10”)]	$8.63E{-}003$	$1.33E{-}002$
32.	Pmax=? [X (X (“110”)) & (F “Pr2”)]	$2.10E{-}002$	$3.42E{-}002$
33.	Pmax=? [X (X (“110”)) & (F “Pr0”)]	$1.60E{-}001$	$1.32E{-}001$
34.	Pmax=? [X (X (“210”)) & (F “Pr10”)]	$4.89E{-}002$	$7.97E{-}002$
35.	Pmax=? [X (X (“210”)) & (F “Pr2”)]	$1.19E{-}001$	$2.08E{-}001$
36.	Pmax=? [X (X (“210”)) & (F “Pr0”)]	$3.73E{-}001$	$3.08E{-}001$
37.	Pmax=? [X (X (“220”)) & (F “Pr10”)]	$1.04E{-}001$	$2.64E{-}001$
38.	Pmax=? [X (X (“220”)) & (F “Pr2”)]	$2.50E{-}001$	$3.86E{-}001$
39.	Pmax=? [X (X (“220”)) & (F “Pr0”)]	$3.27E{-}001$	$3.92E{-}001$
40.	Pmax=? [X (X (X (“222”))) & (F “Pr10”)]	$7.62E{-}002$	$1.65E{-}001$
41.	Pmax=? [X (X (X (“222”))) & (F “Pr2”)]	0.00	$1.58E{-}001$
42.	Pmax=? [X (X (X (“222”))) & (F “Pr0”)]	0.00	$1.19E{-}001$
43.	Pmax=? [X (X (X (“221”))) & (F “Pr10”)]	0.00	$9.95E{-}002$
44.	Pmax=? [X (X (X (“221”))) & (F “Pr2”)]	$2.50E{-}001$	$2.27E{-}001$
45.	Pmax=? [X (X (X (“221”))) & (F “Pr0”)]	0.00	$2.18E{-}001$
46.	Pmax=? [X (X (X (“211”))) & (F “Pr10”)]	0.00	$2.31E{-}002$
47.	Pmax=? [X (X (X (“211”))) & (F “Pr2”)]	0.00	$7.82E{-}002$
48.	Pmax=? [X (X (X (“211”))) & (F “Pr0”)]	$2.86E{-}001$	$1.79E{-}001$
49.	Pmax=? [X (X (X (“111”))) & (F “Pr10”)]	0.00	$3.87E{-}003$
50.	Pmax=? [X (X (X (“111”))) & (F “Pr2”)]	0.00	$9.91E{-}003$
51.	Pmax=? [X (X (X (“111”))) & (F “Pr0”)]	$1.23E{-}001$	$7.66E{-}002$
52.	Pmax=? [(X (X (X (“221”)))) & (X (X (X (X (“222”)))))]	0.00	$6.37E{-}002$
53.	Pmax=? [(X (X (X (“221”)))) & (X (X (X (X (“121”)))))]	0.00	$1.64E{-}001$
54.	Pmax=? [(X (X (X (“211”)))) & (X (X (X (X (“221”)))))]	0.00	$5.00E{-}002$
55.	Pmax=? [(X (X (X (“211”)))) & (X (X (X (X (“211”)))))]	0.00	$1.29E{-}001$
56.	Pmax=? [X (X (“110”)) & (X (X (X (“112”)))) & (F “Pr0”)]	$3.73E{-}002$	$6.14E{-}002$
57.	Pmax=? [X (X (“110”)) & (X (X (X (“111”)))) & (F “Pr0”)]	$4.27E{-}002$	$9.50E{-}002$
58.	Pmax=? [X (X (“110”)) & (X (X (X (“112”)))) & (F “Pr2”)]	$1.19E{-}002$	$2.49E{-}002$
59.	Pmax=? [X (X (“110”)) & (X (X (X (“111”)))) & (F “Pr2”)]	$5.59E{-}003$	$1.25E{-}002$
60.	Pmax=? [X (X (“110”)) & (X (X (X (“112”)))) & (F “Pr10”)]	$4.89E{-}003$	$9.57E{-}003$
61.	Pmax=? [X (X (“110”)) & (X (X (X (“111”)))) & (F “Pr10”)]	$2.30E{-}003$	$4.88E{-}003$

The properties are given in PRISM syntax. For conciseness, *blank*, *apple*, *bar* are represented by 0,1,2, respectively

References

Aarts, F., & Vaandrager, F. W. (2010). Learning I/O automata. In *Proceedings of the international conference on concurrency theory* (CONCUR 2010), pp. 71–85.

Ammons, G., Bodík, R., & Larus, J. R. (2002). Mining specifications. In *Proceedings of the SIGPLAN-SIGACT symposium on principles of programming language* (POPL 2002), pp. 4–16.

Angluin, D. (1987). Learning regular sets from queries and counterexamples. *Journal of Information and Computation*, *75*, 87–106.

Baier, C., & Katoen, J. P. (2008). *Principles of model checking*. Cambridge, MA: The MIT Press.

Baier, C., Haverkort, B., Hermanns, H., & Katoen, J. P. (2003). Model-checking algorithms for continuous-time Markov chains. *Journal of IEEE Transaction on Software Engineering*, *29*(6), 524–541.

Behrmann, G., David, A., Larsen, K. G., Pettersson, P., & Yi, W. (2011). Developing uppaal over 15 years. *Journal of Software: Practice and Experience*, *41*(2), 133–142.

Bollig, B., Katoen, J. P., Kern, C., & Leucker, M. (2010). Learning communicating automata from MSCs. *IEEE Transactions on Software Engineering*, *36*(3), 390–408.

Bouyer, P., Larsen, K. G., & Markey, N. (2008). Model checking one-clock priced timed automata. *Journal of Logical Methods in Computer Science*, *4*(2), 1–28.

Bouyer, P., Fahrenberg, U., Larsen, K. G., & Markey, N. (2011). Quantitative analysis of real-time systems using priced timed automata. *Communications of the ACM*, *54*(9), 78–87.

Carrasco, R., & Oncina, J. (1994). Learning stochastic regular grammars by means of a state merging method. In *Proceedings of the international colloquium on grammatical inference and applications* (ICGIA 1994), pp. 139–152.

Carrasco, R. C., & Oncina, J. (1999). Learning deterministic regular grammars from stochastic samples in polynomial time. *Journal of Theoretial Informatics and Applications*, *33*(1), 1–20.

Castro, J., & Gavaldà, R. (2008). Towards feasible PAC-learning of probabilistic deterministic finite automata. In *Grammatical inference: Algorithms and applications*, pp. 163–174.

Chen, T., Han, T., Katoen, J. P., & Mereacre, A. (2009). Quantitative model checking of continuous-time Markov chains against timed automata specifications. In *24th annual IEEE symposium on logic in computer science* pp. 309–318.

Chen, Y., & Nielsen, T. D. (2012). Active learning of Markov decision processes for system verification. In *Proceedings of the international conference on machine learning and applications* (ICMLA 2012), pp. 289–294.

Chen, Y., Mao, H., Jaeger, M., Nielsen, T. D., Larsen, K.G., & Nielsen, B. (2012). Learning Markov models for stationary system behaviors. In *NASA formal methods symposium* (NFM), pp. 216–230.

Clark, A., & Thollard, F. (2004). PAC-learnability of probabilistic deterministic finite state automata. *Journal of Machine Learning Research*, *5*, 473–497.

Cobleigh, J. M., Giannakopoulou, D., & Pasareanu, C. S. (2003). Learning assumptions for compositional verification. In *Proceedings of the 9th international conference on tools and algorithms for the construction and analysis of systems* (TACAS), pp. 331–346.

Courcoubetis, C., & Yannakakis, M. (1995). The complexity of probabilistic verification. *Journal of the ACM*, *42*(4), 857–907.

Cox, D. R. (1953). Some simple approximate tests for Poisson variates. *Biometrika*, *40*(3/4), 354–360.

de Higuera, C., & Oncina, J. (2004). Learning stochastic finite automata. In *Proceedings of the international conference on grammatical inference*, pp. 175–186.

de la Higuera, C., & Thollard, F. (2000). Identification in the limit with probability one of stochastic deterministic finite automata. In *Proceedings of the international colloquium on grammatical inference: Algorithms and application* (ICGI 2000), pp. 141–156.

Desharnais, J., Gupta, V., Jagadeesan, R., & Panangaden, P. (1999). Metrics for labeled Markov systems. In *Proceedings of international conference on concurrency theory* (CONCUR), pp. 258–273.

Feng, L., Han, T., Kwiatkowska, M. Z., & Parker, D. (2011). Learning-based compositional verification for synchronous probabilistic systems. In *9th international symposium on automated technology for verification and analysis* (ATVA), pp. 511–521.

Gehan, E. A., & Thomas, D. G. (1969). The performance of some two-sample tests in small samples with and without censoring. *Biometrika*, *56*(1), 127–132.

Giannakopoulou, D., & Păsăreanu, C. S. (2005). Learning-based assume-guarantee verification (Tool Paper). In P. Godefroid (Ed.), *Model Checking Software: 12th International SPIN Workshop* (pp. 282–287). Berlin, Heidelberg: Springer.

Grinchtein, O., Jonsson, B., & Leucker, M. (2005). Inference of timed transition systems. *Journal of Electronic Notes in Theoretical Compututer Science*, *138*(3), 87–99.

Grinchtein, O., Jonsson, B., & Pettersson, P. (2006). Inference of event-recording automata using timed decision trees. In *Proceedings of the international conference on concurrency theory* (CONCUR), pp. 435–449.

Haverkort, B. R., Hermanns, H., & Katoen, J. P. (2000). On the use of model checking techniques for dependability evaluation. In *Proceedings of the IEEE symposium on reliable distributed systems* (SRDS 2000), pp. 228–237.

Hérault, T., Lassaigne, R., Magniette, F., & Peyronnet, S. (2004). Approximate probabilistic model checking. In Steffen, B., Levi, G. (Eds.), *Verification, model checking, and abstract interpretation*. Lecture Notes in Computer Science, Vol. 2937, Springer, Berlin, pp. 307–329.

Higuera, Cd. (2010). *Grammatical inference: Learning automata and grammars*. Cambridge: Cambridge University Press.

Jaeger, M., Mao, H., Larsen, K. G., & Mardare, R. (2014). Continuity properties of distances for Markov processes. In *Proceedings of QEST 2014*, LNCS, Vol. 8657, pp. 297–312.

Jansen, D. N. (2002). Probabilistic UML statecharts for specification and verification a case study. In *Proceedings of the workshop on critical systems development with UML*, pp. 121–132.

Komuravelli, A., Pasareanu, C. S., & Clarke, E. M. (2012). Learning probabilistic systems from tree samples. In *Proceedings of the 27th annual IEEE/ACM symposium on logic in computer science*, pp. 441–450.

Kwiatkowska, M.Z., Norman, G., & Parker, D. (2011). Prism 4.0: Verification of probabilistic real-time systems. In *Proceedings of the international conference on computer aided verification* (CAV'11), pp. 585–591.

Laroussinie, F., Larsen, K. G., & Weise, C. (1995). From timed automata to logic- and back. In *Proceedings of international symposim on mathematical foundations of computer science* (MFCS 1995), pp. 529–539.

Legay, A., Delahaye, B., & Bensalem, S. (2010). Statistical model checking: An overview. In *Proceedings of the first international conference on runtime verification*, Springer, Berlin, RV'10, pp. 122–135.

Leucker, M. (2007). Learning meets verification. In *Proceedings of the international conference on formal methods for components and objects* (FMCO 2007), pp. 127–151.

Mao, H., & Jaeger, M. (2012). Learning and model checking networks of I/O automata. In *Proceedings of the fourth Asian conference on machine learning* (ACML), pp. 285–300.

Mao, H., Chen, Y., Jaeger, M., Nielsen, T. D., Larsen, K. G., & Nielsen, B. (2011). Learning probabilistic automata for model checking. In *Proceedings of the international conference on quantitative evaluation of system* (QEST 2011), pp. 111–120.

Mao, H., Chen, Y., Jaeger, M., Nielsen, T. D., Larsen, K. G., & Nielsen, B. (2012). Learning Markov decision processes for model checking. In *Proceedings of the first workshop on quantities in formal methods* (QFM), pp. 49–63.

Niese, O. (2003). An integrated approach to testing complex systems. PhD thesis, Universität Dortmund.

Oncina, J., Garcia, P., & Vidal, E. (1993). Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis Machine Intelligence*, *15*(5), 448–458.

Pnueli, A. (1977). The temporal logic of programs. In *Proceedings of the annual symposium on foundations of computer science* (FOCS) pp. 46–57.

Rabin, M. O. (1963). Probabilistic automata. *Information and Control, 6*(3), 230–245. doi:10.1016/S0019-9958(63)90290-0. http://www.sciencedirect.com/science/article/pii/S0019995863902900

Raffelt, H., & Steffen, B. (2006). Learnlib: A library for automata learning and experimentation. In *Proceedings of the international conference on fundamental approaches to software engineering* (FASE), pp. 377–380.

Ron, D., Singer, Y., & Tishby, N. (1996). The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, *25*(2–3), 117–149.

Ron, D., Singer, Y., & Tishby, N. (1998). On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences*, *56*(2), 133–152.

Segala, R. (1996). *Modeling and verification of randomized distributed real-time systems*. Technical report. Cambridge, MA.

Sen, K., Viswanathan, M., & Agha, G. (2004a). Learning continuous time Markov chains from sample executions. In *Proceedings of international conference on quantitative evaluation of systems* (QEST), pp. 146–155.

Sen, K., Viswanathan, M., & Agha, G. (2004b). Statistical model checking of black-box probabilistic systems. In Alur, R., Peled, D. (Eds.), *Computer aided verification*. Lecture Notes in Computer Science, Vol. 3114, pp. 202–215.

Singh, R., Giannakopoulou, D., & Pasareanu, C. S. (2010). Learningcomponent interfaces with may and must abstractions. In *Computer aided verification*. Lecture Notes in Computer Science, Vol. 3576, pp. 527–542.

Thollard, F., Dupont, P., & de la Higuera, C. (2000). Probabilistic DFA inference using kullback-leibler divergence and minimality. In *Proceedings of the international conference on machine learning* (ICML), pp. 975–982.

Tzeng, W. G. (1992). Learning probabilistic automata and markov chains via queries. *Machine Learning*, *8*, 151–166.

van Breugel, F., & Worrell, J. (2005). A behavioural pseudometric for probabilistic transition system. *Theoretical Computer Science*, *331*, 115–142.

Vardi, M. Y. (1985). Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the IEEE symposium on foundations of computer science* (FOCS), pp. 327–338.

Vardi, M. Y. (1999). Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *Proceedings of the international AMAST workshop on formal methods for real-time and probabilstic systems* (ARTS 1999), pp. 265–276.

Verwer, S. (2010). Efficient identification of timed automata—Theory and practice. PhD thesis, Technical University Delft.