Predictive Run-Time Verification of Discrete-Time Reachability Properties in Black-Box Systems Using Trace-Level Abstraction and Statistical Learning

Reza Babaee(✉), Arie Gurfinkel, and Sebastian Fischmeister

Electrical and Computer Engineering, University of Waterloo, Waterloo, Canada
{rbabaeec,arie.gurfinkel,sebastian.fischmeister}@uwaterloo.ca

Abstract. Run-time Verification (RV) has become a crucial aspect of monitoring black-box systems. Recently, we introduced $\mathcal{P}revent$, a predictive RV framework, in which the monitor is able to predict the future extensions of an execution, using a model inferred from the random sample executions of the system. The monitor maintains a table of the states of the prediction model, with the probability of the extensions from each state that satisfy a safety property.

The size of the prediction model directly influences the monitor's memory usage and computational performance, due to the filtering techniques used for run-time state estimation, that depends on the size of the model. Hence, achieving a small prediction model is key in predictive RV.

In this paper, we use symmetry reduction to apply abstraction, that, in the absence of a model in black-box systems, is performed on the observation space. The symmetry relation is inferred based on k-gapped pair model, that lumps symbols with similar empirical probability to reach a set of target labels on a set of samples. The obtained equivalence classes on the observation space are used to abstract traces that are used in training the prediction model.

We demonstrate the soundness of the abstraction, in the case that the generating abstract model is a deterministic Discrete-Time Markov Chain (DTMC). We use Hidden Markov Models (HMMs) to handle the abstraction-induced non-determinism by learning the distribution of a hidden state variable. We implemented our approach in our tool, $\mathcal{P}revent$, to empirically evaluate our approach on the Herman's randomised self-stabilising algorithm. Our results show that the inferred abstraction significantly reduces the size of the model and the training time, without a meaningful impact on the prediction accuracy, with better results from the HMM models.

1 Introduction

Run-time Verification (RV) [28] has become a crucial element in monitoring and analysing safety aspects of black-box stochastic systems [29,39], where there is almost surely a non-zero probability of failure. In RV, a monitor checks the current execution, that is a finite prefix of an infinite path, against a given property,

C. Colombo and M. Leucker (Eds.): RV 2018, LNCS 11237, pp. 187–204, 2018.
https://doi.org/10.1007/978-3-030-03769-7_11

typically expressed in Linear Temporal Logic (LTL) [36], that represents a set of acceptable infinite paths. If any infinite extension of a prefix belongs (does not belong) to the set of infinite paths that satisfy the property, the monitor accepts (resp. rejects) the prefix. However, if the monitor is not able to reach a verdict with the given prefix because it can be extended to satisfy or violate the property, the monitor outputs *unknown* [5].

We introduced the predictive RV framework [3], where the monitor finitely extends the prefix based on a *prediction model* that is trained on the set of *independent and identically distributed* (*iid*) sample traces. This gives the monitor the ability to detect a monitorable [14] property's satisfaction/violation before its occurrence. In this paper, we propose inferring trace-level abstraction to reduce the size of the prediction model. Our focus is on the discrete-time reachability properties, specified as *target labels* on the observation space Σ. The maximum length of the extensions is specified as the *prediction horizon* (Fig. 1).

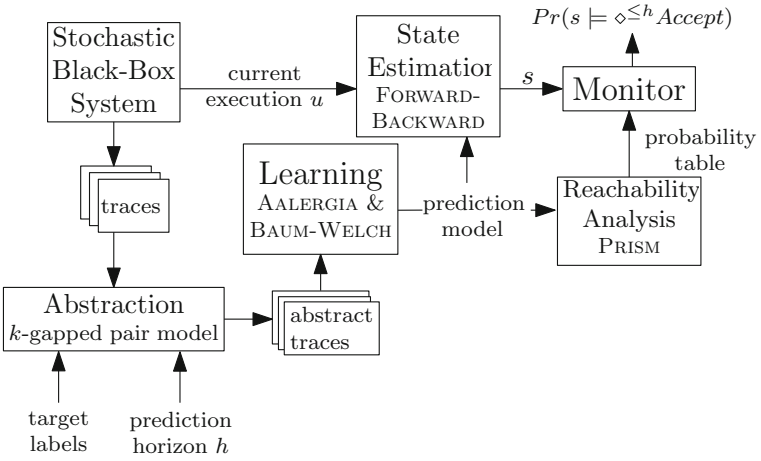


Fig. 1. Overview of predictive RV with abstraction.

Abstraction. The key idea in our abstraction is to decrease the size of the observation space by extracting symbols that have similar *transient probability* [24] to reach the target labels. Therefore, our approach can be seen as a form of symmetry reduction [23] on the observation space that is implemented at the trace-level [33]. We use *k-gapped pair model* [13] to detect the symbols that have similar empirical probability to reach the target symbols within k steps. The symbols that have probability *zero* are considered irrelevant in the prediction. The *symmetrical* symbols are then lumped into equivalence partition over the observation space to convert the traces of the training set into *abstract* traces.

Learning. If there is no non-determinism in the generating model of the abstract traces, given that the property is a bounded reachability LTL formula, any learning technique to infer the deterministic model, such as Aalergia [30], suffices to

provide the prediction model that in the limit is as accurate as the actual model. However, the trace-level abstraction may induce non-determinism in the trained model [33]. We use Hidden Markov Model [37] (HMM) to infer the induced non-determinism by the abstraction, and compare the results with the deterministic as well as the actual model [30]. We train an HMM using Baum-Welch algorithm [37], that is an approximation to find the parameters of the HMM that maximizes the likelihood of the training data.

Reachability Analysis and Monitor Construction. The monitor in our framework is the result of a bounded reachability analysis on the prediction model. We use Prism [34] to perform the probabilistic bounded reachability analysis on the prediction model. The monitor is implemented as a lookup table, where each entry is a combination of three elements: an integer variable t, a state of the prediction model, and the probability that from that state the system reaches the states with the target labels, within t steps. The value of t is constrained by the prediction horizon that is expected from the prediction model.

State Estimation. Given that the original DTMC is deterministic, the system state can be determined by the observed path u. However, if the prediction model is non-deterministic, the state of the system needs to be estimated [40] based on the prefix u. Any *filtering techniques* [38] can be exploited to estimate the state at run-time. If the size of the model is large, approximate techniques such as the Viterbi algorithm [3,43] can be applied too. Since the purpose of the current paper is to reduce the size of the prediction model, and also for evaluation purposes, we apply a direct approach and compute the posterior probability distribution of the states in the prediction model after observing u, using Forward-Backward algorithm [37].

The output of the monitor is the probability that from the estimated state s the system reaches the target labels within at most h steps. This probability is retrieved from the probability table after estimating the current state. Respectively, the size of the prediction model dictates both the size and the computational overload of the monitor.

In summary, our paper makes the following contributions:

- define trace-level abstraction as a symmetry relation in the observation space, and infer it using k-gapped pair model
- resolve non-determinism induced by the abstraction using hidden state variable
- demonstrate the validity of our approach on a distributed randomized algorithm, by showing the significant reduction in the prediction models with minimum impact on the prediction accuracy.

The rest of the paper is organized as follows: in Sect. 2 we introduce our running example, followed by preliminaries in Sect. 3, and a brief description of the prediction procedure in Sect. 4. In Sect. 5 we explain the trace-level abstraction using symmetry relation on the observation space and our algorithm to infer the

symmetry. Learning deterministic models, and some theoretical guarantee, as well as learning non-deterministic models are described in Sect. 6. We conclude the paper by discussing the results of our case study in Sect. 7.

2 Running Example

We use the die example [34] as the running example throughout the paper. This example demonstrates the simulation of throwing a fair 6-sided die with flipping a fair coin [22]. Let C be the output of the flipped coin ($C \in \{ii, hh, tt\}$), where hh, tt display *head* and *tail*, resp., and ii is a special symbol to indicate the initial state of the coin. Also, let D be the output of the simulated die ($D \in \{0, \ldots, 6\}$), where $1, \ldots, 6$ is the simulated output of the die, and 0 shows that the coin needs to be flipped again, and the output of the die is not determined yet. The coin needs to be flipped at least three times to simulate observing a number on the die. We define $\Sigma_{die} : C \times D$ as the observation space, that denotes the output of the coin and the die in the process.

Suppose checking the reachability property *eventually the outcome of the die is either "1" or "6"*, at run-time, which translates to the LTL property $\varphi : \Diamond D = 6 \, or \, D = 1$, based on the symbols in the defined observation space. Any (infinite) execution paths with the prefix $u : (ii, 0)(tt, 0)(hh, 0)(hh, 6)$ satisfies φ. However, the result on the prefix $u' : (ii, 0)(tt, 0)(hh, 0)(tt, 0)$ is *unknown* [6], as it can be extended to an infinite path that satisfies φ (e.g., $(ii, 0)(tt, 0)(hh, 0)\ (tt, 0)(hh, 0)(hh, 6)^{\omega}$), or an infinite path that violates φ (e.g., $(ii, 0)(tt, 0)(hh, 0)(tt, 0)(tt, 0)(tt, 5)^{\omega}$).

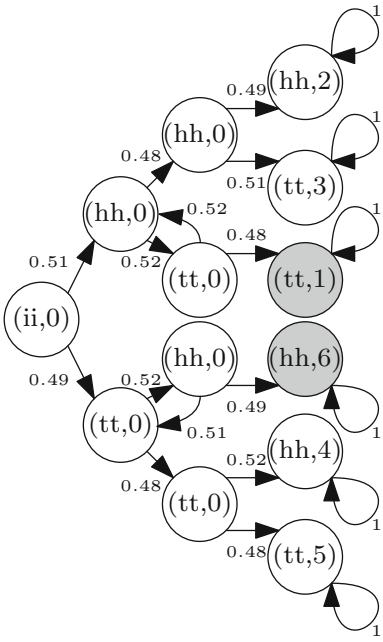


Fig. 2. The learned model of the die example from 1000 samples used as the prediction model.

To deal with the inconclusive results due to unknown extensions [5], we provide the monitor with a *prediction model* to extend the prefix and generate the results based on the probability of the extensions that satisfy the given property. In the die example, the model in Fig. 2 is created from 1000 *iid* samples, with which the monitor is able to compute the probability of all the extensions that satisfy φ (the shaded states). We limit the length of the extensions to some boundary, which we call the *prediction horizon*.

3 Preliminaries

In this section, we briefly introduce definitions and notations. A probability distribution over a finite set S is a function $P : S \to [0, 1]$ such that $\sum_{s \in S} P(s) = 1$. We use $X_{1:n}$ to denote a sequence $x_1, x_2, \ldots, x_n$ of values of a random variable X, and use u and w to, respectively, denote a finite and an infinite path.

Discrete-Time Markov Chains (DTMC).

Definition 1 (DTMC). *A Discrete-Time Markov Chain (DTMC) is a tuple $\mathcal{M} : (S, \Sigma, \pi, \mathbf{P}, L)$, where S is a non-empty finite set of states, Σ is a non-empty finite alphabet, $\pi : S \to [0, 1]$ is the initial probability distribution over S, $\mathbf{P} : S \times S \to [0, 1]$ is the transition probability, such that for any $s \in S$, $P(s, \cdot)$ is a probability distribution, and $L : S \to \Sigma$ is the labeling function.*

Let $\mathcal{M}$ be a DTMC. The sequence $\sigma_0\sigma_1 \ldots$ is an execution path on $\mathcal{M}$ iff $\mathbf{P}(s_i, s_{i+1}) > 0$, where $L(s_i) = \sigma_i, i \geq 0$. An execution path can be finite or infinite. The probability measure on the execution paths is defined using cylinder sets [24]. We deal with finite paths in the remaining of the paper. We denote by $Path_k(\sigma)$ the set of all finite execution paths of length k that start with σ. In the following, we define the finite extension of a path.

Definition 2 (Finite Extensions of a Path). *Let $\sigma_0\sigma_1 \cdots \sigma_n$ be a finite execution path on a DTMC. The set of finite extensions of u is denoted by Ext_u and defined as follows:*

$$Ext_u = \{u \in Path_k(\sigma_n) | k \geq 0\}$$

We denote by $Ext_u^{\leq h}$, if $k \leq h$ in Definition 2.

Definition 3 (Deterministic State). *State $s \in S$ of a DTMC is called deterministic iff for all $\sigma \in \Sigma$, there exists at most one $s' \in S$ such that $\mathbf{P}(s, s') > 0$ and $L(s') = \sigma$.*

Definition 4 (Deterministic DTMC [30]). *A DTMC is deterministic iff:*

- *There exists $s_{init} \in S$ such that $\pi(s_{init}) = 1$.*
- *For all $s \in S$, s is deterministic.*

Hypothesis Testing. Let X be some random variable with *unknown* mean μ and variance σ^2. Let $x_1, \ldots, x_n$ be *iid* random samples of X, with mean $\bar{X}$ and variance v^2. A two-sided null hypothesis test enables us to check $H_0 : |\mu - \bar{X}| = 0$. According to the central theorem [15], as the number of samples grows $|\mu - \bar{X}|$ follows a student's t-distribution [15]. Therefore using confidence α and the t-distribution we are able to accept H_0, if $\frac{|\bar{X} - \mu|}{\frac{\sqrt{v}}{n}} \leq t_{n-1,\alpha}$.

4 Prediction as Quantitative Bounded Reachability Analysis

Let $\mathcal{M}$ be a DTMC, $\mathcal{G} \subseteq \Sigma$ be the set of *target* labels of some goal states in $\mathcal{M}$, and $\mathcal{L}(\mathcal{G})$ be the set of finite execution paths on $\mathcal{M}$ that end with $\sigma \in \mathcal{G}$. The monitor's purpose is to estimate the probability of all the extensions of length at most h that satisfy $\Diamond\mathcal{G}$. In a discrete-time setting, the variable h is a positive integer, which we call the *prediction horizon*. Computing the probability of the satisfying extensions is achieved by performing a bounded reachability analysis on the prediction model.

Let $u \notin \mathcal{L}(\mathcal{G})$ be the execution on $\mathcal{M}$ observed so far, and Ext_u be the set of finite extensions of u. The output of the monitor is the probability measure of the set of the paths of length at most h that satisfy $\Diamond\mathcal{G}$, i.e.,

$$Pr(\mathcal{C}), \text{ where } \mathcal{C} = \{v \in Ext_u^{\leq h} | uv \in \mathcal{L}(\mathcal{G})\}) \tag{1}$$

Suppose also that s is the state corresponding to the last label in u, which is obtained by some state estimation technique [3,21,40]. As u expands, the new state is consequently estimated at run-time.

The probability (1) can be obtained by recursively computing the *transient probability* in $\mathcal{M}$ [24]: starting from s the probability of being at state s' after h steps, such that $L(s') \in \mathcal{G}$. We can effectively turn computing the transient probability into checking the following quantitative PCTL property [17,24]:

$$\mathtt{P}_{=?}[s \models \Diamond^{\leq h}\, \mathcal{G}] \tag{2}$$

Due to multiplications of large and typically sparse matrices, the calculation of (2) is not practical during run-time [24]. As a result, we use PRISM [34] to execute the quantitative reachability analysis [4] on all the states of $\mathcal{M}$ off-line, and store the values in a look-up table. The size of the table is $O(|\mathcal{M}|)$ [3], where $|\mathcal{M}|$ is the size of the prediction model. Subsequently, reducing the size of the model results in a smaller look-up table, in addition to the performance improvement due to a faster state estimation.

In our example, $\mathcal{G}_\varphi = \{(tt,1),(hh,1),(hh,6),(tt,6)\}$. Using the model in Fig. 2, the monitor is able to retrieve the value of $\mathtt{P}_{=?}(s_{u'} \models \Diamond^{\leq h}\mathcal{G}_\varphi)$, where $s_{u'}$ is the state of the model after generating u'. For instance, $\mathtt{P}_{=?}(s_{u'} \models \Diamond^{\leq 20}\mathcal{G}_\varphi) = 0.34$, which translates to the probability of all the extensions of length at most 20, that terminate with the labels where the outcome of the die is 6 or 1.

5 Trace-Level Abstraction by Inferring Symmetry in the Observation Space

We use a finite partitioning of the observation space, Σ, to achieve the trace-level abstraction in the form of a *projection*. The abstract traces are then used to train the prediction models, which ideally have smaller sizes than the model trained from the concrete traces, but are good enough approximates.

The simplest abstraction is to divide the observation space into two partitions, the atomic propositions that appear in the target labels and the non-target labels [33], i.e., $\mathcal{G}$, and $\Sigma - \mathcal{G}$, which we denote by $\Sigma_{\bar{\mathcal{G}}}$. The projection $R_{\mathcal{G}} : \Sigma \rightarrow \{gg, nn\}$ simply maps a symbol of a path to the symbol gg, if it is a target label, i.e., $R(\sigma) = gg$ iff $\sigma \in \mathcal{G}$, and to the symbol nn otherwise. Consider $\mathcal{G}_\varphi$ and the traces u and u'. The projection of u and u', using the projection relation $R_{\mathcal{G}_\varphi}$, are respectively, $\tilde{u} : (nn)(nn)(nn)(gg)$ and $\tilde{u'} : (nn)(nn)(nn)(nn)$.

The projection $R_{\mathcal{G}}$ may collapse the non-target symbols, that have a non-zero probability to reach a target label within some bounded steps, with symbols that never reach the target labels, i.e., have probability zero. For instance, compare the symbols $(hh, 0)$ and $(hh, 4)$. The former appears immediately before the target label $(hh, 6)$ often in a simulated path; whereas the latter has no appearance before any of the target labels (see Fig. 2). Both are replaced with nn in $R_{\mathcal{G}}$, thus the predictive information from $(hh, 0)$ and $(hh, 4)$ are combined.

The key insight in our abstraction method is to not only detect the symbols with no significant correlation with the target labels, but also recognize the ones that have similar empirical probability to reach the target labels within a fixed number of steps, and collapse them together. As a result, we leverage the notion of *symmetry* [23] on the observation space to recognize the symbols with similar prediction power and lump them into the same partition. We define the symmetry relation with respect to reaching the target labels. More specifically, we say two symbols are symmetrical *iff* the probability measure of a fixed length path, that starts from either symbols and ends with some target labels, is equal.

Let $\mathcal{M}$ be a deterministic DTMC. Let $P_k : \Sigma_{\bar{\mathcal{G}}} \rightarrow \Sigma_{\bar{\mathcal{G}}}$ be the permutations on the non-target labels, such that $Pr(Path_k^{\mathcal{G}}(P_k(\sigma))) = Pr(Path_k^{\mathcal{G}}(\sigma))$ for all $\sigma \in \Sigma_{\bar{\mathcal{G}}}$, and some fixed integer $k > 0$, where $Path_k^{\mathcal{G}}(.) = Path_k(.) \cap \mathcal{L}(\mathcal{G})$. A group of permutations defined by P_k on $\Sigma_{\bar{\mathcal{G}}}$ provides an equivalence relation (so-called the *orbits*) on $\Sigma_{\bar{\mathcal{G}}}$ that with $\mathcal{G}$ defines the equivalence classes over the observation space. We denote by Σ_k the abstract alphabet set that contains a unique representative symbol for each partition, and by $R_k : \Sigma \rightarrow \Sigma_k$ the corresponding projection that maps each symbol to its rep. in the abstract alphabet.

Let $\mathcal{M}_k$ be the quotient of $\mathcal{M}$, where Σ is replaced with Σ_k, and $L_k : S \rightarrow \Sigma_k$ such that $L_k(s) = R_k(L(s))$. Given that $\mathcal{M}_k$ is deterministic, we can show that $\mathcal{M}_k$ is bisimilar to $\mathcal{M}$. Therefore bounded predictions from any state in both models are equal (see Theorem 1). In the case that $\mathcal{M}_k$ is non-deterministic, we employ a hidden state variable to infer the non-determinism imposed by abstraction (see Sect. 6.2).

In the remainder, we use k-gapped pair model (Sect. 5.1) combined with hypothesis testing (Sect. 5.2) to infer R_k, and consequently, Σ_k.

5.1 *k*-gapped Pair Model

The k-gapped pair model [13] has been successfully applied in mining biological sequences [20], and in context-dependent text prediction [11]. We use the k-gapped pair model to extract the predictive symbols with respect to the target labels in the training set.

1 ComputePredictionSupport$(S, \sigma, \mathcal{G}, k)$

inputs : The *iid* sample set $S = [u_1 \dots u_m]$, $\sigma \in \Sigma$, the set of target labels $\mathcal{G}$, and an integer $k \geq 0$

output: $[F_{u_1} \dots F_{u_m}]$

2 **begin**

3 | **foreach** $u_i \in S$ **do**

4 | | $n \leftarrow length(u_i)$

5 | | $F_{u_i} \leftarrow \frac{1}{n-k-1} \sum_{j=1}^{n-k-1} \mathbb{1}_{(\sigma_j = \sigma \text{ and } \sigma_{j+k+1} \in \mathcal{G})}$

6 | **end**

7 **end**

Algorithm 1: Computing the k-prediction support of σ over the sample set.

A k-gapped pair model is a triplet (σ, σ', k), where $\sigma, \sigma' \in \Sigma$, and $k \geq 0$ is an integer that indicates the number of steps (gaps) between σ and σ'. If $k = 0$ the k-gapped pair is equivalent to a *bigram* [31].

The *k-gapped occurrence frequency* of the symbols σ and σ', is the frequency that σ appeared within exactly k steps before σ' over the sample path. Assuming that $\sigma' \in \mathcal{G}$, we use the sum of k-gapped occurrence frequency of a given symbol over all the target labels in the sample set, and define it as the *k-prediction support*. Algorithm 1 shows computing k-prediction support of symbol σ. Symbols σ_j and σ_{j+k+1} are the j^{th} and $(j+k+1)^{th}$ symbols of the sample path u_i in each iteration of the loop in line **3**, and $\mathbb{1}_{(\sigma_i = \sigma \text{ and } \sigma_{i+k+1} = \sigma')}$ in line **5** is the indicator function that returns 1 if $\sigma_i = \sigma$ and $\sigma_{i+k+1} = \sigma'$; and 0 otherwise. The output of Algorithm 1 is the vector $[F_{u_1} \dots F_{u_m}]$, the k-prediction support values of each sample path for symbol σ.

The k-prediction support of σ is essentially the empirical estimation of $Pr(Path_k^{\mathcal{G}}(\sigma))$. Under the assumption that $F_{u_1}, \dots, F_{u_m}$ is *covariance-stationary* [16], i.e., the mean is time-invariant and the autocovarinace function depends only on the distance k, both of which hold if the samples are *iid*, and the underlying generating model is a deterministic DTMC, we are able to use hypothesis testing to extract R_k.

5.2 Using Hypothesis Testing to Extract Symbols with Equivalent Prediction Support

Algorithm 2 demonstrates the procedure of extracting the abstract alphabet set, based on the symmetry between the k-prediction support of the symbols. The algorithm receives the sample set, the alphabet set, the set of target labels, and k, as inputs, and infer R_k, by generating the partitions $V_1, \dots, V_t$.

The algorithm iterates over the symbols not considered in any equivalence classes, that are stored in $\mathcal{R}$ (the loop in lines **5**–**17**). In each iteration, the symbol with the maximum k-prediction support score is found in $\mathcal{R}$ and stored in σ_{max} with its score in F_{max} (line **6**). The score 0 indicates that there is no

```
1  ExtractAbstractAlphabet(Sample set S, Σ, G, k)
   output: Partition [G ∪ V_1 ∪ ··· ∪ V_t ∪ R] over Σ
2  begin
3      t ← 1
4      R ← Σ − G
5      while R ≠ ∅ do
6          [σ_max F_max] ← max_{σ∈R} Σ_{i=1}^{m} F_{u_i}
7          if F_max = 0 then break
8
9          V_t ← {σ_max}
10         for σ ∈ R − {σ_max} do
11             F_σ ← ComputePredictionSupport(S, σ, G, k)
12             if HypothesisTesting(F_max − F_σ) then
13                 V_t ← V_t ∪ {σ'}
14             end
15         end
16         R ← R − V_t
17         t ← t + 1
18     end
19 end
```

Algorithm 2: Extracting the equivalence classes on the alphabet set.

path of length k to any target labels from the symbols in $\mathcal{R}$ and we can end the procedure (line **7**); otherwise, the symbols with statistically similar k-prediction support score to σ_{max} are extracted from $\mathcal{R}$, and inserted in V_t (for loop in **9**–**13**). The statistical testing is conducted via the function HypothesisTesting, which performs a two-sided hypothesis t-test to check $H_0 : F_{max} - F_\sigma = 0$. Depending on the number of samples, a proper confidence is chosen to test H_0.

Algorithm 2 terminates, if there is no more symbol to classify, i.e., $\mathcal{R} = \emptyset$, or if all the remaining symbols in $\mathcal{R}$ have no k-prediction support. We dedicate a representative symbol for each extracted partition, including $\mathcal{G}$ and $\mathcal{R}$ if it is not empty, and define R_k accordingly.

At worst, a total number of $O(|\Sigma|^2)$ comparisons is required to extract the abstract alphabet set. Given that the size of the actual model is at least as large as $|\Sigma|$, storing the entire vector of k-prediction support score for all symbols is impractical for large models. In fact, to make the usage of memory independent of the size of the alphabet, the computation of F in Algorithm 1 is performed on-the-fly, which only depends on the size of sample data. Compared to inferring the abstract alphabet from a model that is trained from the concrete traces, our approach is more memory-efficient.

Table 1 demonstrates the k-prediction support of the symbols $(tt, 0)$ and $(hh, 0)$, for $k = 0, 1, 2$, over 1000 samples. The equivalence classes obtained by Algorithm 2 for $k = 2$, where $R_2(\sigma_g) = gg, \sigma_g \in \mathcal{G}_\varphi, R_2(\sigma_v) = v1, \sigma_v \in V_1 = \{(ii, 0), (hh, 0), (tt, 0)\}, R_2(\sigma_n) = nn, \sigma_n \in \mathcal{R} = \Sigma_{die} - \mathcal{G}_\varphi - V_1$. Notice that

according to the original model in Fig. 2 the probability of reaching any of the labels in $\mathcal{G}_\varphi$ from $(ii, 0), (hh, 0), (tt, 0)$ in 3 steps (within 2 gaps) is equal.

6 Learning

6.1 Learning Deterministic DTMC

Amongst *probably almost correct* (PAC) techniques to train deterministic DTMCs [42], state-merging algorithms [9], are known to be effective. We use ALERGIA [1,30], that generates a *frequency prefix tree acceptor* (FPTA) from the training data, and then applies data-dependent compatibility criterion, parameterized by α, to merge the states of the FPTA, and finally transforms it into a DTMC by normalizing the frequencies. The learned model converges to the generating deterministic DTMC, $\mathcal{M}$, in the limit for any $\alpha > 1$ [33]. Figure 3 depicts the learned model from the abstract traces, based on the extracted symbols in Table 1.

In the following, we demonstrate the correctness of the predictions made by the deterministic DTMC trained from the abstract traces.

Theorem 1. *Let $\mathcal{M}_k : (S, \Sigma_k, \pi, \mathbf{P}, L_k)$ be the representation of $\mathcal{M}$, where the states are relabelled based on the symbols in Σ_k. Suppose $\mathcal{G} \subseteq \Sigma$ is the set of target symbols, and $'gg'$ is their representative symbol in Σ_k. Also let $\mathcal{M}^{\#} : (\tilde{S}, \Sigma_k, \tilde{\pi}, \tilde{\mathbf{P}}, \tilde{L})$ be the learned model from the samples of $\mathcal{M}_k$, using any PAC learning algorithm. Then, under the assumptions of the convergence of the learning algorithm,*

$$\Pr(\tilde{s} \models \Diamond^{\leq h}(gg)) = Pr(s \models \Diamond^{\leq h}\mathcal{G}), \forall \tilde{s} \in \tilde{S}, s \in S. \tag{3}$$

Table 1. The k-prediction support of all the symbols except the target labels, $(hh, 6)$ and $(tt, 1)$, for $k = 0, 1, 2$, obtained from 1000 sample paths of the die example (scale $\times 10^{-3}$).

	$k=0$	$k=1$	$k=2$
$(ii, 0)$	0	0	22.01
$(hh, 0)$	9.26	20.05	23.20
$(tt, 0)$	10.21	21.05	23.22
$(hh, 2)$	0	0	0
$(tt, 3)$	0	0	0
$(hh, 4)$	0	0	0
$(tt, 5)$	0	0	0

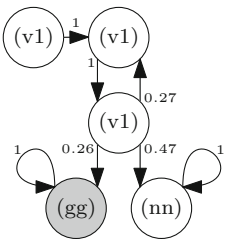


Fig. 3. The learned deterministic DTMC from the abstract traces using the alphabet $\{gg, v1, nn\}$ using 2-prediction support.

Proof. Notice that since prediction in our setting is a *bounded* LTL property, (3) is valid for the initial state, regardless of whether $\mathcal{M}_k$ is deterministic or not (see Theorem 1 in [33]). If $\mathcal{M}_k$ is a deterministic DTMC, the states of the trained model $\mathcal{M}^{\#}$, almost surely bisimulates the states of $\mathcal{M}_k$ in the limit (see Theorem 1 in [30]). As a result, under the assumptions of the convergence of the learning algorithm, (3) is followed. □

6.2 Learning Non-deterministic DTMC Using HMM

A Hidden Markov Model (HMM) specifies the distribution of a sequence as the joint distribution of two random variables: the hidden state variable, and the observation variable. More particularly, an HMM is the joint distribution over $X_{1:n}$, the sequence of the state variable, and $Y_{1:n}$, the sequence of observations (both with identical lengths). The joint distribution is such that $Pr(y_i \mid X_{1:i}, Y_{1:i}) = Pr(y_i \mid x_i)$ for $i \in [1, n]$, i.e., the current observation is conditioned only on the current state, and $Pr(x_i \mid X_{1:i-1}, Y_{1:i-1}) = Pr(x_i \mid x_{i-1})$ for $i \in [1, n]$ i.e., the current state is only conditioned on the previous hidden state. We use π to denote the initial probability distribution over the state space, i.e., $Pr(x_1) = \pi(x_1)$. As a result, an HMM can be defined with three distributions:

Definition 5 (HMM). *A finite discrete Hidden Markov Model (HMM) is a tuple $\mathcal{H} : (S, \Sigma, \pi, T, O)$, where S is the non-empty finite set of states, Σ is the non-empty finite set of observations, $\pi : S \to [0, 1]$ is the initial probability distribution over S, $T : S \times S \to [0, 1]$ is the transition probability, and $O : S \times \Sigma \to [0, 1]$ is the observation probability. We use $\Theta_{\mathcal{H}}$ to denote π, T, and O.*

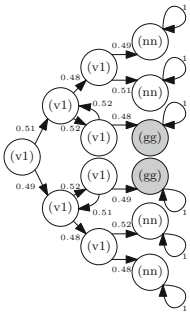
In our setting, observations are the symbols of the abstract alphabet, Σ_k, and hidden states are the states of the generating model, i.e., the deterministic DTMC. The random hidden state variable creates an extra degree of freedom which allows to distinguish states that emit the same symbol but have different joint probability distributions.

Notice that the PAC learning of an HMM is a hard problem under cryptographic assumptions [10,41] because the probability distribution over the state sequence X is unknown, therefore, the likelihood function does not have a closed form [41]. Subsequently, approximate algorithms, such as Expectation-Maximization (EM) [8], are employed. Training an HMM using EM is known as the Baum-Welch algorithm [37] (BWA), which calculates the parameters of the HMM by finding the maximum likelihood of the sample data. BWA requires the number of hidden states in a finite state HMM as an input, or *hyper-parameter*. The hyper-parameters are typically chosen based on some criterion that prevents overfitting (e.g., Bayesian Information Criterion (BIC) [12]).

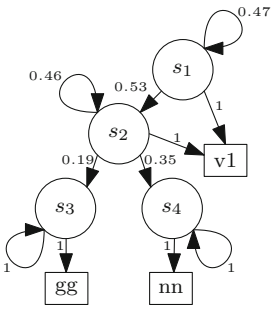
To run the reachability analysis on the HMM, $\mathcal{H} : (S, \Sigma, \pi, T, O)$, we adopt the direct method in [45] to create an equivalent DTMC, $\mathcal{M}_H : (S \times \Sigma, \Sigma, \pi(s, .), \mathbf{P}, L(., \sigma))$, where $\mathbf{P}((s, \sigma), (s', \sigma')) = T(s, s') \times O(s', \sigma')$.

Figure 4 displays the non-deterministic DTMC and the trained HMM over the abstract traces obtained by R_2 from Table 1. The DTMC in Fig. 4a is similar to

the DTMC in Fig. 2, except that the states are relabeled using R_2 (see Sect. 5.2). Relabeling the model creates non-determinism as the sequence $(v1, v1, v1)$ corresponds to several state sequences in the model. The HMM in Fig. 4b is obtained by training HMM that has 4 hidden states. Each hidden state corresponds to the set of states in the DTMC in Fig. 4a with the same labels. There are two hidden states associated with $(v1)$ to distinguish between the states of the model in Fig. 4a that reach the target states (labelled (gg)) with different joint probabilities.



(a) The abstract DTMC obtained by relabeling the states of the DTMC in Fig. 2 using R_2.



(b) The learned HMM from the abstract traces created by R_2 with 4 hidden states.

Fig. 4. Training non-deterministic DTMC using HMM.

7 Case Study: Randomised Self-Stabilising Algorithm

We use Herman's self-stabilising algorithm [18] to experiment our approach in *Prevent*.[1] The algorithm provides a protocol for N identical processes (N is odd) in a token ring network, with unidirectional synchronous communication. Starting from an arbitrary configuration, the network will eventually converge to a defined *stable* state within a finite number of steps. The token is infinitely circulated in the ring amongst the processes in a fair manner. The stable state is defined as such that there is exactly one process that has the token. The process i has a local Boolean variable x_i. If $x_i = x_{i-1}$ there is a token with process i, in which case process i randomly chooses to set x_i to the next value or leave it unchanged (i.e., equal to x_{i-1}).

The observation space for a ring with N processes has 2^N symbols, each representing the values of the local variable in each process. The observation space maps one-to-one to the state space of the corresponding DTMC. We are interested in monitoring the property φ_{stable} : $\Diamond^{\leq h}$ "*stable*" which translates into the target symbols in which only one process has the same label as its

[1] Available at https://bitbucket.org/rbabaeecar/prevent/.

left neighbour, i.e., there exists only one i such that $x_i = x_{i-1}$. The monitoring procedure throughout our experiments is performed off-line; however, in principle the on-line monitoring procedure would be the same, except that the execution path keeps expanding as the system continues running.

We collected 1000 *iid* samples from the DTMC using PRISM simulation tool [34]. The length of the samples is constrained by a maximum, and is distributed uniformly. We first run Algorithm 2 to extract the predictive symbols for the target symbols specified by φ_{stable}. We replaced the symbols of the sampled traces based on the found partitioning, and performed the training algorithms to learn a deterministic DTMC as well as an HMM. The number of states in the HMM is chosen so that it is comparable to the size of the learned DTMC.

Prediction Evaluation: Using the values obtained from the actual models, we use Mean-Square Prediction Error (MSPE) [15] to measure the performance of the predictions by each model. The evaluation is conducted on a separate *iid* sample from the training samples, where the following is computed for each instance i that the prediction is made (i.e., a target label is not still observed):

$$\varepsilon_i^2 = (Pr(s \models \Diamond^{\leq h}\mathcal{G}) - Pr(\hat{s} \models \Diamond^{\leq h}\mathcal{G}))^2 \tag{4}$$

where s is the state in the actual DTMC, and $\hat{s}$ is the state in the prediction model, estimated using FORWARD-BACKWARD algorithm [37]. The FORWARD-BACKWARD algorithm computes the posterior probability of the state space given the observation u, i.e., $Pr(X_n = s \mid u, \Theta)$, where n is the length of u. We define MSPE as the average of (4), i.e., $\frac{1}{t}\sum_{i=1}^{t}\varepsilon_i^2$ where t indicates the number of points on the sample where a prediction is made.

Table 2. The prediction results of different models on 100 random samples.

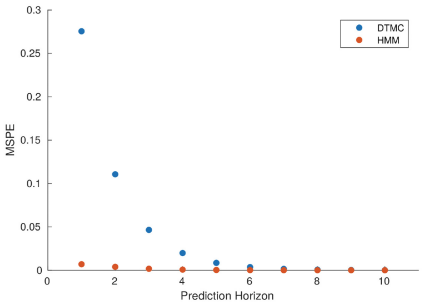
N	Orig. Alph.	Learned DTMC *conc.*		Abst. Alph.	Learned DTMC *abst.*			Learned HMM		
		Size	Training time (s)		Size	Training time (s)	MSPE $e{-}02$	Size	Training time (s)	MSPE $e{-}02$
5	32	18	1047.12	5	5	16.12	27.57	4(7)	9.95	0.70
7	128	1319	19605.47	3	3	866.27	32.28	3(5)	61.13	1.39
9	512	7914	135004.38	2	2	275.54	79.36	3(4)	47.16	1.79
11	2048	O/M	–	2	2	2696.73	87.52	2(3)	2496.20	1.35

Table 2 summarizes our results of three different prediction models compared to the prediction results obtained from the original model for $N \in \{5, 7, 9, 11\}$, $k = 0$ (a bigram model), and prediction horizon equal to one step. The size of the original model is identical to the size of the alphabet, as there is exactly one state corresponding to the valuation of the local variables in each process. We used AALERGIA [1] to train DTMC from both the concrete and abstract traces, and Matlab HMM toolbox to train the HMMs. The training was performed on an Ubuntu 17.10 machine with 24 GB RAM. Training a DTMC from concrete

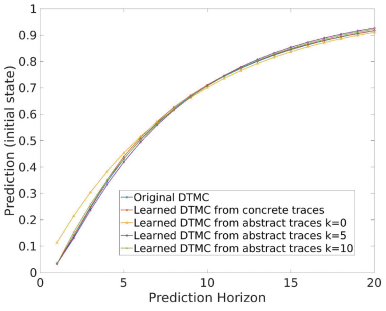
traces was aborted for $N = 11$ due to lack of memory, as the length of the FPTAs grows exponentially with the size of the alphabet. The trained DTMCs from abstract traces have significantly smaller size in direct relation to the size of the inferred abstract alphabet, and consequently a shorter training time. This result is consistent with the fact that the actual model is highly symmetrical with respect to the stable states, i.e., the probability of reaching the stable states from the states within an equivalence class in one step is equal.

The sizes of the trained HMMs are determined by the BIC score, and the size of their equivalent DTMCs are shown in parentheses. As we can see an HMM with comparable size is substantially more accurate in making predictions than a DTMC. The state estimation also benefits from the small size of the HMM with virtually no computational overhead. Since the prediction horizon is formulated as an upper-bound, the probability of an accepting extension increases as the prediction horizon increases, which in turn results in a lower MSPE. However, as depicted in Fig. 5a trained HMM has lower error-rate even for shorter range of predictions, e.g., for $h = 5$, the MSPE of the HMM is $0.03e{-}02$ as opposed to $0.85e{-}2$ for the DTMC.

Figure 5b demonstrates the prediction results of the DTMC trained from concrete traces, and the traces abstracted by k-prediction support, using $k = 0, 5, 10$. The prediction results are from the initial state, and as we can see, the models learned from abstract traces almost follow the values of the actual model. The best result belongs to $k = 10$, which echos the maximum expected time to reach a stable state, i.e., that the path to the target symbols from the initial state is of length at most about 9.



(a) MSPE of the trained DTMC and HMM on abstract traces with $k = 0$, $N = 5$ (prediction horizon [1,10]).



(b) The prediction results from the initial state of different prediction models compared to the original model for $N = 9$.

Fig. 5. The experiment results for abstract models in the Herman algorithm.

8 Related Work

To the best of our knowledge our approach is novel in terms of applying learning and abstraction to predictive RV, and using HMM to handle non-determinism at the trace level. Xian Zhang *et al.* [46] introduce a predictive LTL semantics definition, that is applied on white-box systems (i.e., using the control flow graphs), to find the extensions that evaluate an LTL formula to *true* or *false*. Martin Leucker [27] follows the same idea and extends the semantics and the monitor construction for when an over-approximation of the program is given. Our approach works on black-box systems and finds the probability of the extensions that satisfy a reachability property based on the sample executions of the system that form a probabilistic model. Furthermore, we define the abstraction by inferring a projection function over the observation space using the sample executions.

Sistla *et al.* [39] propose an *internal* monitoring approach (i.e., the property is specified over the hidden states) using specification automata and HMMs with infinite states. Learning an infinite-state HMM is a harder problem than the finite HMMs, but does not require inferring the size of the model [7].

Nouri *et al.* [33] use abstraction and learning to expedite statistical model checking [25]. Their approach is the probabilistic variant of *black box checking* [35] in which the inferred model, in the form of definite finite automata (DFA), is checked against some properties. In our case, we use abstraction to obtain a smaller prediction model for predictive RV. In [33] the atomic propositions in the property are used for abstracting the traces. We perform a statistical analysis on the traces to obtain partitions that leave the prediction probability intact. We also use HMM to handle the potential non-determinism introduced by the abstraction.

Aichernig and Tappler [2] employ black box checking in the context of reachability analysis of stochastic systems with controllable inputs. They use Markov Decision Process (MDP), an extension of a Markov chain with non-deterministic choices, as a model that is trained from random samples. They use the inferred MDP to obtain an *adversary* with which they collect new samples and incrementally train new MDPs. LAR [44] is a combination of probabilistic model learning and counterexample guided abstraction refinement (CEGAR) [19]. These approaches are orthogonal to our technique and it is straightforward to extend the training to other models such as MDP, and apply probabilistic CEGAR to obtain a model that guarantees checking affirmative properties.

9 Conclusion

We propose inferring a projection relation from a random set of samples to abstract traces that are used in building prediction models to monitor a discrete-time reachability property. Our inference technique is based on finding a symmetrical relation between the symbols of the alphabet, using k-gapped pair model, and lumping them into same equivalence classes. We use the abstract

traces to train deterministic DTMC as well as HMM to handle the possible non-determinism induced by abstraction. We show that the prediction results remain intact with the model trained from abstract traces, under the condition that the generating DTMC of the abstract traces is deterministic and under the constraints of the learning algorithm. We evaluated our approach on a distributed randomized algorithm, and demonstrated that in general the trained HMM from the abstract traces is more accurate than the trained DTMCs.

Our approach is most effective on the systems with large observation space, and where the model benefits from symmetry in the probability of reaching some states. The statistical analysis of the traces proposed in this paper to reduce the observation space requires enough number of executions with the target labels to reliably detect similar symbols. If the target labels indicate some rare events (e.g., *error* with very low probability), techniques such as [26] are required to simulate enough traces with target labels. The trained model from such simulated sample traces need to be adjusted.

Lastly, an implementation of $\mathcal{P}revent$ with the application of on-line learning methods (such as state merging or splitting techniques [32]) is necessary to apply the framework to the real-world scenarios.

References

1. Aalergia: http://mi.cs.aau.dk/code/aalergia/. Accessed 15 Mar 2018
2. Aichernig, B.K., Tappler, M.: Probabilistic black-box reachability checking. In: Lahiri, S., Reger, G. (eds.) RV 2017. LNCS, vol. 10548, pp. 50–67. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67531-2_4
3. Babaee, R., Gurfinkel, A., Fischmeister, S.: $\mathcal{P}revent$: A Predictive Run-Time Verification Framework Using Statistical Learning. In: Johnsen, E.B., Schaefer, I. (eds.) SEFM 2018. LNCS, vol. 10886, pp. 205–220. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92970-5_13
4. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
5. Bauer, A., Leucker, M., Schallhart, C.: The good, the bad, and the ugly, but how ugly is ugly? In: 7th International Workshop RV, pp. 126–138 (2007)
6. Bauer, A., Leucker, M., Schallhart, C.: Comparing LTL semantics for runtime verification. J. Log. Comput. **20**(3), 651–674 (2010)
7. Beal, M.J., Ghahramani, Z., Rasmussen, C.E.: The infinite hidden Markov model. In: Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, pp. 577–584. MIT Press, Cambridge (2001)
8. Bilmes, J.A.: A gentle tutorial of the EM algorithm and its applications to parameter estimation for Gaussian mixture and hidden Markov models. Technical report TR-97-021, International Computer Science Institute, Berkeley, CA (1997)
9. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: Carrasco, R.C., Oncina, J. (eds.) ICGI 1994. LNCS, vol. 862, pp. 139–152. Springer, Heidelberg (1994). https://doi.org/10.1007/3-540-58473-0_144
10. Castro, J., Gavaldà, R.: Learning probability distributions generated by finite-state machines. In: Heinz, J., Sempere, J.M. (eds.) Topics in Grammatical Inference, pp. 113–142. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-48395-4_5

11. Chan, S.W.K., Franklin, J.: A text-based decision support system for financial sequence prediction. Decis. Support Syst. **52**(1), 189–198 (2011)
12. Claeskens, G., Hjort, N.L.: Model Selection and Model Averaging. Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge (2008)
13. Dong, G., Pei, J.: Sequence data mining. In: Advances in Database Systems, vol. 33, Kluwer (2007)
14. Falcone, Y., Fernandez, J.-C., Mounier, L.: Runtime verification of safety-progress properties. In: Bensalem, S., Peled, D.A. (eds.) RV 2009. LNCS, vol. 5779, pp. 40–59. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04694-0_4
15. Geisser, S.: Predictive Inference, Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, UK (1993)
16. Hamilton, J.D.: Time Series Analysis. Princeton University Press, Princeton (1994)
17. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Asp. Comput. **6**(5), 512–535 (1994)
18. Herman, T.: Probabilistic self-stabilization. Inf. Process. Lett. **35**(2), 63–67 (1990)
19. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 162–175. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70545-1_16
20. Huang, S., Liu, R., Chen, C., Chao, Y., Chen, S.: Prediction of outer membrane proteins by support vector machines using combinations of gapped amino acid pair compositions. In: Fifth IEEE International Symposium on Bioinformatic and Bioengineering (BIBE 2005), 19–21 October 2005, Minneapolis, MN, USA, pp. 113–120. IEEE Computer Society (2005)
21. Kalajdzic, K., Bartocci, E., Smolka, S.A., Stoller, S.D., Grosu, R.: Runtime verification with particle filtering. In: Legay, A., Bensalem, S. (eds.) RV 2013. LNCS, vol. 8174, pp. 149–166. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40787-1_9
22. Knuth, D.: The complexity of nonuniform random number generation. In: Algorithm and Complexity, New Directions and Results, pp. 357–428 (1976)
23. Kwiatkowska, M., Norman, G., Parker, D.: Symmetry reduction for probabilistic model checking. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 234–248. Springer, Heidelberg (2006). https://doi.org/10.1007/11817963_23
24. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-72522-0_6
25. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: an overview. In: Barringer, H. (ed.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16612-9_11
26. Legay, A., Sedwards, S., Traonouez, L.-M.: Rare events for statistical model checking an overview. In: Larsen, K.G., Potapov, I., Srba, J. (eds.) RP 2016. LNCS, vol. 9899, pp. 23–35. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45994-3_2
27. Leucker, M.: Sliding between model checking and runtime verification. In: Qadeer, S., Tasiran, S. (eds.) RV 2012. LNCS, vol. 7687, pp. 82–87. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35632-2_10
28. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebr. Program. **78**(5), 293–303 (2009)
29. Maler, O.: Some thoughts on runtime verification. In: Falcone, Y., Sánchez, C. (eds.) RV 2016. LNCS, vol. 10012, pp. 3–14. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46982-9_1

30. Mao, H., Chen, Y., Jaeger, M., Nielsen, T.D., Larsen, K.G., Nielsen, B.: Learning probabilistic automata for model checking. In: Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011, Aachen, Germany, 5–8 September, pp. 111–120 (2011)
31. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. CoRR, abs/1301.3781 (2013)
32. Mukherjee, K., Ray, A.: State splitting and merging in probabilistic finite state automata for signal representation and analysis. Sign. Process. **104**, 105–119 (2014)
33. Nouri, A., Raman, B., Bozga, M., Legay, A., Bensalem, S.: Faster statistical model checking by means of abstraction and learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 340–355. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11164-3_28
34. Parker, D., Norman, G., Kwiatkowska, M.: Prism model checker. http://www.prismmodelchecker.org/. Accessed 14 Aug 2017
35. Peled, D.A., Vardi, M.Y., Yannakakis, M.: Black box checking. J. Automata, Lang. Comb. **7**(2), 225–246 (2002)
36. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, pp. 46–57 (1977)
37. Rabiner, L.R.: A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE **77**(2), 257–286 (1989)
38. Roweis, S.T., Ghahramani, Z.: A unifying review of linear Gaussian models. Neural Comput. **11**(2), 305–345 (1999)
39. Sistla, A.P., Žefran, M., Feng, Y.: Monitorability of stochastic dynamical systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 720–736. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_58
40. Stoller, S.D., Bartocci, E., Seyster, J., Grosu, R., Havelund, K., Smolka, S.A., Zadok, E.: Runtime verification with state estimation. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 193–207. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29860-8_15
41. Terwijn, S.A.: On the learnability of hidden Markov models. In: Adriaans, P., Fernau, H., van Zaanen, M. (eds.) ICGI 2002. LNCS (LNAI), vol. 2484, pp. 261–268. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45790-9_21
42. Verwer, S., Eyraud, R., de la Higuera, C.: PAUTOMAC: a probabilistic automata and hidden Markov models learning competition. Mach. Learn. **96**(1–2), 129–154 (2014)
43. Viterbi, A.J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Trans. Inform. Theor. **13**(2), 260–269 (1967)
44. Wang, J., Sun, J., Qin, S.: Verifying complex systems probabilistically through learning, abstraction and refinement. CoRR, abs/1610.06371 (2016)
45. Zhang, L., Hermanns, H., Jansen, D.N.: Logic and model checking for hidden Markov models. In: Wang, F. (ed.) FORTE 2005. LNCS, vol. 3731, pp. 98–112. Springer, Heidelberg (2005). https://doi.org/10.1007/11562436_9
46. Zhang, X., Leucker, M., Dong, W.: Runtime verification with predictive semantics. In: Goodloe, A.E., Person, S. (eds.) NFM 2012. LNCS, vol. 7226, pp. 418–432. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28891-3_37