Learning Probabilistic Automata for Model Checking

Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen
Aalborg University
Dept. of Computer Science
Email: [huamao,ykchen,jaeger,tdn,kgl,bnielsen] @cs.aau.dk

Abstract—**Obtaining accurate system models for verification is a hard and time consuming process, which is seen by industry as a hindrance to adopt otherwise powerful model-driven development techniques and tools. In this paper we pursue an alternative approach where an accurate high-level model can be automatically constructed from observations of a given black-box embedded system. We adapt algorithms for learning finite probabilistic automata from observed system behaviors. We prove that in the limit of large sample sizes the learned model will be an accurate representation of the data-generating system. In particular, in the large sample limit, the learned model and the original system will define the same probabilities for linear temporal logic (LTL) properties. Thus, we can perform PLTL model-checking on the learned model to infer properties of the system. We perform experiments learning models from system observations at different levels of abstraction. The experimental results show the learned models provide very good approximations for relevant properties of the original system.**

I. Introduction

To enable the development of complex embedded software systems industry and researchers are increasingly focusing on a model-driven development approach (MDD). Once a model capturing the behavioral requirements for the system (or component) has been constructed, it can be used for a variety of computer aided development tasks like unambiguous documentation, simulation, model-checking, performance evaluation and prediction, parameter optimization, controller synthesis, test generation, and, in restricted cases, even code generation.

However, constructing accurate models of industrial systems is hard and time consuming, and is seen by industry as a hindrance to adopt otherwise powerful MDD techniques and tools. Especially, the necessary accurate, updated, and detailed documentation rarely exist for legacy software or 3rd party components. We therefore seek an experimental approach where an accurate high-level model can be automatically constructed or *learned* from observations of a given black-box embedded system component. Given a learned and explicitly represented model, it can be formally analyzed (e.g. by model-checking) in the context of other existing component models.

For deterministic system models it has been suggested to use Angluin's [1] approach to learning deterministic finite automata [4], [11], [12]. For complex systems that are only partially observable via their interactions with the user, it can be quite unrealistic to assume that an adequate deterministic model exists. For example, an embedded system in a mobile device may react differently to identical input sequences when operated in different physical environments. Data obtained by observing the behavior of such a system could not be used in Angluin-style learning, where it is assumed that data is noise-free, which, in this case, means that each finite input sequence is uniquely labeled as either accepted or rejected by the system.

To avoid these difficulties, we are interested in learning probabilistic system models. Probabilistic models provide the ability to construct accurate quantitative models of a system's observable (and possibly non-deterministic) behavior. The data we require for learning only consists of previously observed system behaviors. It is not assumed that the system for which a model is to be constructed is available for testing or interactive data generation in the learning process (however, our techniques might be refined by *active learning* techniques that take advantage of such interactive data acquisition).

In this paper we show that methods for learning probabilistic finite automata [5], [6], [8] can be adapted for the task of learning Markov Chain system models for verification.

In related work, Sen et al. [13] previously also adapted the algorithm from [5] for learning Markov Chain models for verification. Our work differs from [13] in that we use a different modification of the original algorithm, which leads to somewhat stronger consistency results for learning in the limit, and that we analyse theoretically and experimentally how the convergence of the learned model relates to the convergence of probability estimates for system properties expressed in linear time temporal logic (LTL).

Statistical model-checking [18], [14] can also be used to check (probabilistic) properties of a system. Statistical model-checking uses hypothesis testing based on sample runs of a system (obtained from system traces or using Monte-Carlo model simulation) that allows the engineer to check to a desired level of confidence whether a given logical property holds with a given (minimum) probability. Our ambition goes beyond statistical model checking in that we extract an explicit probabilistic model to support verification (including probability estimation) of a large class of properties without the potential costly re-sampling of the system, as well as being able to use the model for other MDD tasks.

Learning supports model construction at different levels of abstraction since it takes place relative to a set of observations that the learner can make of the system under test. These observations may be based on input and output actions, or may be based on states by evaluating a set of state predicates. By selecting different actions or predicates, different learned models reflect different views of the underlying system. Thus, tailoring the observations to the properties of interest in the succeeding verification tasks will make verification easier and more efficient. Further, for such abstractions we find it more informative to preserve probabilistic information about system choices, rather than representing them as a purely deterministic or non-deterministic model.

In this paper, we thus make the following main contributions: we propose a modified version of the ALERGIA learning algorithm for finite stochastic automata, establish the consistency of the algorithm for learning in the large-sample limit, and we provide a theoretical and empirical analysis of the consistency and accuracy of model-checking LTL properties in the learned model.

This paper is structured as follows: Section III describes the AALERGIA algorithm for learning Deterministic Labeled Markov Chains. Section IV presents theoretical convergence guarantees of the algorithm, and Section V demonstrates empirically that model-checking our learned system models provide accurate and fast approximations to model-checking the original system.

II. PRELIMINARIES

A. Strings

We use Σ to denote a (finite) alphabet, Σ^* and Σ^ω to denote the set of all finite, respectively infinite strings over Σ. The empty string is denoted e. For a set S of strings, $\text{prefix}(S)$ denotes the set of all prefixes of strings $s \in S$. We assume an ordering on Σ that induces the lexicographic ordering on Σ^*.

B. Markov System Models

A *Labeled Markov chain (LMC)* is a tuple $M = \langle Q^M, \Sigma, \pi^M, \tau^M, L^M \rangle$, where

- Q^M is a finite set of states,
- $\pi^M : Q^M \to [0,1]$ is an *initial probability distribution* such that $\sum_{q \in Q^M} \pi^M(q) = 1$,
- $\tau^M : Q^M \times Q^M \to [0,1]$ is the *transition probability function* s.t. for all $q \in Q^M$, $\sum_{q' \in Q^M} \tau^M(q, q') = 1$.
- $L^M : Q^M \to \Sigma$ is a *labeling function*

Labeling functions that assign to states a subset of atomic propositions *AP* are accommodated in our framework by defining $\Sigma = 2^{AP}$.

A labeled Markov chain is *deterministic (DLMC)*, if

- There exists a state $start^M \in Q^M$ with $\pi^M(start^M) = 1$
- For all $q \in Q^M$ and $\sigma \in \Sigma$: there exists at most one $q' \in Q^M$ with $L^M(q') = \sigma$ and $\tau^M(q, q') > 0$. We then also write $\tau^M(q, \sigma)$ instead of $\tau^M(q, q')$.

When there is no risk of ambiguity, we usually omit the superscript M when denoting components of M. A (D)LMC defines a probability distribution P_M on Σ^ω according to standard definitions (see e.g. [3, Section 10.1]). For $q \in Q^M$ we denote by $P_{M,q}$ the distribution obtained by (re-)defining q as the unique *start* state.

In a DLMC there is a tight connection between strings and states: given an observed string s with $P_M(s) > 0$ there is a unique state q that the Markov Chain must be in. Conversely, every state q is associated with the set *strings*(q) of all strings that lead from the start state to q. We therefore use symbols q for states and s for strings to some extent interchangeably: s can also denote the state in a DLMC reached by the string s. The association of strings with states, on the other hand, is not one-to-one. We can still identify q with the lexicographically minimal $s \in$ *strings*(q), and may use q also to denote this string.

A *(Deterministic) Probabilistic Finite Automaton (D)PFA* is defined similar as a (Deterministic) Labeled Markov Chain, with the following modification: τ is now defined on $Q^M \times (Q^M \cup \{e\})$, where $\tau(q, e)$ stands for the probability at state q that the symbol generating process ends. A DPFA M defines the probability of a string $s = \sigma_1 \ldots \sigma_n$ as

$$P_M(s) = \prod_{i=2}^{n} \tau(\sigma_1 \ldots \sigma_{i-1}, \sigma_i)\tau(s, e)$$

if $\sigma_1 = L(start)$. $P_M(s) = 0$ otherwise. This defines a probability distribution on Σ^* provided $\sum_s P_M(s) = 1$, which will be the case if every strongly connected component (SCC) of M contains at least one state q with $\tau(q, e) > 0$. A subset T of Q^M is called *strongly connected* if for each pair (q_i, q_j) of states in T there exists a path $q_0 q_1 \ldots q_n$ such that $q_k \in T$ for $0 \le k \le n$, $\tau(q_k, q_{k+1}) > 0$, $q_0 = q_i$, and $q_n = q_j$. Note that our definition of (D)PFA differs from the more standard ones in that, as in (D)LMCs, 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 $|\Sigma|$. A minor implication of our definitions is also that we assume that the label of the start state is always observed, and so all strings with nonzero probability start with the same symbol, and the probability of the empty string is zero.

C. Probabilistic LTL

Linear time temporal logic (LTL) over the vocabulary Σ is defined as usual by the syntax

$$\varphi ::= \mathit{true} \mid \sigma \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \mathsf{U} \varphi_2 \qquad (\sigma \in \Sigma).$$

For better readability, we also use the derived temporal operators $\Box$ (always) and $\Diamond$ (eventually).

Let φ be an LTL formula over Σ. For $s = \sigma_0\sigma_1\sigma_2 \ldots \in \Sigma^\omega$, $s[j\ldots] = \sigma_j\sigma_{j+1}\sigma_{j+2}\ldots$ is the suffix of s starting at σ_j and $s[j]$ is the prefix $\sigma_0 \ldots \sigma_j$. The LTL semantics for infinite words over Σ are as follows:

- $s \models$ *true*
- $s \models \sigma$ iff $\sigma = \sigma_0$
- $s \models \varphi_1 \wedge \varphi_1$ 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 $0 \leq i < j$

The syntax of probabilistic LTL (PLTL) is:

$$\phi ::= P_{\bowtie r}(\varphi) \quad (\bowtie \in \{\geq, \leq, =\};\ r \in [0,1];\ \varphi \in \text{LTL}).$$

A labeled Markov chain M satisfies the PLTL formula $P_{\bowtie}(\varphi)$ iff $P_M(\varphi) \bowtie r$, where P_M is the probability distribution defined by the LCM M, and $P_M(\varphi)$ is short for $P_M(\{s \mid s \models \varphi, s \in \Sigma^\omega\})$.

III. Learning

We begin by making some assumptions on the process that generates the data from which we learn: the data is generated by a LMC M, and $S_1^\omega, S_2^\omega, \ldots$ is an independent family of P_M-distributed random variables (with values in Σ^ω). Let $L_1, L_2, \ldots$ be an independent family of integer-valued random variables, such that the L_i are also independent of the S_i^ω. We assume that we observe the finite observation sequences $S_i := S_i^\omega[L_i]$, i.e., the first L_i symbols of S_i^ω. Thus, we observe independent runs of the system for a period of time that is determined independently of the observed behavior (in particular, the observation does not automatically end when the system enters a deadlock or failure state – such a situation would rather lead to repeated deadlock or failure observations in the final part of the sequence). We assume that the L_i are unbounded, i.e. $P(L_i > k) > 0$ for all k. This will be satisfied, e.g., by a geometric distribution, which, furthermore, correspond to the natural model according to which at each point in time it is decided with a fixed probability p whether to terminate the observation.

Finally, we denote with $S[n] = S_1, \ldots, S_n$ the sample consisting of the first n observations.

Our algorithm, named AALERGIA, for learning LMCs is a modified version of the ALERGIA algorithm for learning DPFAs from data [5], [8].

AALERGIA starts by building a special DLMC representation of the dataset S called a *frequency prefix tree acceptor FPTA*(S). This is a tree T with a state for each $s \in \text{prefix}(S)$. The state s is labeled with the *frequencies* $f^T(s,\sigma)$ $(\sigma \in \Sigma)$ and $f^T(s,e)$, where $f^T(s,\sigma)$ is the number of strings in S with prefix $s\sigma$, and $f^T(s,e)$ is the number of occurrences of s in S. By normalizing the frequencies $f^T(s,\cdot)$ at each node to probabilities $\tau^T(s,\cdot)$ an FPTA can also be seen as a DPFA. The tree T is kept as a data representation from which relevant statistics are retrieved during the execution of the algorithm.

A second FPTA representation A of S is constructed, and then iteratively transformed by merging states which according to a *compatibility* test (line 8 of Algorithm 1) are determined to correspond to the same state in the data-generating automaton. Following the terminology from [8], Algorithm 1 is described in terms of two sets of states: RED states, which have already been determined as states that will be included in the final output DLMC, and BLUE states which still need to be tested for compatibility with some RED state.

The *Merge* procedure (line 9 of Algorithm 1) is exactly as described in [8]: first, the (unique) transition leading into $q_b \in$ BLUE is re-directed into $q_r \in$ RED, i.e., the unique state q' for which $f^A(q', q_b) > 0$, one sets $f^A(q', q_r) \leftarrow f^A(q', q_b)$ and $f^A(q', q_b) = 0$. Then, recursively, the frequencies $f^A(q_b s, \sigma)$ $(s \in \Sigma^*)$ are added to the frequencies $f^A(q_r s, \sigma)$. Thus, during the whole execution of the algorithm, frequency counts f^A are maintained at the states, which define normalized transition probabilities τ^A.

At termination of the algorithm, A is a DPFA (with τ^A defined by the normalized frequency counts f^A) that represents the data generating distribution, and includes a model for the observation lengths L_i via the termination probabilities $\tau^A(q,e)$. Since we are interested in the underlying DLMC defining a distribution over infinite strings Σ^ω, in the last step the DPFA is turned into a DLMC by renormalizing the transition probabilities $\tau(q,\sigma) \leftarrow \tau(q,\sigma)/(1-\tau(q,e))$.

The main difference between our and previous versions of the Alergia algorithm is the implementation of the compatibility test. In [6] the test *Compatible*() is conducted by recursively testing compatibility of the termination and the next symbol probabilities at states $q_r s$, $q_b s$ in T $(s \in \Sigma^*)$. The compatibility tests are based on Hoeffding bounds for the difference $|\tau^T(q_r s,\sigma) - \tau^T(q_b s,\sigma)|$. In the DSAI algorithm [8], compatibility is tested by the condition

$$d_\infty(P_{A,q_r}, P_{A,q_b}) < \mu, \tag{1}$$

for a fixed threshold μ, where for two distributions P and Q

$$d_\infty(P,Q) := \max_{s \in \Sigma^*} |P(s) - Q(s)|. \tag{2}$$

Our algorithm is related to DSAI, but differs in two aspects: first, we determine the compatibility of $q_r, q_b \in A$ not via the probability distributions P_{A,q_r}, P_{A,q_b} defined in the current automaton A, but via the probability distributions P_{T,q_r}, P_{T,q_b} of the corresponding states in the original tree T (i.e., the states associated in T with the lexicographic minimal strings in *strings*(q_r), respectively *strings*(q_b).) . The reason for this is that the latter distributions have a clear interpretation as empirical probabilities defined by the data S, which is instrumental in proving the consistency of the learning approach.

The second modification is that we replace the fixed threshold μ by a data-dependent threshold. For this, we denote with $f^T(s) := f^T(s,e) + \sum_\sigma f^T(s,\sigma)$ (this is just the number of strings in S with prefix s), and define as our compatibility criterion

$$d_\infty(P_{T,q_r}, P_{T,q_b}) < I_\alpha(f^T(q_r)) + I_\alpha(f^T(q_b)), \tag{3}$$

where for $\alpha > 0$ one defines $I_\alpha(n) := \sqrt{6\alpha \log(n)/n}$.

The use of this criterion is based on Lemma 2 due to Angluin [2] (cf. Appendix A), and instrumental for our consistency proof. We call the resulting algorithm AALERGIA, with the extra 'A' standing for the Angluin-based compatibility criterion. Algorithms 2 and 3 show the practical implementation of criterion (3).

The algorithm takes the parameter α as an input. Larger values of α lead to a larger ϵ bound in the compatibility

Algorithm 1 AALERGIA

Input: : A set S of strings and a parameter $\alpha > 0$.
Output: : A DLMC A.

$T \leftarrow$ *FPTA*(S) and $A \leftarrow$ *FPTA*(S)
RED $\leftarrow$ *start*^{A}
BLUE $\leftarrow \{q : q = \mathit{start}^A\sigma, \sigma \in \Sigma, \mathit{start}^A\sigma \in \text{prefix}(S)\}$;
while BLUE $\neq \emptyset$ **do**
  $q_b \leftarrow$ lexicographically minimal $q \in$ BLUE
  merged $\leftarrow$ *false*
  for $q_r \in$ RED & !*merged* /* q_r in lexicographic order */ **do**
    if *Compatible*(T, q_r, q_b, α) **then**
      Merge(A, q_r, q_b)
      merged $\leftarrow$ *true*
    end if
  end for
  if !*merged* **then**
    RED $\leftarrow$ RED $\cup\, q_b$
  else
    BLUE $\leftarrow$ BLUE $\setminus q_b \cup \{q = q_b\sigma \mid \sigma \in \Sigma, q_b\sigma \in \text{prefix}(S)\}$
  end if
end while
return *makeDLMC*(A);

Algorithm 2 *Compatible*

Input: : FPTA T, states q_r, q_b, and $\alpha > 0$
Output: : *true* if the distributions P_{T,q_r}, P_{T,q_b} are within Angluin's bound

if $L(q_r) \neq L(q_b)$ /* Equality of labeling symbol */ **then**
  return *false*
end if
$\epsilon_r \leftarrow I_\alpha(f^T(q_r))$ and $\epsilon_b \leftarrow I_\alpha(f^T(q_b))$
return *Compatible_recurse*$(T, q_r, q_b, 1, 1, \epsilon_r + \epsilon_b)$

test, and hence to more merge operations and smaller output models in AALERGIA. According to the theoretical analysis of Section IV, any $\alpha > 1$ is admissible to obtain convergence guarantees in the large sample limit. However, for any particular finite sample size n we try to tune the choice of α so as to obtain the best approximation to the true model.

For this we run AALERGIA with different α values, and evaluate the learned model using the *Bayesian Information Criterion (BIC)* score. This score evaluates the learned models based on likelihood, but subtracts a penalty term for the size of the model. Concretely, the BIC score of a DLMC A given data $S[n]$ is defined as

$$\mathit{BIC}(A \mid S[n]) := \mathit{log}(P_A(S[n])) - 1/2\, |A|\, \mathit{log}(N),$$

where $|A|$ is the size of A, and $N = \sum l_i$ is the total size of the data.

Using a golden section search [15, Section E.1.1] we systematically search for an α value optimizing the BIC score of the learned model. Our implementation of AALERGIA is based

Algorithm 3 *Compatible_recurse*

Input: : $T, q_r, q_b, p_r, p_b, \epsilon$
Output: : *true* if $|p_r \cdot P_{T,q_r}(s) - p_b \cdot P_{T,q_b}(s)| < \epsilon$ for all $s \in \Sigma^*$

if $p_r < \epsilon$ and $p_b < \epsilon$ **then**
  return *true*
end if
if $p_r > \epsilon$ and $p_b = 0$ **then**
  return *false*
end if
if $p_b > \epsilon$ and $p_r = 0$ **then**
  return *false*
end if
if $|p_r \cdot \tau^T(p_r, e) - p_b \cdot \tau^T(p_b, e)| > \epsilon$ **then**
  return *false*
end if
for $\sigma \in \Sigma$ **do**
  if !*Compatible_recurse*$(T, q_r\sigma, q_b\sigma, p_r \cdot \tau^T(q_r, \sigma), p_b \cdot \tau^T(q_b, \sigma), \epsilon)$ **then**
    return *false*
  end if
end for
return *true*

on the Matlab gitoolbox (http://code.google.com/p/gitoolbox/) and is available at mi.cs.aau.dk/code/aalergia.

IV. CONVERGENCE ANALYSIS

Convergence guarantees of AALERGIA are derived in two steps: first, it is established that the algorithm will identify the correct structure of the data-generating automaton M. Then, convergence of the estimates for the learned transition probabilties τ is used to establish that PLTL queries will be answered (approximately) correctly.

The following lemma provides a simple characterization of bisimulations in DLMCs.

Lemma 1: Let M be a DLMC. The equivalence relation $q \sim q' :\Leftrightarrow P_{M,q} = P_{M,q'}$ is a probabilistic bisimulation.
Proofs of results in this section are given in Appendix B. We denote with $M/\!\sim$ the quotient automaton defined by $\sim$.

In the following we assume that all random variables and events of interest are defined on an underlying probability space Ω with probability measure P. The distribution of a random variable X defined on this space is denoted P_X, and the probability of a measureable event $E \subseteq \Omega$ is $P(E)$. For a sequence of events $(E_n)_n$ indexed by $n \in \mathbb{N}$, we write E_n *a.a.* ("almost always") for the event that all but finitely many E_n take place, and E_n *i.o.* ("infinitely often") for the event that infinitely many E_n take place.

For a DLMC $M = \langle Q^M, \Sigma, \mathit{start}^M, \tau^M, L^M\rangle$ we define the *structure* of M as $\hat{M} := \langle Q^M, \Sigma, \mathit{start}^M, \hat{\tau}^M, L^M\rangle$, where $\hat{\tau}^M \subseteq Q^M \times Q^M$ is the transition relation defined by $(q, q') \in \hat{\tau}^M \Leftrightarrow \tau^M(q, q') > 0$. The following theorem states that the structure of M will be identified if M is a DLMC.

Theorem 1: If M is a DLMC, A^n the DLMC returned by AALERGIA on input $S[n]$, and $\alpha > 1$, then

$$P(\hat{A}^n = \widehat{M/\sim}\ a.a.) = 1.$$

The theoretical consistency result requires an α-parameter greater than 1. Since Theorem 1 is a large-sample result, it may nevertheless be the case that for smaller datasets running AALERGIA with $\alpha < 1$ leads to better approximations of the true model M.

Lemma 1 together with Theorem 1 lead to:

Theorem 2: Under the assumptions of Theorem 1: for all LTL properties ϕ:

$$P(\lim_{n\to\infty} P_{A^n}(\phi) = P_M(\phi)) = 1. \tag{4}$$

An LTL property ϕ is *bounded* if ϕ only contains until-operators in the time-bounded form $\mathsf{U}^{\leq L}$. For bounded ϕ, we obtain that Theorem 2 also holds when the source model is not deterministic:

Theorem 3: If M is a LMC, and A^n as in Theorem 1, then (4) holds for all bounded LTL properties ϕ.

The proof of this theorem is quite straightforward and omitted due to space constraints.

Even though the learned model converges to an automaton that is bisimilar to the true one, this does not imply an immediate generalization of Theorem 2 to PCTL formulas. Consider as an example the PCTL formula $\phi = \bigcirc P_{=0.5}\Diamond a$. Suppose M consists of four states *start*, q_1, q_2, q_3 all labeled with b, except q_3 labeled with a. Let $\tau(\mathit{start}, q_1) = \tau(q_2, q_2) = \tau(q_3, q_3) = 1$, and $\tau(q_1, q_2) = \tau(q_1, q_3) = 1/2$. Then $P_M(\phi) = 1$. However, in any learned model A^n with only a close approximation of the transition probabilities $\tau(q_1, \cdot)$ one obtains $P_{A^n}(\phi) = 0$. This example shows that an extension of Theorem 2 to PCTL could only be based on a fragment of PCTL in which strict equalities $P_{=r}$ are not allowed, or be based on a notion of approximate satisfaction of PCTL formulas.

V. EXPERIMENTS

In order to test the proposed algorithm we have generated observation sequences from two known system models. We applied the learning algorithm on the sampled sequences, and compared the resulting models with the known generating models in terms of their PLTL properties. For the actual comparison of the models, we considered relevant system properties expressed by PLTL formulas as well as a set Φ of randomly generated PLTL formulas. The formulas were generated using a stochastic context-free grammar, and each formula was restricted to a maximum length of 30.

In 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 LTL formula $\phi \in \Phi$ we 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 the three models (i.e., $P_A(\phi) = P_M(\phi) = P_B(\phi)$), then ϕ was removed from Φ. We finally evaluated the learned models by comparing the mean absolute difference in probability (calculated using PRISM [10]) over the generated formulas for the models M and A:

$$D_A = \frac{1}{|\Phi|}\sum\nolimits_{\phi\in\Phi} |P_M(\varphi) - P_A(\varphi)| \tag{5}$$

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

For the actual empirical analysis we have considered model learning at different levels of abstraction, thus allowing the learning task to be tailored to the properties of interest (c.f. Section I). Specifically, we have performed experiments using the randomized self-stabilizing algorithm by [9] and the craps gambling game described in [3].

In the craps gambling game we start off by rolling two fair six-sided dice. If the outcome is 7 or 11, the game is won, and if the outcome is 2, 3, or 12 then the game is lost. For all other outcomes, the dice are rolled again. If the outcome of the new roll is 7 the game is lost, but if the new roll is equal to the original roll (now called the "point") the game is won. For any other outcome, the dice are rolled again. This process continues until the game is either won or lost. Fig. 1 shows a Markov chain model of the game, which includes an explicit state representation of the possible values of the "point".

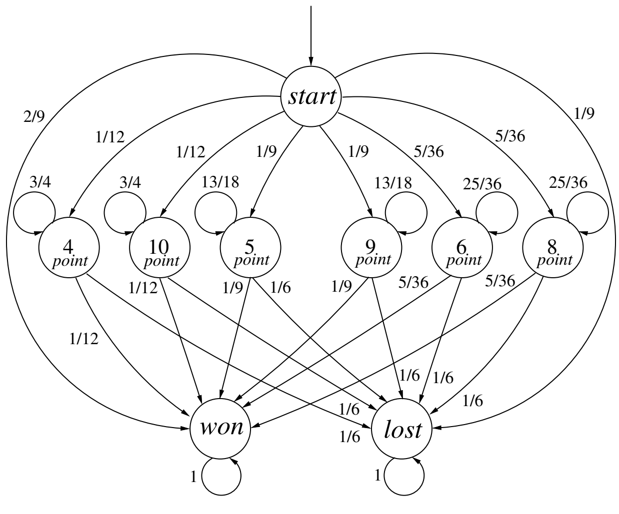

Fig. 1. A Markov chain model for the craps gambling game [3].

Suppose now that we are only interested in the probability of winning the game within i rolls of the dice. For this situation we can abstract away information about the actual value of the "point" and only consider the predicates *start*, *point*, *lost*, and *won*. With this abstracted set of predicates we generated sets of observation sequences whose lengths are geometrically distributed with $p = 0.1$. More specifically, we first generated observation sequences from the model in Fig. 1, and then replaced the values of the dice rolls with the abstract state *point*. Observe that by performing this type of abstraction over the observation sequences, we are effectively sampling observations from a non-deterministic LMC. The results of the experiments are listed in Table I. In the table, #Data is the

TABLE I
THE EXPERIMENTAL RESULTS FOR THE CRAPS GAMBLING MODEL.

#Data	#Seq	Time	Size	α range	D_A	P_1	P_2	P_3	P_4	P_5	P_6	$P(\Diamond won)$
80	5	0.29	31	$[2^{-6}; 2^6]$	0.06402	0.2	0.36	0.3997	0.4	0.4	0.4	0.4
160	19	0.017	4	$[2^{-1}; 2^6]$	0.12169	0.1579	0.2011	0.2619	0.2841	0.287	0.2874	0.2874
320	35	0.03	5	$[2^{-1}; 2^6]$	0.06829	0.1765	0.2405	0.3332	0.369	0.3741	0.3749	0.375
640	65	0.05	4	$[2^{-1}; 2^6]$	0.01967	0.1967	0.283	0.4023	0.4443	0.4496	0.4502	0.4503
1280	141	0.07	4	$[2^{-5}; 2^6]$	0.02945	0.2031	0.2948	0.43	0.4842	0.4923	0.4935	0.4938
2560	291	0.092	4	$[2^{-5}; 2^6]$	0.01183	0.2188	0.2906	0.4037	0.4555	0.4648	0.4665	0.4669
5120	530	0.16	4	$[2^{-6}; 2^6]$	0.01376	0.2063	0.2692	0.3754	0.4314	0.4437	0.4464	0.4472
10240	1081	0.24	4	$[2^{-6}; 2^6]$	0.00619	0.2214	0.2869	0.3942	0.4475	0.4583	0.4604	0.461
20480	2122	0.32	4	$[2^{-6}; 2^6]$	0.00658	0.2297	0.2944	0.4013	0.4554	0.4667	0.469	0.4697
40960	4251	0.48	4	$[2^{-6}; 2^6]$	0.00483	0.2248	0.2885	0.3955	0.4511	0.4632	0.4658	0.4665
M			9			0.2222	0.2994	0.3916	0.4429	0.4551	0.4579	0.4588

size of the data set (total number of symbols) used for learning, #Seq is the number of observation sequences in the data set, 'Size' is the number of nodes in the learned LMC, 'α range' is the interval (identified using the golden section search) for α for which a BIC-optimal LMC is learned, D_A is calculated according to Equation 5 using 1102 formulas, and 'Time' is the average run time (in seconds) of the AALERGIA algorithm; the average is calculated wrt. the iterations performed by the golden section search. Typically the golden section search terminated after 25 to 35 iterations. Finally, the columns labeled P_i list the probabilities $P(\mathit{true}\,\mathsf{U}^{\leq i}\,\mathit{won})$. From the results we see that by using the abstracted alphabet and only 35 observation sequences we still obtain accurate probability estimates of winning the game within i rolls (for $1 \leq i \leq 6$) and we also achieve a relatively small mean difference in probability of the randomly generated LTL formulas; for comparison, $D_B = 0.48933$. Fig. 2 shows an abstract model structure learned from 65 observation sequences.

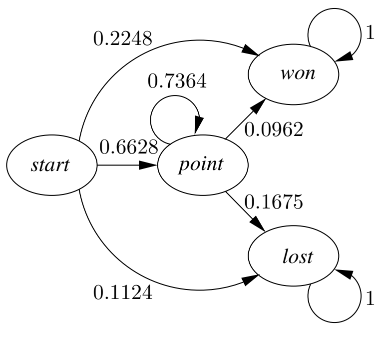

Fig. 2. An LMC model for the craps gambling game learned using 65 observations sequences over the alphabet $\Sigma = \{\mathit{start}, \mathit{point}, \mathit{lost}, \mathit{won}\}$.

Consider now the randomized self-stabilizing protocol by [9]. This algorithm is designed for ring networks with an odd number of processes, and where each process p_i is equipped with a Boolean variable X_i. The protocol operates synchronously such that if $X_i = X_{i-1}$, then p_i makes a uniform random choice about the next value of X_i; otherwise it sets X_i to the current value of X_{i-1}. For each pair of neighboring processes with the same value assigned to their Boolean variables we have a so-called token. The network is stable if it only contains a single token.

Using the protocol above we have first analyzed the behavior of the learning algorithm by varying the number of processes and changing the level of abstraction. For a given number of processes, we have generated sets of observation sequences whose lengths are geometrically distributed with $p = 0.05$. In the first experiment, each generated data point corresponds to a value assignment to the set of Boolean variables associated with the processes. Thus, with n processes, there are 2^n such assignments. In the second experiment, we replaced the data points with abstract states representing the number of tokens defined by the corresponding value assignments. For both experiments we also observed when the network was stable. The results of the experiments with 3, 7, 11, 19, and 21 processes are given in Table II (listing the learning times and the sizes of the learned models) and in Fig. 3 (showing the probability of reaching a stable configuration within L steps as a function of L). From Table II we see (as expected) that the time complexity of learning an abstract model is significantly lower than that of learning a full model. Note that due to time complexity, we have not learned full models for networks with 11, 19, and 21 processes.

Fig. 3 provides a comparison of the probability values $P(\mathit{true}\,\mathsf{U}^{\leq L}\mathit{stable})$ defined by the true models and the models learned (both full and abstract) with datasize 10000. Overall, one observes a very good match between the probability values computed in the different models, with some discrepancies emerging for larger values of L in the 11 and 19 processes models. To illustrate the accuracy of the models learned for smaller amounts of data, we have collected in Table III additional summary statistics for the accuracy obtained from different datasets. The first two columns specify the number of processes, and whether a full (F) or abstract (A) model was learned. The table then contains for different sample sizes (number of symbols) 80,...,10000 the *total variation distance* between the distributions over the time taken to reach the stable state defined by the learned and true models A, M:

$$\mathit{TVD}(A, M) = 1/2 \sum_{i=1}^{\infty} \mid P_M(L) - P_A(L) \mid,$$

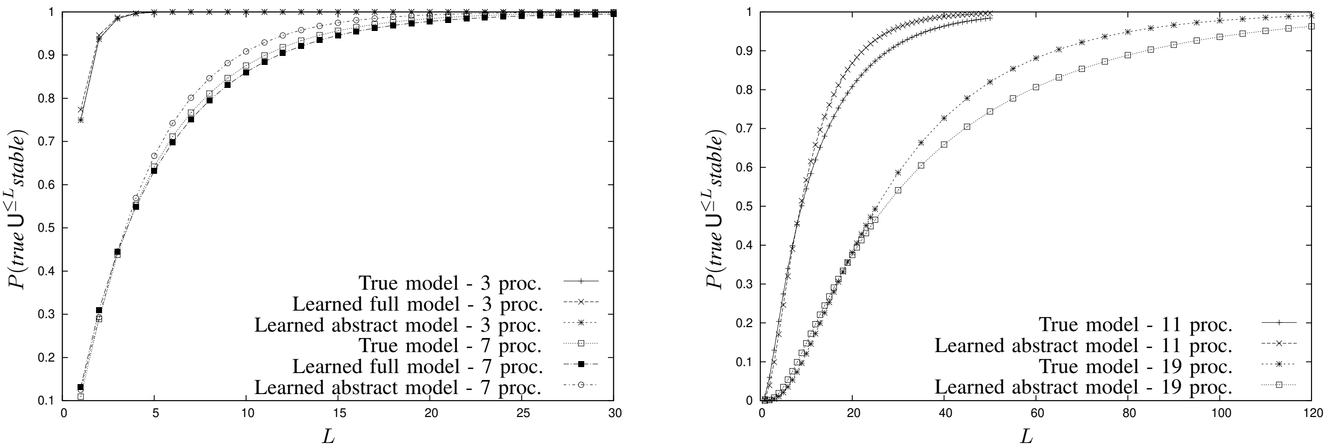

Fig. 3. The figures show $P(true\,\mathsf{U}^{\leq L} stable)$ as a function of the number of steps L. On the left the results for networks with 3 and 7 process are shown, on the rigth the results for networks with 11 and 19 processes. For 3, 7, and 11 processes the networks are learned from data sets with 10000 symbols and an average sequence length of 20. For the results with 19 processes a data set with 100000 symbols were used.

TABLE II
EXPERIMENTAL RESULTS FOR THE SELF-STABILIZING PROTOCOL.

		Full model			Abstract model		
	#Data	Time	Size	α range	Time	Size	α range
3 proc.	80	0.06	37	$[2^{-1}; 2^{6}]$	0.01	2	$[2^{-4}; 2^{6}]$
	320	0.065	10	$[2^{1}; 2^{6}]$	0.025	2	$[2^{-6}; 2^{6}]$
	1280	0.21	8	$[2^{0}; 2^{6}]$	0.05	2	$[2^{-6}; 2^{6}]$
	5120	0.76	8	$[2^{-5}; 2^{6}]$	0.1	2	$[2^{-6}; 2^{6}]$
	10000	1.4	8	$[2^{-6}; 2^{6}]$	0.15	2	$[2^{-9}; 2^{6}]$
7 proc.	80	0.17	71	$[2^{-6}; 2^{6}]$	0.02	11	$[2^{-1}; 2^{6}]$
	320	2.4	291	$[2^{-6}; 2^{6}]$	0.045	4	$[2^{0}; 2^{6}]$
	1280	22	809	$[2^{-1}; 2^{6}]$	0.11	4	$[2^{-4}; 2^{6}]$
	5120	23	387	$[2^{0}; 2^{2}]$	0.32	4	$[2^{-5}; 2^{6}]$
	10000	50	182	$[2^{0}; 2^{3}]$	0.5	4	$[2^{-6}; 2^{6}]$
11 proc.	80				0.25	29	$[2^{-1}; 2^{6}]$
	320				0.3	30	$[2^{-1}; 2^{6}]$
	1280				0.74	40	$[2^{0}; 2^{6}]$
	5120				0.8	6	$[2^{1}; 2^{6}]$
	10000				1.5	8	$[0.334; 0.511]$
19 proc.	80				8.4	52	$[2^{-2}; 2^{6}]$
	320				73	128	$[2^{-1}; 2^{6}]$
	1280				93.2	128	$[2^{0}; 2^{7}]$
	5120				20.2	35	$[2^{-1}; 2^{8}]$
	10000				34.4	34	$[2^{-2}; 2^{9}]$
21 proc.	80				24.4	70	$[2^{-6}; 2^{6}]$
	320				81.4	111	$[2^{-1}; 2^{6}]$
	1280				67.8	113	$[2^{-1}; 2^{7}]$
	5120				55.2	77	$[2^{-2}; 2^{7}]$
	10000				37	40	$[2^{-2}; 2^{7}]$

where $P(L)$ is short for $P(\neg stable\,\mathsf{U}^{=L} stable)$. The summation extends to $L = \infty$, since for some models learned from small datasets one has $P_A(\infty) := P_A(\neg\Diamond stable) > 0$. In fact, this is a main contributing factor for the relatively large distances found for the smaller data sizes. All results in Table III are averages taken over 5 different data samples of the specified size. The results show that for very small sample sizes not very accurate models are learned.

TABLE III
ACCURACY RESULTS FOR RING NETWORK

# P	F/A	Sample Size				
		80	320	1280	5120	10000
3	F	0.166	0.070	0.052	0.034	0.031
3	A	0.206	0.082	0.011	0.015	0.010
7	F	0.698	0.540	0.405	0.057	0.033
7	A	0.435	0.121	0.078	0.042	0.038
11	A	0.497	0.143	0.077	0.065	0.067
19	A	0.704	0.472	0.208	0.107	0.106

Since the abstract models are often significantly smaller than the generating models, the time required for model checking using the abstract models is also expected to be lower. We have analyzed this hypothesis further by measuring the time complexity for evaluating the PLTL property $P(true\,\mathsf{U}^{\leq L} stable)$ for 19 and 21 processes. For the generating model, the total time is calculated as the time used for compiling the PRISM model description to the internal PRISM representation as well as the time used for the actual model checking. For the abstract model, the total time is calculated as the time used for model learning (which produces a model in the PRISM file format), model compilation, and model checking. Fig. 4 shows the (accumulated) time used by both approaches as a function of L. The time complexity of using the abstract models is close to constant. It consists of a constant time (2403 sec. and 2697 sec., respectively) for model learning and model compilation, and a negligible additional linear time for model checking.

Going back to the α ranges reported in the tables above, we see that for the craps gambling game (Table I) and the randomized self-stabilizing protocol (Table II), the learning is quite robust with regard to the α value. This suggests that simply using a theoretically justified constant $\alpha > 1$ will lead to good results.

Finally, we would like to note that for the experiments above, the observed learning time grows roughly linearly in the size of the data set although the worst case time complexity is

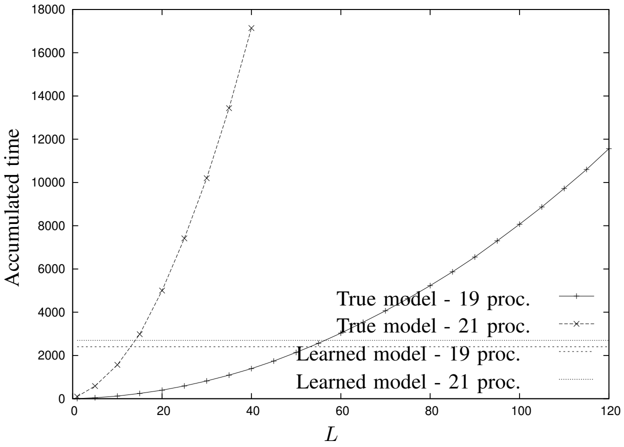

Fig. 4. The figure shows the accumulated time for calculating $P(\mathit{true}\,\mathsf{U}^{\leq L}\mathit{stable})$, i.e., at step i we have the total time for calculating $P(\mathit{true}\,\mathsf{U}^{\leq j}\mathit{stable})$, for $1 \leq j \leq i$.

cubic. This is consistent with the results reported in [5], where a similar behavior is observed.

VI. Conclusion

In this paper we have shown how Machine Learning techniques for probabilistic model learning can be used to obtain system models for model-driven development. The main contributions of this paper are: the AALERGIA algorithm obtained as a modification of previous DPFA-learning algorithms, a completely new proof of consistency for this algorithm that avoids some difficulties found in earlier proofs of related results, novel results that link the convergence of the automata learning procedures to convergence of probability estimates of LTL-definable system properties, and an experimental evaluation that demonstrates the feasibility of the approach in practice.

Compared to statistical model checking [18], [14] the model learning approach offers the advantage that it allows us to generalize from the observation of finite behaviors to infinite behaviors. Thus, for example, in the ring network we obtain from the models learned from a sufficient amount of data also the correct probability for the unbounded property $\phi = \Diamond \mathit{stable}$. This generalization ability, of course, is dependent on the correctness of the assumption that the source model is a DLMC. However, as Theorem 3 shows, in the case where the assumption is wrong we still obtain similar guarantees for the analysis of bounded properties as one obtains with statistical model checking.

In this paper we have focused on the simplest case that system behaviors only consist of sequences of symbols representing system outputs. In ongoing work we are extending the learning procedure to interactive systems with both inputs and outputs.

Appendix A
Consistency: A Review

We first review briefly previous consistency arguments for Alergia-style algorithms [5], [6]. Then we motivate our version of the algorithm and its consistency proof presented in Appendix B by relating it to a basic learning by enumeration approach described by Angluin [2].

A proof of consistency for the Alergia algorithm was first sketched in [5], and then elaborated in [6]. As mentioned in Section III, in Alergia compatibility of the distributions P_{T,q_r}, P_{T,q_b} is tested by recursively applying a Hoeffding test for the equality of the local termination and next symbol probabilities $P_{T,q_r s}(\sigma), P_{T,q_b s}(\sigma)$, $(s \in \Sigma^*,\ \sigma \in \Sigma \cup \{e\})$. The proof of convergence then is based on the claim that as $n \to \infty$, with probability one, all tests will return the correct result (i.e., the test will reject equality iff the local transition probabilities $P_{M,q_r}(\sigma), P_{M,q_b}(\sigma)$ are different in the true model M).

The argument presented in [6] does take into account that as n increases the number of tests performed will also increase (linearly in n), and therefore the level of significance is reduced as a function of n so as to ensure that if each test achieves the prescribed level of significance, then also the probability of making an error in any test can be made small. What the argument of [6] neglects to consider is the fact that as n increases, there will always be tests performed on local distributions $P_{T,q_r s}, P_{T,q_b s}$ for which the data support $f^T(q_r s), f^T(q_b s)$ is small (typically, when $q_r s, q_b s$ are at or near the leaves of T). The concrete test used in [6] will never reject equality of $P_{T,q_r s}, P_{T,q_b s}$ when this data support is sufficiently small, and therefore there will always be some tests performed which are not known to return correct results.

A more complete argument is given in [13]. Here the authors use the fact that one only needs to bound the type 1 errors (rejecting a correct compatibility hypothesis) for a number of tests that is linearly increasing, whereas the type 2 error (accepting an incorrect compatibility hypothesis) only needs to be bounded for a number of tests that depends on the (sample-size independent) number of states in the true model. Sen et al. [13] only show that the probability of not learning the correct model can be made arbitrarily small, which is weaker than the probability one convergence of our Theorem 1.

To motivate our approach, we begin with a short review of Angluin's [2] *learning by enumeration* principle, which is based on the following lemma.

Lemma 2 (Angluin): Let P_S be a distribution on a countable set A, S_i $(i > 0)$ i.i.d. P_S-distributed random variables, and $P_{\bar{S}[n]}$ the empirical distribution defined by $S_1, \ldots, S_n$. Let $\alpha > 1$. Then

$$P(d_\infty(P_S, P_{\bar{S}[n]}) < I_\alpha(n) \quad a.a.) = 1. \tag{6}$$

Now let $\mathcal{M}$ be a countable set of models, so that each $M \in \mathcal{M}$ defines a distribution P_M on a countable set S. Let $M_1, M_2, \ldots$ be a fixed enumeration of $\mathcal{M}$. Then, for a given data sample with empirical distribution $\bar{S}[n]$ return the first model M_i in the list that satisfies $d_*(P_{M_i}, P_{\bar{S}[n]}) < I_a(n)$.

By Lemma 2, any M_i with P_{M_i} equal to the data-generating distribution P_S will satisfy the condition a.a. with probability one. For any M_j with $P_{M_j} \neq P_S$ the condition will not be satisfied i.o., since $I_a(n) \to 0$ for $n \to \infty$. Since only finitely such M_j precede the first M_i with $P_{M_i} = P_S$ in the enumeration, with probability one a.a. none of them will satisfy the condition, and so a.a. the first M_i with $P_{M_i} = P_S$ will be returned by the algorithm.

Theoretically very elegant, learning by enumeration is obviously quite impractical. Nonetheless, learning algorithms in the Alergia family can perhaps best be understood as operational algorithms implementing the basic enumeration idea. The first modification for learning probabilistic automata is to split the learning problem into the two parts of identifying the structure of the automaton, and the estimation of its numerical parameters. Only the first part is done by an enumeration-style procedure, and as a result one is also only guaranteed to identify the structure from (large) finite samples, whereas the parameter estimates will only converge in the limit to the correct values. Secondly, the pointwise tests $P_S = P_{M_j}$ are replaced by membership tests $P_S \in \{P_M \mid M \in \mathcal{M}_j\}$ for sets $\mathcal{M}_j$ according to a fixed protocol that guarantees that for each S a fixed finite sequence of tests (independent of n) has to be performed correctly until a unique M with $P_M = P_S$ is identified.

APPENDIX B
PROOFS

Proof of Lemma 1: Let $q \sim q'$. Then $L(q) = L(q')$, because $P_{M,q}$ and $P_{M,q'}$ assign probability one to the set of strings starting with $L(q)$, respectively $L(q')$. For each $\sigma \in \Sigma$ one furthermore has $\tau(q,\sigma) = \tau(q',\sigma)$, and for σ with $\tau(q,\sigma) > 0$ also $P_{M,q\sigma} = P_{M,q'\sigma}$, i.e., $q\sigma \sim q'\sigma$. Thus, for each $\sim$ equivalence class $A \subseteq Q^M$, one has $\tau(q,A) = \sum_{\sigma:\ q\sigma\in A} \tau(q,\sigma) = \sum_{\sigma:\ q'\sigma\in A} \tau(q',\sigma) = \tau(q',A)$. ■

To prepare the proof of Thereom 1 we begin with a generalization of Lemma 2:

Lemma 3: Let P_S, S_i as in Lemma 2. Let T_i $(i > 0)$ be another i.i.d. family of A-valued random variables distributed according to a distribution P_T. Let $g(n), h(n)$ $(n \geq 1)$ be integer-valued random variables, such that with probability one the sequences $(g(n))_n$ and $(h(n))_n$ are non-decreasing and unbounded. Then $P_S = P_T$ iff

$$P(d_*(P_{\bar{T}[g(n)]}, P_{\bar{S}[f(n)]}) < I(g(n)) + I(f(n))\ a.a.) = 1,$$

and $P_S \neq P_T$ iff

$$P(d_*(P_{\bar{T}[g(n)]}, P_{\bar{S}[f(n)]}) > I(g(n)) + I(f(n))\ a.a.) = 1.$$

Observe that apart from the i.i.d. assumption on the families $(S_i)_i$, respectively $(T_i)_i$, no further independence assumptions are made. In particular, S_i need not be independent of T_j, or any of these be independent of the $g(n), h(n)$.

Proof: For the first statement it is sufficient to show that

$$P(d_*(P_S, P_{\bar{S}[f(n)]}) < I(f(n))\ a.a.) = 1 \tag{7}$$

(and analogously for $T, \bar{T}[g(n)]$). The statement then follows with the triangle inequality. To show (7) we write

$$d_*(P_S, P_{\bar{S}[n]}) < I(n)\ a.a. = \cup_{k\geq 0} A_k,$$

with

$$A_k := \bigcap_{l\geq k} \{d_\infty(P_S, P_{\bar{S}[l]}) < I(l)\}.$$

Then

$$A_k \setminus \{d_\infty(P_S, P_{\bar{S}[f(n)]}) < I(f(n))\ a.a.\} = \{f(n) < k\ i.o.\} \tag{8}$$

According to our assumptions, the event on the right of (8) has probability zero. It follows that $\cup_{k\geq 0} A_k \setminus \{d_\infty(P_S, P_{\bar{S}[f(n)]}) < I(f(n))\ a.a.\}$ also has probability zero, which with (6) yields (7).

To show the second statement of the lemma, let $\delta := d_\infty(P_T, P_S) > 0$. It is sufficient to show that

$$P(I(g(n) + I(f(n)) < \delta/2\ a.a.) = 1, \tag{9}$$

and

$$P(d_\infty(P_{\bar{T}[g(n)]}, P_{\bar{S}[f(n)]}) > \delta/2\ a.a.) = 1. \tag{10}$$

(9) follows immediately from our assumptions on $g(n), f(n)$, and the fact that $I(n) \to 0$ for $n \to \infty$.

(10) follows by an identical argument as for the first part of the lemma, where instead of (6) one uses that by the strong law of large numbers (or, as a special case of (6))

$$P(d_\infty(P_S, P_{\bar{S}[n]}) < \delta/4\ a.a.) = 1.$$

■

We are now ready to proceed to the proof of Theorem 1, for which we need the following additional notation and definitions.

In the following we use $\bullet$ to denote either r or b, i.e., definitions or statements containing expressions involving $\bullet$ represent simultaneous definitions or statements for both r and b.

As in the statement of the theorem, M always denotes the true model from which the data was generated. We denote with Σ_M^* the set of words that have nonzero probability under P_M. Any execution of the inner loop at lines 7-12 is called an iteration of the algorithm.

We denote with $A_i^n,\ q_{\bullet,i}^n$ the values of A, respectively $q_\bullet$ immediately before the ith iteration when AALERGIA is run on data $S[n]$. In particular, A_1^n is the initial FPTA constructed in line 1. We also recall that A^n is the automaton returned by AALERGIA.

Proof of Theorem 1:

In order to make the association between strings and states more explicit, we now sometimes write $q(s)$ for the state reached by string s, and $s(q)$ for the lexicographically minimal string associated with q. We say that the FPTA A_1^n is *rich*, if it contains

(a) for all $m \in M/\!\sim$ a node for $s(m)$, as well as nodes for $s(m)\sigma$ for all $\sigma \in \Sigma$ with $s(m)\sigma \in \Sigma_M^*$,
(b) nodes for all lexicographic predecessors in Σ_M^* of nodes included under (a).

We prove the theorem via a sequence of three claims.

Claim 1: $P(A_1^n$ is rich $a.a.) = 1$.

Proof of Claim 1: immediate from the strong law of large numbers.

We say that the call *compatible*$(A_i^n, q_{r,i}^n, q_{b,i}^n, \alpha)$ returns the correct result (abbreviated r.c.r.), if the value *true* is returned iff

$$P_{M,s(q_{r,i}^n)} = P_{M,s(q_{b,i}^n)}.$$

For $n, i \geq 1$ we define the event

$$G_i^n := \{A_1^n \text{ rich}\} \cap \bigcap_{j<i} \{\textit{compatible}(A_j^n, q_{r,j}^n, q_{b,j}^n, \alpha) \text{ r.c.r.}\}.$$

Intuitively, G_i^n represents the event that for the first $i-1$ iterations the algorithm performed a correct and typical (for large samples) run.

The following claim essentially states that when G_i^n holds, then the call *compatible*$(A_i^n, q_{r,i}^n, q_{b,i}^n, \alpha)$ will test the identity $P_{M,s_r} = P_{M,s_b}$ for fixed s_r, s_b that do not further depend on A_i^n. Furthermore, for a fixed k, if G_{k+1}^n holds, then the algorithm terminates after at most k iterations with the correct model structure $\widehat{M/\sim}$.

Claim 2: There exists $k > 0$, $s_{\bullet,i} \in \Sigma_M^*$, $(1 \leq i \leq k)$, such that:

(i) $G_i^n \subseteq \{s(q_{\bullet,i}^n) = s_{\bullet,i}\}$

(ii) $G_{k+1}^n \subseteq \{A^n = A_{k+1}^n\} \cap \{\hat{A}^n = \widehat{M/\sim}\}$

Proof of Claim 2:

Let k be the number of nodes required to be included in A_1 for A_1 to be rich. (i): For $i < k$, given G_i^n, within the first $i-1$ iterations exactly the same merge operations were performed, and therefore before the ith iteration the lexicographically minimal elements of *Blue* and *Red* are fixed strings $s_{\bullet,i}$.

(ii): If G_{k+1}^n holds, then after at most k iterations A_i^n will contain for every $m \in M/\sim$ one red node $q_r(m)$ with $s(q_r(m)) = s(m)$ (and no other red nodes). Furthermore, for each $\sigma \in \Sigma$ with $P_{M/\sim,m}(\sigma) > 0$, i.e., *first*$(m)\sigma \in \Sigma_M^*$, the node *first*$(m)\sigma \in A_1^n$ has been merged with $q_r(\delta(m,\sigma))$, which means that the transition relations in M and A_{k+1}^n are isomorphic, and that no blue nodes remain in A_{k+1}^n.

Claim 3: Let k be as in Claim 2. Then

$$P(\textit{compatible}(A_i^n, q_{r,i}^n, q_{b,i}^n, \alpha) \text{ r.c.r. } a.a. \mid G_i^n \ a.a.) = \mathit{1}$$

Proof of Claim 3 The assumptions of Claim 2 (i) are satisfied a.a., and therefore Compatible$(A_i^n, q_{r,i}^n, q_{b,i}^n)$ is a.a. equivalent to

$$d_\infty(P_{A_1^n,s_{r,i}}, P_{A_1^n,s_{b,i}}) < I(f^{T^n}(s_{r,i})) + I(f^{T^n}(s_{b,i}))$$

$P_{S_n,s_{\bullet,i}}$ is the empirical distribution of an i.i.d. sample of size $f^n(s_{\bullet,i})$ from $P_{M,s_{\bullet,i}}$.

With probability one the sample sizes $f^n(s_{\bullet,i})$ grow unbounded, so that the conditions of Lemma 3 are satisfied. It thus follows that *compatible*$(A_i^n, q_{r,i}^n, q_{b,i}^n, \alpha)$ r.c.r. a.a. with probability one.

Combining claims 1, 2(i), and 3 now yields that $P(G_{k+1}^n \ a.a.) = 1$, which, with claim 2(ii) concludes the proof of Theorem 1. ∎

Proof of Theorem 2: By Lemma 1 $P_M(\phi) = P_{M/\sim}(\phi)$. By Theorem 1 A^n with probability 1 a.a. has the same structure as $M/\sim$. The transition probabilities $\delta^{A^n}(m,\sigma)$ are the empirical probabilities from i.i.d. samples from $\delta^{M/\sim}(m,\sigma)$, and therefore converge with probability 1 to $\delta^{M/\sim}(m,\sigma)$.

Using the automata-theoretic approach to verification [16], [7], [17], the probabilities $P_{A^n}(\phi), P_{M/\sim}(\phi)$ can be identified with reachability probabilities in the product of the Markov chains $A^n, M/\sim$ and a Büchi automaton B representing ϕ. The state spaces of all these product Markov chains will a.a. be identical, and the transition probabilities in the products $A^n \times B$ converge to the transition probabilities in the product $M/\sim \times B$. Since the reachability probabilities are continuous functions of the transition probabilities, this implies the convergence $P_{A^n}(\phi) \to P_M(\phi)$. ∎

REFERENCES

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[2] D. Angluin. Identifying languages from stochastic examples. Technical Report YALEU/DCS/RR-614, Yale University, 1988.

[3] C. Baier and J. P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[4] T. Berg, B. Jonsson, M. Leucker, and M. Saksena. Insights to angluin's learning. *Electron. Notes Theor. Comput. Sci.*, 118:3–18, February 2005.

[5] R. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *Grammatical Inference and Applications*, volume 862 of *Lecture Notes in Computer Science*, pages 139–152. Springer Berlin / Heidelberg, 1994.

[6] R. C. Carrasco and J. Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *ITA*, pages 1–20, 1999.

[7] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

[8] C. de Higuera. *Grammatical Inference — Learning Automata and Grammers*. Cambridge University Press, 2010.

[9] T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35(2):63–67, 1990.

[10] M. Kwiatkowska, G. Norman, and D. Parker. Prism 2.0: a tool for probabilistic model checking. *In proceedings of the first international conference on the Quantitative Evaluation of system (QEST)*, pages 322–323, 2004.

[11] M. Leucker. Learning meets verification. In *Proceedings of the 5th international conference on Formal methods for components and objects*, FMCO'06, pages 127–151, Berlin, Heidelberg, 2007. Springer-Verlag.

[12] H. Raffelt, B. Steffen, and T. Berg. Learnlib: a library for automata learning and experimentation. In *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, FMICS '05, pages 62–71, New York, NY, USA, 2005. ACM.

[13] K. Sen, M. Viswanathan, and G. Agha. Learning continuous time markov chains from sample executions. In *Proc. of QEST'04*, pages 146–155, 2004.

[14] K. Sen, M. Viswanathan, and G. Agha. Statistical model checkling of black-box probabilistic systems. In R. Alur and D. Peled, editors, *Computer Aided Verification 2004, LNCS 3114*, 2004.

[15] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 2006.

[16] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 327–338, 1985.

[17] M. Y. Vardi. Probabilistic linear-time model checking: an overview of the automata-theoretic approach. In J.-P. Katoen, editor, *Formal methods for real-time and probabilstic systems (ARTS-99) : 5th International AMAST Workshop*, volume 1601 of *LNCS*, 1999.

[18] H. L. S. Younes and R. G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Computer Aided Verification 2002, LNCS 2404*, 2002.