Learning Markov Decision Processes for Model Checking

Hua Mao, Yingke Chen, Manfred Jaeger, Thomas D. Nielsen, Kim G. Larsen, and Brian Nielsen

Department of Computer Science
Aalborg University
Denmark

[huamao,ykchen,jaeger,tdn,kgl,bnielsen] @cs.aau.dk

Constructing an accurate system model for formal model verification can be both resource demanding and time-consuming. To alleviate this shortcoming, algorithms have been proposed for automatically learning system models based on observed system behaviors. In this paper we extend the algorithm on learning probabilistic automata to reactive systems, where the observed system behavior is in the form of alternating sequences of inputs and outputs. We propose an algorithm for automatically learning a deterministic labeled Markov decision process model from the observed behavior of a reactive system. The proposed learning algorithm is adapted from algorithms for learning deterministic probabilistic finite automata, and extended to include both probabilistic and nondeterministic transitions. The algorithm is empirically analyzed and evaluated by learning system models of slot machines. The evaluation is performed by analyzing the probabilistic linear temporal logic properties of the system as well as by analyzing the schedulers, in particular the optimal schedulers, induced by the learned models.

1 Introduction

Model checking is successfully used in many areas to check a formal system model against a specification given by a logical expression. However, to construct an accurate model of an industrial system is usually difficult and time consuming. The difficulty of model construction, or system modeling, is regarded by industry as a challenge to adopt other powerful model-driven development (MDD) techniques and tools as well. Meanwhile, the necessary accurate, updated and detailed documentations rarely exist for legacy software or 3rd party components. Therefore, we consider system model learning techniques [12–14, 16], which can automatically construct or *learn* an accurate high-level model from observations of a given black-box embedded system component. Afterwards, given a learned and explicitly represented model, model checking and other MDD techniques can be applied with other existing component models.

For learning non-probabilistic system models, Angluin's approaches [2] has been well developed and implemented [1, 12, 14]. However, a disadvantage of those system models is that complex systems are often only partially observable via their interactions with the user. Even worse, the observation is often not noise-free. Compared with deterministic models, probabilistic models are more feasible to model a complicated real system and its physical components, unpredictable user interactions and the usage of randomized algorithms. In this paper, we focus on probabilistic models. Sen et al. [16] adapted the algorithm from [5] for learning Markov chain models in purpose of verification. In [13], a learning approach related to [16] is developed, and strong theoretical and experimental consistency results are established. Considering a limited situation that the target system is not fully under control and only a *single* observation sequence is available, the algorithm for learning variable order Markov chains [15] is developed to verify stationary system properties on the learned models [6].

U. Fahrenberg, A. Legay and C. Thrane:
Quantities in Formal Methods (QFM 2012)
EPTCS 103, 2012, pp. 49–63, doi:10.4204/EPTCS.103.6

In Markov chains, probabilistic choices may serve to model and quantify possible outcomes of randomized actions or the interface between a system and its environment. This, nevertheless, requires abundant statistical experiments to obtain adequate distributions to model the average behavior of the environment. It is a natural choice to model by nondeterminism a system which is open for interaction from environment, system properties then need to be guaranteed for all potential environments [17]. Therefore, Markov decision processes (MDPs), which exhibit both nondeterministic and probabilistic behavior, are widely used for modeling reactive systems [3]. In this paper, we adapted the algorithm for learning deterministic probabilistic finite automata to include nondeterministic actions. Particularly, we learn deterministic labeled Markov decision processes (DLMDPs), where input actions are chosen nondeterministically and outputs given inputs are determined probabilistically, from the observed input and output behavior of a reactive system. This leads to another motivation of the learning purposes. For large systems, we may be interested in only one component, and it receives certain inputs from the environment or other components. Then the learner can output a model which is the representation of this component.

Besides model learning, statistical model checking (SMC) [11, 20] techniques can also be used to analyze black-box systems. Statistical model-checking uses hypothesis testing based on sampling runs of a system that allows the user to check to a desired level of confidence whether a given logical property holds with a given (minimum) probability. Unfortunately, this technique is not well suited to MDPs since the presence of nondeterminism making running for sample paths is not well defined [4] without an extra scheduler. Moreover, the model output by the model learning approach can be used to other properties without re-sampling, as well as being used for other MDD tasks.

The main contribution of this paper is the development of IOALERGIA algorithm for learning DLMDP, which is obtained as an adaptation of the previous ALERGIA [5] algorithm. In order to demonstrate the applicability, the new algorithm is applied to learning models for slot machines from observed system behavior, which is in the form of alternating sequences of inputs and outputs. The evaluation is performed by analyzing and comparing probabilistic linear time properties in the learned model and the known generating model, as well as maximal expected reward and optimal schedulers.

This paper is structured as follows: section 2 contains background material. Section 3 describes the procedure of generating learning data, while section 4 describes IOALERGIA algorithm. Section 5 demonstrate its applicability through a case study concerning slot machine. Section 6 concludes the paper.

2 Preliminaries

2.1 Labeled Markov Decision Processes

Definition 1 (LMDP) *A* labeled Markov decision processes (LMDP) *is a tuple* $M = (Q, \Sigma_I, \Sigma_O, \pi, \tau, L)$

- Q *is a finite set of states,*
- Σ_I *is a finite input alphabet, and* Σ_O *is a finite output alphabet,*
- $\pi : Q \to [0,1]$ *is an initial probability distribution such that* $\sum_{q \in Q} \pi(q) = 1$,
- $\tau : Q \times \Sigma_I \times Q \to [0,1]$ *is the transition probability function such that for all* $q \in Q$ *and all* $\alpha \in \Sigma_I$, $\sum_{q' \in Q} \tau(q, \alpha, q') = 1$, *or* $\sum_{q' \in Q} \tau(q, \alpha, q') = 0$,
- $L : Q \to \Sigma_O$ *is a labeling function.*

An input $\alpha \in \Sigma_I$ is enabled in state $q \in Q$ if and only if $\sum_{q' \in Q} \tau(q, \alpha, q') = 1$. Let $Act(q)$ denote the set of enabled actions in q.

Definition 2 (DLMDP) *A LMDP is deterministic if*

- *There exists a state* $q_s \in Q$ *with* $\pi(q_s) = 1$,
- *For all* $q \in Q$, $\alpha \in \Sigma_I$ *and* $\sigma \in \Sigma_O$, *there exists at most one* $q' \in Q$ *with* $L(q') = \sigma$ *and* $\tau(q, \alpha, q') > 0$. *We then also write* $\tau(q, \alpha, \sigma)$ *instead of* $\tau(q, \alpha, q')$.

2.2 Strings

Let $\Sigma_O(\Sigma_I\Sigma_O)^*$ and $\Sigma_O(\Sigma_I\Sigma_O)^\omega$ denote the set of all finite, respectively infinite strings of alternative input and output symbols. For a finite string $s = \sigma_0\alpha_1\sigma_1 \ldots \alpha_n\sigma_n$, $\alpha_i \in \Sigma_I$ and $\sigma_i \in \Sigma_O$, the set of all its prefixes is defined as:

$$\text{prefix}(s) = \{\sigma_0\alpha_1\sigma_1 \ldots \alpha_k\sigma_k \mid 0 \leq k \leq n, k \in \mathbb{N}\}$$

For a set of strings S, $\text{prefix}(S)$ denotes the set of all prefixes of strings $s \in S$. We assume an lexicographic ordering on $\Sigma_O(\Sigma_I\Sigma_O)^*$.

In a DLMDP there is a tight connection between strings and states: given an observed string s there is a unique state q that the LMDP 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 DLMDP 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.

2.3 Scheduler

A scheduler [3] for a MDP M is a function $\mathfrak{S} : Q^+ \to \Sigma_I$ such that $\mathfrak{S}(q_0q_1 \ldots q_n) \in Act(q_n)$ for all $q_0, q_1, \ldots, q_n \in Q^+$. The scheduler chooses in any state q one action $\alpha \in \Sigma_I$, and induces a Markov chain, i.e., the behavior of an MDP M under the decisions of scheduler $\mathfrak{S}$ can be formalized by a Markov chain $M_{\mathfrak{S}}$ [3, Section 10.6].

A labeled Markov chain (LMC) $M_{\mathfrak{S}}$ of an LMDP M induced by a scheduler $\mathfrak{S}$ defines a probability measure $P_{M_{\mathfrak{S}}}$ on $(\Sigma_O)^\omega$ which is the basis for associating probabilities with events in the LMC $M_{\mathfrak{S}}$. The probability of a string $s = \sigma_0\sigma_1 \ldots \sigma_n, \sigma \in \Sigma_O$ defined by $M_{\mathfrak{S}}$ is:

$$P_{M_{\mathfrak{S}}}(s) = \prod_{i=1}^{n} \tau_{\mathfrak{S}}(\sigma_0\sigma_1 \ldots \sigma_{i-1}, \sigma_i)$$

where $\tau_{\mathfrak{S}}$ is the transition probability function of $M_{\mathfrak{S}}$.

2.4 Probabilistic LTL

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

$$\varphi ::= a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \mathsf{U} \varphi_2 \qquad a \in \Sigma_O$$

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

Let φ be an LTL formula. For $s = \sigma_0\sigma_1 \ldots \in (\Sigma_O)^\omega$, $s[j \ldots] = \sigma_j\sigma_{j+1}\sigma_{j+2} \ldots$ is the suffix of s starting with the (j)th symbol σ_j. Then the LTL semantics for infinite words over Σ_O 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 $s[1\ldots] \models \varphi$
- $s \models \varphi_1 \mathrm{U} \varphi_2$, 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:

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

A labeled Markov decision process M satisfies the PLTL formula $P_{\bowtie r}(\varphi)$ iff $P_{M_{\mathfrak{S}}}(\varphi) \bowtie r$ for all schedulers of M, where $P_{M_{\mathfrak{S}}}$ is the probability distribution defined by the LMC induced by a scheduler $\mathfrak{S}$ of M, and $P_{M_{\mathfrak{S}}}(\varphi)$ is short for $P_{M_{\mathfrak{S}}}(s|s \models \varphi, s \in (\Sigma_O)^\omega)$

The quantitative analysis of an MDP M against specification φ amounts to establishing the lower and upper bounds that can be guaranteed, when ranging over all schedulers. This corresponds to computing

$$P_M^{\max}(\varphi) = \sup_{\mathfrak{S}} P_{M_{\mathfrak{S}}}(\varphi) \text{ and } P_M^{\min}(\varphi) = \inf_{\mathfrak{S}} P_{M_{\mathfrak{S}}}(\varphi)$$

where the infimum and the supremum are taken over all schedulers for M.

3 Data Generation

The data we learn from is generated by observing the running reactive system. From the system we can observe input actions which determine probability distributions over successor states, and outputs which are labels of successor states. The learning algorithm requires that all nondeterministic choices are resolved by a *fair* scheduler $\mathfrak{S}$ which means each input action will be chosen infinitely often. We assume that the input and output will be observed alternately, and every observation sequence starts from the label of the initial state, and ends in a state, i.e. $\sigma_0\alpha_1\sigma_1\ldots\alpha_n\sigma_n$, with $\alpha_i \in \Sigma_I$ and $\sigma_i \in \Sigma_O$.

Usually, enabled and disabled actions for states in a black-box system are unknown. Therefore, we allow that all actions can be chosen on each state of the system. For enabled actions, the system will transit to other states, and the input and the corresponding label of the successor state will be collected. For disabled actions, the system will stay in the same state but give a special error message. Through this setting, enabled and disabled inputs could be distinguished. Furthermore, we denote the prompted error by *err*, thus the output alphabet Σ_O is extended to $\Sigma_O \cup \{err\}$. Due to the memoryless scheduler, the same disabled input on the same state could be chosen more than once, and the statistic information about *err* will be found necessary in the following *compatibility* test.

After all nondeterministic choices have been resolved, let $S_1^\omega, S_2^\omega, \ldots$ be an independent family of $P_{M_{\mathfrak{S}}}$-distributed random variables (with values in $\Sigma_O(\Sigma_I\Sigma_O)^\omega$), and $L_1, L_2, \ldots$ 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 $S_i := \sigma_0\alpha_1\sigma_1\ldots\alpha_{L_i}\sigma_{L_i}$, i.e., the first L_i symbols of S_i^ω. Thus, we observe the independent run 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 $k \in \mathbb{N}$.

This will be satisfied by a geometric distribution for the L_i. For some models, there exists a uniquely labeled absorbing state which can be identified by its observation (e.g., a failure state which can not recover). When prior knowledge is available, observations can be stopped when the model reaches that state.

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

4 Learning

IOALERGIA for learning DLMDP consists of two phases. Firstly, represent the data as I/O frequency prefix tree acceptor (IOFPTA) where common prefixes are combined together. Then, do *compatibility* test on the tree following lexicographical order. If two states are compatible which requires that the next state distributions given the same input are compatible, they and their successor states will be merged correspondingly.

4.1 IOFPTA

The *input and output frequency prefix tree acceptor* IOFPTA is constructed as a representation of the set of strings S which captures the behavior of the reactive system under observation. Since in DLMDP, same sequences will lead to the same state, then in IOFPTA common prefixes are merged together and result in a tree shaped automaton. Each node in the tree is labeled by an output symbol $\sigma \in \Sigma_O$, and each edge is labeled by an input action $\alpha \in \Sigma_I$. Every path from the root to a node corresponds to a string $s \in \text{prefix}(S)$. The node s is associated with the frequency function $f(s,\alpha,\sigma)$ $(\alpha \in \Sigma_I,\ \sigma \in \Sigma_O)$ which is the number of strings in S with the prefix $s\alpha\sigma$, and $f(s,\alpha) = \sum_{\sigma\in\Sigma_O} f(s,\alpha,\sigma)$. From one node in IOFPTA, given an input action and an output symbol, the next state can be uniquely determined. An IOFPTA can be transformed to DLMDP by normalizing frequencies $f(s,\alpha,\cdot)$ to $\tau(s,\alpha,\cdot)$. As assumed in data generation phase, when the scheduler chooses a disabled input on a state in LMDP, the model will stay in the current state, and output the symbol *err*. We are going to take the special meaning of the *err* symbol into account in the IOFPTA construction. Specifically, s and $s\alpha err$ would lead to the same state from the root state. We will take the special treatment for the err symbol, but there is no difference between it and other symbols in learning. A new node labeled by *err* will not be created as a successor node or we can say that the *err* nodes are folded up.

Example 1 IOFPTA

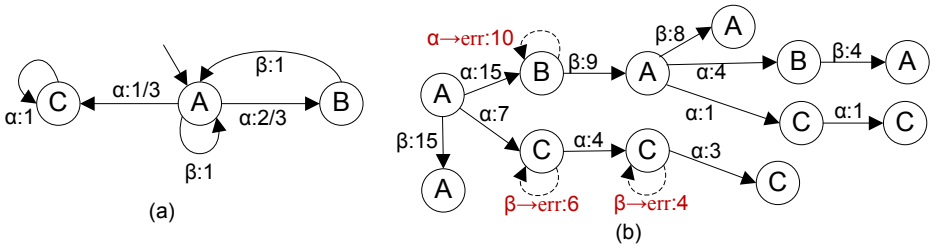

Figure 1: (a) A DLMDP over $\Sigma_O = \{A,B,C,err\}$ and $\Sigma_I = \{\alpha,\beta\}$; (b) The corresponding IOFPTA.

The IOFPTA *in Fig.1(b) is constructed from sample sequences generated by a DLMDP M in Fig.1(a). The root node is labeled by A. From the root, given input α, successor nodes which are labeled by B and C, will be reached by strings with the prefix $A\alpha B$ or $A\alpha C$, respectively. For the frequency, $f(A,\alpha,B) = 15$ and $f(A,\alpha,C) = 7$. The input action β is disabled in the state with label C of (a). Then the tree will stay*

in node labeled by C when we meet the input β (which is drawn and linked by dash lines in Fig.1(b)). Then for each node the incoming frequencies are not equivalent to the outgoing frequencies.

4.2 IOALERGIA

IOALERGIA algorithm, is an adapted version of the ALERGIA algorithm [5, 8]. As seen in Example 1, the same state in generating LMDP could be reached by more than one sequences through running, which will create more than one node in the IOFPTA. The basic idea of this learning algorithm is to approximate the generating model by grouping together the nodes in IOFPTA which can be mapped to the same state in the generating model. The partition which is introduced by grouping nodes will be inferred by pairwise testing. The *compatibility* of two nodes is tested by comparing distributions defined by nondeterministic choices, and recursively testing on successor nodes. If two nodes in the tree pass the *compatibility* test which means they can be mapped to the same state in the generating model, then they will be merged, as well as their successor nodes.

Algorithm 1 IOALERGIA

Input: : A dataset S and a parameter $\varepsilon \in (0,1]$;
Output: : A DLMDP A;
1: $T, A \leftarrow \text{IOFPTA}(S)$;
2: $\text{RED} \leftarrow q_s^A$;
3: $\text{BLUE} \leftarrow \{q \mid q = q_s^A \alpha\sigma, \alpha \in \Sigma_I, \sigma \in \Sigma_O, q_s^A \alpha\sigma \in \text{prefix}(S)\}$; /* immediate successor states */
4: **while** $\text{BLUE} \neq \emptyset$ **do**
5: $q_b \leftarrow$ lexicographically minimal $q \in \text{BLUE}$;
6: *merged* $\leftarrow$ *false*;
7: **for** $q_r \in \text{RED}$ /* in lexicographic order */ **do**
8: **if** *Compatible*$(T, q_r, q_b, \varepsilon)$ **then**
9: $A \leftarrow$ *Merge*(A, q_r, q_b);
10: *merged* $\leftarrow$ *true*;
11: **end if**
12: **end for**
13: **if** !*merged* **then**
14: $\text{RED} \leftarrow \text{RED} \cup \{q_b\}$;
15: **end if**
16: $\text{BLUE} \leftarrow \text{BLUE} \setminus \{q_b\} \cup \{q = q_r \alpha\sigma \mid \alpha \in \Sigma_I, \sigma \in \Sigma_O, q \in \text{prefix}(S), q_r \in \text{RED}, q \notin \text{RED}\}$;
17: **end while**
18: **return** *makeDLMDP*(A); /* normalize */

In the learning algorithm, firstly, two IOFPTAs T and A are constructed as the representation of the dataset S (line 1 of the Algorithm 1). The IOFPTA T is kept as a data representation from which relevant statistics are retrieved during the execution of the algorithm. The IOFPTA A is iteratively transformed by merging nodes which have passed the *compatibility* test. All compatibility is tested on T, and the reason for this is that it has a clear interpretation as empirical probabilities defined by the data S. Following the terminology from [8], Algorithm 1 maintains two sets of states: RED states, which have already been determined as representative states of partitions and will be included in the final output DLMDP, and BLUE states which are going to be tested. Initially, RED contains only the initial state while BLUE contains the immediate successor states of the initial state. During iterations, the lexicographically minimal node q_b in BLUE will be chosen. If there exists a state q_r in RED which is

compatible with q_b, then q_b and its successor nodes are going to be merged into q_r and its corresponding successor states. If q_b is not compatible with any state in RED, it will be included in RED. At the end of each iteration, BLUE is going to be updated as the margin between RED and the remaining states, in the other word, the set of states which are immediate successor states of RED but not included in it. After merging all possible compatible nodes in the tree, the frequencies in A are going to be normalized by the Algorithm 1 (line 18). Then a DLMDP is constructed.

4.3 Compatibility Test

Algorithm 2 demonstrates the *compatibility* test. It will return true if two nodes are compatible, i.e., the distance of distributions for every action is within the Hoeffding bound [19], Algorithm 3, parameterized by ε. Formally, two nodes q_r and q_b are *ε-compatible* ($1 \geq \varepsilon > 0$), if it holds that:

1. $L(q_r) = L(q_b)$
2. *Hoeffding*$(f(q_r,\alpha,\sigma), f(q_r,\alpha), f(q_b,\alpha,\sigma), f(q_b,\alpha),\varepsilon)$ is TRUE, for all $\alpha \in \Sigma_I$ and $\sigma \in \Sigma_O$.
3. Nodes $q_r\alpha\sigma$ and $q_b\alpha\sigma$ are *ε-compatible*, for all $\alpha \in \Sigma_I$, and $\sigma \in \Sigma_O$

Condition 1) requires two nodes in the tree to have the same label. Condition 2) defines the compatibility between each outgoing transition with the same input action respectively from state q_r and q_b. The last condition requires the compatibility to be recursively satisfied for every pair of successors of q_r and q_b. If two nodes in IOFPTA are compatible, then distributions for all input actions should pass the *compatibility* test.

In the original ALERGIA algorithm, termination probabilities of two nodes are compared, while not in Algorithm 2. The reason is that the termination probability is not included in the definition of DLMDP. In Algorithm 3, the distance of two empirical probabilities are compared with the *Hoeffding* bound. If there is few, even none, statistical evidence to support their difference, the distance is small. In particular, two nodes are compatible, if there is no evidence against that. The *err* information is used to discriminate two nodes which have different enabled actions. For example, there are q_1 and q_2, and input action α is only enabled on q_1. For q_1, $f(q_1,\alpha,\sigma) > 0, \sigma \neq err$ and $f(q_1,\alpha,err) = 0$, while $f(q_2,\alpha) = f(q_2,\alpha,err) > 0$. Comparing the empirical probability distribution over Σ_O including *err*, q_1 and q_2 can not be compatible.

4.4 Merge states

If two states q_r and q_b are *compatible*, q_b will be merged to q_r. The *Merge* procedure (line 9 of the Algorithm 1) follows the same way as described in [8]: firstly, the (unique) transition leading to q_b from its predecessor node q' ($f^A(q',\alpha,q_b) > 0$) is re-directed to q_r by setting $f^A(q',\alpha,q_r) \leftarrow f^A(q',\alpha,q_b)$ and $f^A(q',\alpha,q_b) = 0$. Then, successor nodes of q_b will be recursively folded to the corresponding successor nodes of q_r.

Example 2 *Merge States*

Fig. 2 shows the procedure that the node q_b (shadowed) will be merge to the node q_r (shadowed double circle). In (a), the transition from the node q' to q_b firstly redirected to q_r. In (b), transitions from q_b to three successor nodes labeled with A, B and C, will be folded into the corresponding successor nodes of q_r, respectively. (c) illustrates the result after merge.

Algorithm 2 *Compatible*

Input: : IOFPTA T , nodes q_r and q_b, $\varepsilon \in (0,1]$
Output: : *true* if q_r and q_b are compatible

if $L(q_r) \neq L(q_b)$ **then**
  return *false*
end if
for $\alpha \in \Sigma_I$ **do**
  for $\sigma \in \Sigma_O$ **do**
    if !*Hoeffding*$(f^T(q_r,\alpha,\sigma), f^T(q_r,\alpha)f^T(q_b,\alpha,\sigma), f^T(q_b,\alpha),\varepsilon)$ **then**
      return *false*
    end if
    if !*Compatible*$(T, q_r\alpha\sigma, q_b\alpha\sigma,\varepsilon)$ **then**
      return *false*
    end if
  end for
end for
return *true*

Algorithm 3 *Hoeffding*

Input: : $f_1, n_1, f_2, n_2, \varepsilon \in (0,1]$
Output: : *true* if f_1/n_1 and f_2/n_2 are sufficiently close

if $n_1 == 0$ or $n_2 == 0$ **then**
  return *true*
end if
return $|\frac{f_1}{n_1} - \frac{f_2}{n_2}| < (\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_2}}) \cdot \sqrt{\frac{1}{2}\ln\frac{2}{\varepsilon}}$

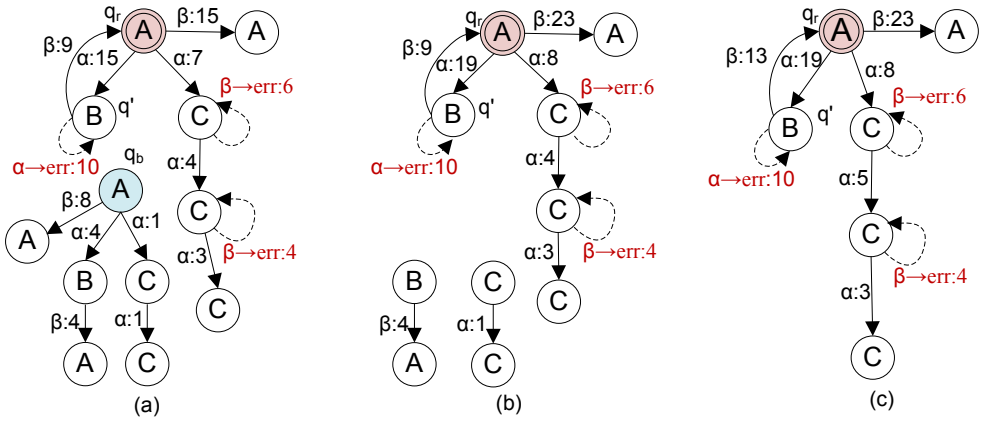

Figure 2: Merge states q_r and q_b

4.5 Discussion

The algorithm takes a set S of sample sequences and a parameter ε as inputs. Here ε is used to bound the type-I error, which is the probability of wrongly rejecting a correct compatibility hypothesis. Smaller

values of ε lead to loose Hoeffding bounds and making IOALERGIA output a smaller model. For any particular finite sample size we try to tune the choice of ε so as to obtain the best approximation to the real model. In order to do this we run IOALERGIA with different ε values, and evaluate the learned model using the *Bayesian Information Criterion (BIC)* score. This score combines the likelihood of a model with a term penalizing model complexity. Concretely, the BIC score of a DLMDP A given data S is defined as

$$BIC(A \mid S) := log(P_A(S)) - 1/2 \, |A| \, log(N)$$

where $|A| = |Q| \cdot |\Sigma_I| \cdot |\Sigma_O|$ is the number of free parameters in the model. N is the number symbols in the data. Using a golden section search [18, Section E.1.1] we systematically search for an ε value maximizing the BIC score of the learned model. Our algorithm is implemented in Matlab and is available for download at `http://mi.cs.aau.dk/code/ioalergia`.

A convergence analysis, similar to the analysis in [7, 13] for deterministic Markov chain models, can be obtained for IOALERGIA: first, one can show that in the large sample limit, IOALERGIA will identify up to bisimulation equivalence the structure of the true model from which the data was sampled; the structure of a model refers to all of its components, except the probability values of transitions. Second, the parameters in the learned model will converge to the corresponding parameter values in the true model. As a slight refinement of Theorem 2 in [13], one then obtains that for any LTL formula φ:

$$P(\lim_{n\to\infty} P^{\max}_{A^n}(\varphi) = P^{\max}_{M}(\varphi)) = 1, \text{and} \qquad P(\lim_{n\to\infty} P^{\min}_{A^n}(\varphi) = P^{\min}_{M}(\varphi)) = 1;$$

where A^n is the DLMDP returned by IOALERGIA on data $S[n]$. As also observed in [13], similar results do not carry over to PCTL formulas.

5 Experiments

In this section, we are going to show the applicability of the IOALERGIA algorithm using a case study based on the slot machine [9]. The slot machine we considered has 3 reels, named as *reel*-1, *reel*-2 and *reel*-3, and each reel contains 5 different symbols: *lemon*, *grape*, *cherry*, *bar* and, *apple*. The slot machine will return a prize based on the combination of symbols on those 3 reels. The prizes for different configurations are shown in Table 1(a). We extend the basic gambling machine as follows: at each round the player can choose one of the reels to spin, and other reels will be kept. The player starts with paying 1 coin for first 3 spins, and afterwards each extra spin costs 1 additional coin. Each reel must be spun at least once, and the player can quit the game only if all reels have been spun. The behavior of the slot machine contains both probabilistic and nondeterministic aspects. Specifically, the symbol show for each reel is probabilistic, but the choice of which reel to spin is nondeterministic.

In the following parts of this section, the algorithm will be applied for learning deterministic and nondeterministic models for different number of spins. A memoryless and random scheduler with a uniform distribution over all input actions, that modeling the *fair* requirement, is used in the data generation procedure. For experiment, we analyze the behavior of learned models by comparing them with known generating models in terms of maximal and minimal probabilities of winning a specific reward as well as the maximal expected reward in general. These probabilities and rewards are all computed by PRISM [10]. We will also analyze the accuracy that the optimal action in the learned model given symbols on reels and number of times the reels have been spun.

Table 1: Prize

reel-1	*reel*-2	*reel*-3	Prize
bar	bar	bar	10
cherry	cherry	cherry	10
grapes	grapes	grapes	10
?	bar	bar	5
cherry	?	cherry	5
grapes	grapes	?	5
?	?	bar	2
?	?	cherry	1

Table 2: Summery of slot machines

	Deterministic Slot Machine		Nondeterministic Slot Machine	
N	$\|Q\|$	|Tran|	$\|Q\|$	|Tran|
4	437	4021	510	4959
6	867	10721	1012	13291
8	1297	17421	1514	21623
10	1727	24121	2016	29955

5.1 Learning models from Deterministic systems

We implemented the slot machine in PRISM. The distribution for 3 reels showing different symbols are $(0.2,0.2,0.1,0.3,0.2)$, $(0.2,0.1,0.3,0.2,0.2)$, and $(0.2,0.3,0.2,0.1,0.2)$, respectively. In this model, there are 4 actions: spin *reel*-1 (sp_1), spin *reel*-2 (sp_2), spin *reel*-3 (sp_3), and get the prize *(pay)*, thus $\Sigma_I = \{sp_1, sp_2, sp_3, pay\}$. Every state is labeled by the combination of states on the 3 reels and the number of times the reels have been spun. We also attached reward variables to the states which are labeled by *prize*. Table 2 shows statistics for models with various number of spins. Here, N ($N \geq 3$) is the number of spins, $|Q|$ is the number of states, and |Tran| is the number of transitions.

The generating model is a deterministic LMDP. The results of applying the learning algorithm for different data sets are produced by the generating model are summarized in Table 3: $|S|$ is the number of symbols in the dataset ($\times 10^3$), |Seq| is the number of sequences in the dataset; |IOFPTA| is the number of nodes in the IOFPTA; Time is the learning time (in seconds), including the time for constructing IOFPTA and the average time for each iteration performed by the golden section search (typically the golden section search terminated after 14 to 19 iterations); 'ε range' is the interval (identified using the golden section search) for ε for which a BIC-optimal DLMDP is learned, $|Q|$ is the number of states in the learned model.

Fig. 3(a) and (b) show the maximal and minimal probabilities of eventually getting different prizes using the $P^{\max}(\Diamond \mathrm{L}\ coins)$ and $P^{\min}(\Diamond \mathrm{L}\ coins)$, where $\mathrm{L} \in \{0,1,2,5,10\}$ (on both generating model and learned models, for $N = 4,6,8,10$). As the size of dataset increases, the learned models provide better approximations of the maximal and minimal probabilities defined for the generating models. Using PRISM, the maximal expected reward for one gamble ($R^{\max}(\Diamond\ stop)$) can be computed. In Fig. 3(c), for various initially bought spin chances, the maximal expected rewards for the learned models (dashed lines) are all approaching the ones for the generating models as the sizes of the datasets increase.

The optimal action which reel to spin next for a specific configuration of the reels, can also be accurately preserved by learned models. For example, given that there are three *apples* on reels and we only have 1 spin left, the best choice is to spin the 3rd reel since taking any other action will not produce a prize. We consider the 125 configurations where every reel has been spun once. Given a specific configuration C_i, the optimal action in the learned model and the generating model are denoted as Act_i^l and Act_i^g, respectively. We define a criterion which interpret the accuracy of optimal actions inferred by the learned model against the generating model as follows:

$$Acc = \sum_{i=1}^{125} P^{\max}(C_i) \cdot \frac{|Act_i^l \cap Act_i^g|}{|Act_i^l|}$$

Table 3: Experimental results for the slot machine models.

	$\|S\|(\times 10^3)$	|Seq|	|IOFPTA|	Time	ε range	$\|Q\|$
$N=4$	160	5832	20915	9.8	[0.0020; 0.1552]	436
	640	23246	48373	29.9	[0.0020; 0.1552]	437
	1280	46374	64064	50.2	[0.0020; 0.1250]	437
$N=6$	160	5779	33829	16.0	[0.0020; 0.1553]	866
	640	23154	122458	46.9	[0.0020; 0.1553]	867
	1280	46273	231029	84.9	[0.0010; 0.0776]	867
$N=8$	640	23054	148225	66.1	[0.0020; 0.1553]	1297
	1280	46242	283749	116.6	[0.0010; 0.0776]	1297
	2000	72284	429555	153.0	[0.0010; 0.0776]	1297
$N=10$	1280	46241	317794	142.5	[0.0005; 0.0388]	1725
	2000	72250	482943	184.0	[0.0005; 0.0313]	1727
	5000	180755	1135055	454.4	[0.00006; 0.0040]	1727

Where, $P^{\max}(C_i)$ is the maximal probability of reaching configuration C_i. As shown in Fig. 3 (d), by increasing the size of dataset, the learned models have almost the same optimal actions as the generating models. Even with very limited data amount, accuracies for optimal actions in learned models are always greater than 25%, which is the probability of randomly choosing an optimal action.

5.2 Learning models from Nondeterministic systems

In order to make the slot machine more interesting, we increase the prize for three *bar*s but reduce the probability of getting that. This is done by adding another *bar* on *reel*-2, two *bar*s, denoted as b_1 and b_2, that are indistinguishable, but have different mechanical characteristics. The probability for these two bars depend on the symbols on other two reels.

The distributions for all reels are shown in Table 4(a) and Table 4(b). Since reels are no longer independent, we name refer to machine as *hooked slot machine*. In this machine, the probability of getting 3 *bar*s is decreased, but the reward for getting 3 bars is 20 coins. Every other configuration has the same prize as the previous game. After this modification, the generating model becomes nondeterministic, and its statistics listed in Table 2.

Table 4: Probability distributions for 3 reels

(a) Probability distributions for the 1st and the 3rd reel

		lemon	grape	cherry	bar	apple
*reel*1	$r_2 = b_1$	0.2	0.2	0.1	0.3	0.2
	$r_2 = b_2$	0.3	0.2	0.1	0.05	0.35
	other	0.25	0.2	0.1	0.15	0.3
*reel*3	$r_2 = b_1$	0.2	0.3	0.2	0.05	0.25
	$r_2 = b_2$	0.1	0.3	0.2	0.3	0.1
	other	0.2	0.3	0.2	0.15	0.15

(b) Probability distributions for 2nd reel

	$r_1 = b$	$r_3 = b$	$r_1, r_3 = b$	other
lemon	0.2	0.2	0.26	0.2
grape	0.1	0.1	0.1	0.1
cherry	0.3	0.3	0.3	0.3
bar 1	0.18	0.02	0.02	0.1
bar 2	0.02	0.18	0.02	0.1
apple	0.2	0.2	0.3	0.2

In this experiment, we apply IOALERGIA for learning DLMDPs from data generated by the nondeterministic

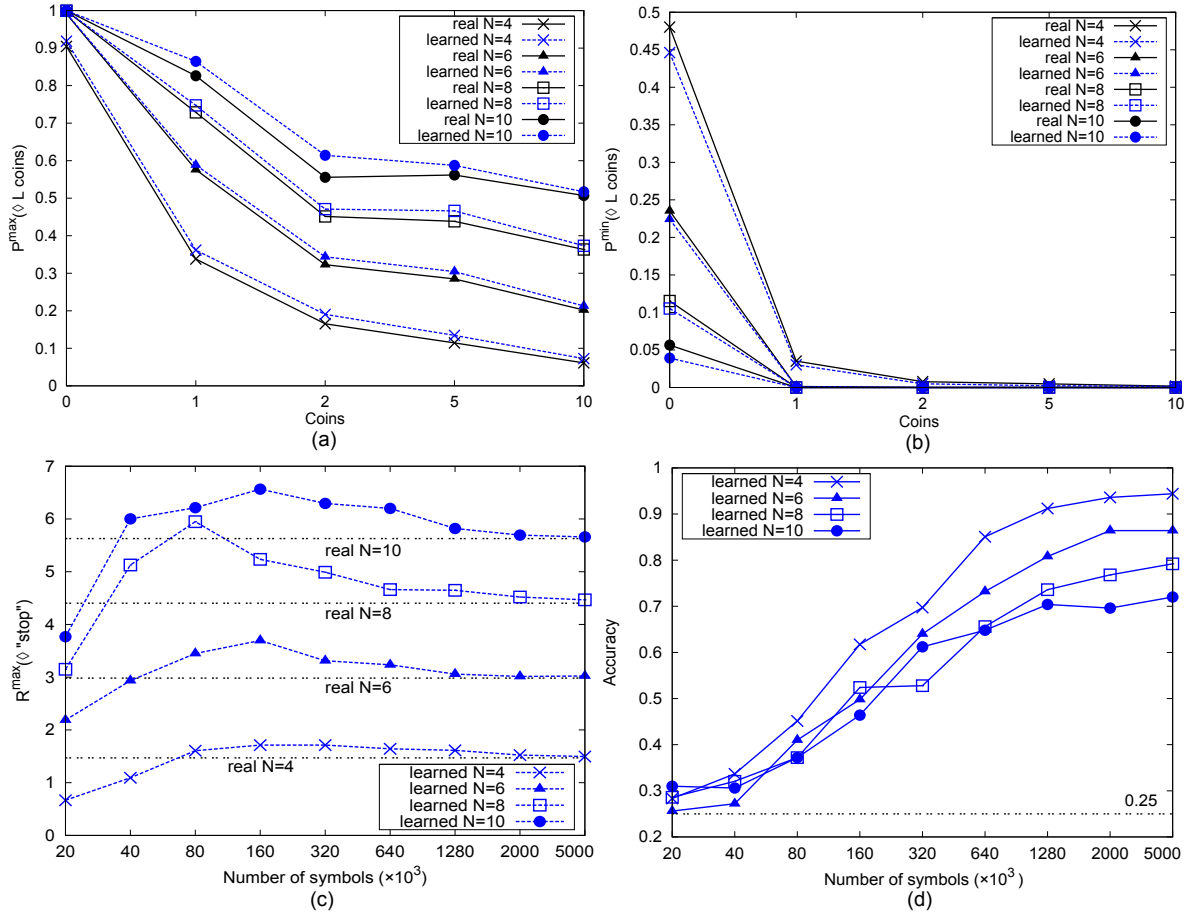

Figure 3: Evaluation results for learning deterministic models. Fig (a) and (b): The maximal and minimal probabilities of eventually being awarded L coins given 4, 6, 8, and 10 initial spins, here L$\in \{0,1,2,5,10\}$. As shown, the model for $N = 4$ is learned from 1280×10^3 symbols, and models for $N = 6,8,10$ are all learned from 5000×10^3 symbols. Fig (c), shows maximal rewards ($R^{\max}(\Diamond\, stop)$) in learned models and the generating model. In Fig (d), the accuracy of the optimal action the learned models is shown.

models. The learning results are summarized in Table 5, where each column has the same meaning as in Table 3. Given sufficient dtat, we observed that learned models have the same number of states as the deterministic models of the previous slot machine, thus the states introduced by the extra symbol on *reel*-2 was not get identified. The reason is that states labeled by b_1 *on reel-2* and b_2 *on reel-2* are mixed and generally observed as *bar on reel-2*.

Fig. 4 shows maximal and minimal probabilities for getting different prizes, maximal rewards from the initial state and the accuracy of the optimal action. Given adequate data, learned deterministic models provide good approximations for nondeterministic generating models in terms of maximal probability, minimal probability and the maximal expected reward. On the other hand, the accuracy of choosing optimal action in next step is no longer as good as before. Nevertheless, the suggestion given by learned model is still better than random choice (which has 25% accuracy) in most cases.

The generating model is a nondeterministic LMDP, so there is no guarantee that the learned model

Table 5: Experimental results for hooked slot machines.

	$\|S\|(\times 10^3)$	|Seq|	|Tree|	time	ε range	$\|Q\|$
$N=4$	160	5794	20768	9.7	[0.0020; 0.1552]	437
	640	23185	48530	29.8	[0.0020; 0.1250]	437
	1280	46308	64354	51.5	[0.0010; 0.0776]	437
$N=6$	160	5737	33755	15.7	[0.0039; 0.2500]	867
	640	23174	122575	46.6	[0.0020; 0.1552]	867
	1280	46380	231260	84.0	[0.0005; 0.0388]	867
$N=8$	640	23143	148730	63.7	[0.0020; 0.1552]	1297
	1280	46260	284310	112.7	[0.0010; 0.0776]	1297
	2000	72212	430102	166.1	[0.0005; 0.0313]	1297
$N=10$	1280	46371	318423	138.6	[0.0010; 0.0776]	1723
	2000	72360	483696	202.5	[0.0005; 0.0313]	1724
	5000	180781	1135149	460.3	[0.0010; 0.0625]	1725

preserves all PLTL properties . For example, suppose there are two *bar*s after two spins, and corresponding to the configurations '*bar*, *bar*, *not-spun*' (C_1), '*bar*, *not-spun*, *bar*' (C_2), and '*not-spun*, *bar*, *bar*' (C_3). From these configurations, we can calculate the maximal probability of getting 3 *bar*s after next spin (see Table 6). The maximal probability in the generating model for different N are the same since there is still one reel is that has not been spun. We can observe that conditional probabilities in learned models are quite different from the ones in generating models.

Table 6: conditional probability

	real	N=4	N=6	N=8	N=10
$P(3\times bars \mid C_1)$	0.30	0.0714	0.0356	0.0327	0.0450
$P(3\times bars \mid C_2)$	0.04	0.0551	0.0659	0.0934	0.0701
$P(3\times bars \mid C_3)$	0.30	0.0940	0.0835	0.0874	0.0885

6 Conclusion

In this paper, we have proposed the IOALERGIA algorithm for learning deterministic labeled Markov processes (DLMDPs). Given sequences of alternating input and output symbols, the algorithm can automatically construct a model, for the reactive system under observation, and we have similar convergence result of the IOALERGIA algorithm as given in [13] for deterministic Markov chain models. The algorithm is empirically analyzed using a case study based on slot machines. The learning results are evaluated by comparing in terms of PLTL properties and maximal expected rewards of both the learned model with the known generating models as well as the accuracy of optimal actions derived from the learned models.

Compared to the learning algorithm for deterministic automata [14], further research is required to make the learning algorithm that suitable for routine use. In addition to empirically demonstrating the learned model is a good approximation, measuring the distance between the learned model and the generating model will be part of our future work. For compositional systems, this learning approach

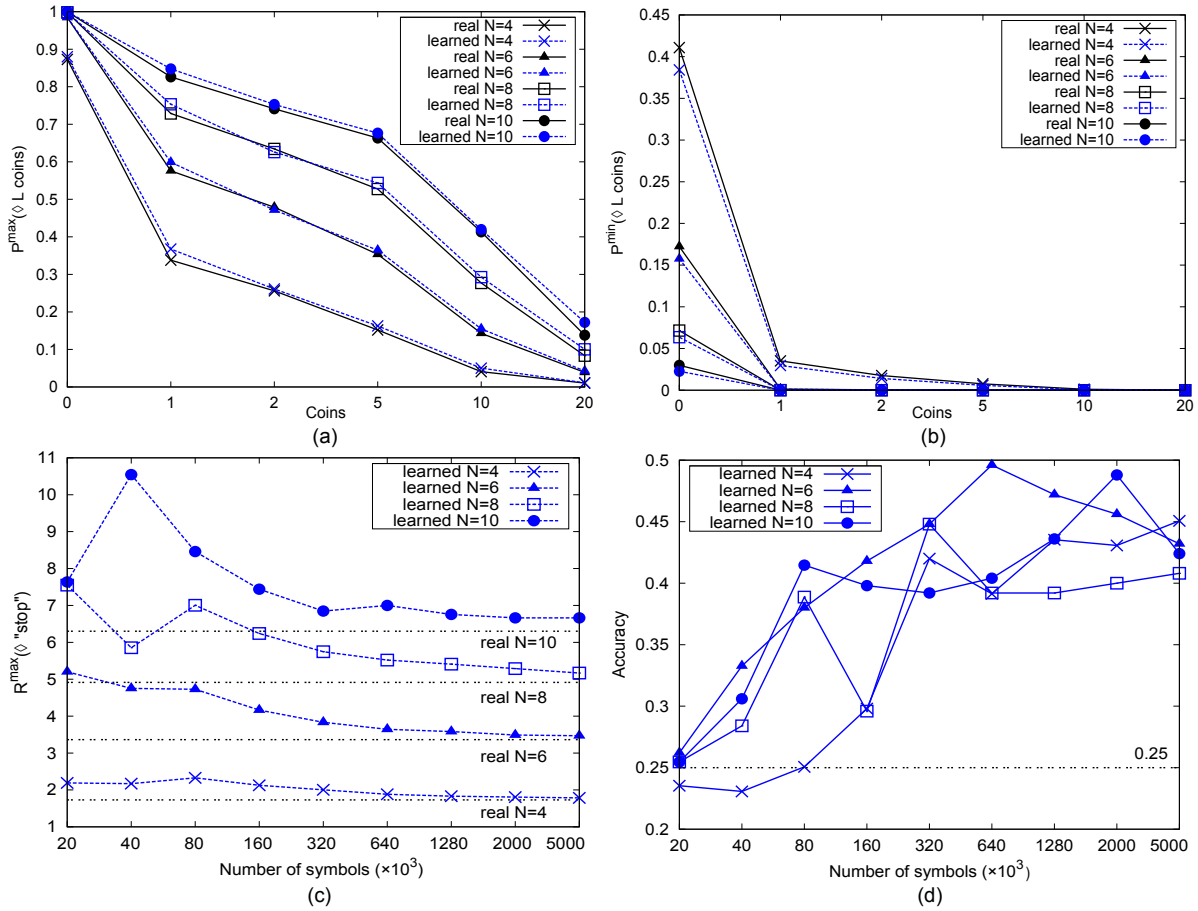

Figure 4: Evaluation results for learning nondeterministic models. (a) and (b) are the maximal and minimal probability of eventually being awarded L coins $L \in \{0,1,2,5,10\}$. The size of each dataset is the same as Fig. 3. (c): maximal rewards computed by $R^{\max}(\Diamond\, stop)$ in learned models and generating models. (d): the accuracy of optimal actions suggested by learned models.

could be extended to learn models for each individual component from the observed interaction among components. Moreover, the approach for learning DLMDP could be refined by *active learning* techniques that take advantage of interactive data acquisition.

References

[1] Fides Aarts & Frits W. Vaandrager (2010): *Learning I/O Automata*. In: *CONCUR*, pp. 71–85. Available at `http://dx.doi.org/10.1007/978-3-642-15375-4_6`.

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

[3] Christel Baier & Joost-Pieter Katoen (2008): *Principles of model checking*. MIT Press.

[4] J. Bogdoll, L. M. F. Fioriti, A. Hartmanns & H. Hermanns (2011): *Partial Order Methods for*

Statistical Model Checking and Simulation. In: *FMOODS/FORTE*, pp. 59–74. Available at http://dx.doi.org/10.1007/978-3-642-21461-5_4.

[5] R. C. Carrasco & J. Oncina (1994): *Learning Stochastic Regular Grammars by Means of a State Merging Method*. In: *ICGI*, pp. 139–152. Available at http://dx.doi.org/10.1007/3-540-58473-0_144.

[6] Y. Chen, H. Mao, M. Jaeger, T. D. Nielsen, K. G. Larsen & B. Nielsen (2012): *Learning Markov Models for Stationary System Behaviors*. In: *NFM*, pp. 216–230. Available at http://dx.doi.org/10.1007/978-3-642-28891-3_22.

[7] Colin de la Higuera & Franck Thollard (2000): *Identification in the Limit with Probability One of Stochastic Deterministic Finite Automata*. In: *ICGI*, pp. 141–156. Available at http://dx.doi.org/10.1007/978-3-540-45257-7_12.

[8] Colin de la Higuera (2010): *Grammatical Inference — Learning Automata and Grammars*. Cambridge University Press.

[9] D. N. Jansen (2002): *Probabilistic UML Statecharts for Specification and Verification a Case Study*. In: *Critical Systems Development with UML – Proc. of the UML'02 workshop*, pp. 121–132.

[10] M. Kwiatkowska, G. Norman & D. Parker (2011): *PRISM 4.0: Verification of Probabilistic Real-time Systems*. In: *CAV*, *LNCS* 6806, Springer, pp. 585–591.

[11] A. Legay, B. Delahaye & S. Bensalem (2010): *Statistical Model Checking: An Overview*. In: *RV*, pp. 122–135. Available at http://dx.doi.org/10.1007/978-3-642-16612-9_11.

[12] Martin Leucker (2006): *Learning Meets Verification*. In: *FMCO*, pp. 127–151. Available at http://dx.doi.org/10.1007/978-3-540-74792-5_6.

[13] H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen & B. Nielsen (2011): *Learning Probabilistic Automata for Model Checking*. In: *QEST*, pp. 111–120. Available at http://doi.ieeecomputersociety.org/10.1109/QEST.2011.21.

[14] H. Raffelt & B. Steffen (2006): *LearnLib: A Library for Automata Learning and Experimentation*. In: *FASE*, pp. 377–380. Available at http://dx.doi.org/10.1007/11693017_28.

[15] D. Ron, Y. Singer & N. Tishby (1996): *The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length*. *Machine Learning* 25(2-3), pp. 117–149. Available at http://dx.doi.org/10.1023/A:1026490906255.

[16] K. Sen, M. Viswanathan & G. Agha (2004): *Learning Continuous Time Markov Chains from Sample Executions*. In: *QEST*, pp. 146–155. Available at http://doi.ieeecomputersociety.org/10.1109/QEST.2004.10014.

[17] Mariëlle Stoelinga (2002): *An Introduction to Probabilistic Automata*. *Bulletin of the EATCS* 78, pp. 176–198.

[18] P.-N. Tan, M. Steinbach & V. Kumar (2006): *Introduction to Data Mining*. Addison Wesley.

[19] H. Wassily (1963): *Probability Inequalities for Sums of Bounded Random Variables*. *Journal of the American Statistical Association* 58(58), pp. 13–30. Available at http://dx.doi.org/10.2307/2282952.

[20] H. L. S. Younes & R. G. Simmons (2002): *Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling*. In: *CAV*, pp. 223–235. Available at http://dx.doi.org/10.1007/3-540-45657-0_17.