Faster Statistical Model Checking by Means of Abstraction and Learning

Ayoub Nouri, Balaji Raman, Marius Bozga, Axel Legay, Saddek Bensalem

Faster Statistical Model Checking by Means of Abstraction and Learning *

Ayoub Nouri[1], Balaji Raman[1], Marius Bozga[1]
Axel Legay[2], and Saddek Bensalem[1]

[1] Univ. Grenoble Alpes, VERIMAG, F-38000 Grenoble, France
CNRS, VERIMAG, F-38000 Grenoble, France
[2] INRIA/IRISA, Rennes, France

Abstract. This paper investigates the combined use of abstraction and probabilistic learning as a means to enhance statistical model checking performance. We are given a property (or a list of properties) for verification on a (large) stochastic system. We project on a set of traces generated from the original system, and learn a (small) abstract model from the projected traces, which contain only those labels that are relevant to the property to be verified. Then, we model-check the property on the reduced, abstract model instead of the large, original system. In this paper, we propose a formal definition of the projection on traces given a property to verify. We also provide conditions ensuring the correct preservation of the property on the abstract model. We validate our approach on the Herman's Self Stabilizing protocol. Our experimental results show that (a) the size of the abstract model and the verification time are drastically reduced, and that (b) the probability of satisfaction of the property being verified is correctly estimated by statistical model checking on the abstract model with respect to the concrete system.

1 Introduction

Statistical Model-Checking (SMC) [12, 17, 24, 27] has recently emerged as an alternative to standard model-checking to avoid exhaustive exploration of the state-space and its associated explosion problem. SMC combines Monte-Carlo simulation [11] on model traces with statistical techniques in order to decide whether some stochastic model satisfies a given property or to compute its satisfaction probability. Nowadays, SMC is getting increased industrial attention [4] and several modeling and/or analysis frameworks include it amongst their (usually, most successful) analysis techniques [5, 16, 15, 3].

SMC is however not a panacea for automated verification. As many other analysis techniques, it still encounters significant difficulties when used on real-life systems. First, the stochastic modeling of these systems might be extremely

* Research supported by the European Community's Seventh Framework Programme [FP7] under grant agreements No. 257414 (ASCENS), No. 288175 (CERTAINTY), and the French BGLE project ACOSE.

cumbersome. Actually, high expertise is generally required to produce any kind of meaningful formal models. For stochastic models, besides functional aspects, they must include stochastic information in form of probabilities. These are hardly available and usually incomprehensible by an average system designer. Second, whenever such stochastic models exist, they can be very detailed and contain too much information than actually needed for verification purposes. This is usually the case when stochastic system models are automatically generated from higher level descriptions, e.g., as part of various designs/analysis flows [2]. In this case, Monte-Carlo simulation becomes problematic as individual simulation time (time to obtain a single execution trace) could be very long. Henceforth, it could not be possible to obtain any but only a limited number of traces and consequently, prevent the use of SMC techniques. Moreover, it is worth mentioning that for verification of system-level properties, the observation of any such trace in detail is rarely needed. Most of the time, such properties are expressed in terms of few observable actions/states of the system while the remaining are completely irrelevant and can be safely ignored.

Our aim is to improve general applicability of SMC techniques. In this work, we propose the combined use of abstraction and learning techniques to automatically construct faithful abstractions of system models and therefore to overcome the issues discussed earlier. Nowadays, machine learning is an active field of research and learning algorithms are constantly developed and improved in order to address new challenges and new classes of problems (see [26] for a recent survey on grammatical inference). In our context, learning is combined with abstraction as follows. Given a property of interest and a (usually large) sample of partial traces generated from a concrete system (model), we first use abstraction to restrict the amount of visible information on traces to the minimum required to evaluate the property and then, use learning to construct an abstract, probabilistic model which conforms to the abstracted sample set. Under some additional restrictions discussed later, the resulting model is a sound abstraction of the concrete model with respect to the satisfaction of the property. Hence, it can be used to correctly predict/generate the entire abstract behavior of the model, in particular, as an input model for SMC.

The above approach has multiple benefits. First of all, the sample set of traces can be generated directly from an existing black-box implementation of the system, as opposed to a concrete detailed model. In many practical situations, such detailed system models simply do not exist and the cost for building them using reverse-engineering could be prohibitive. In such cases, learning provides an effective, automated way to obtain a model and to get some valuable insight on the system behavior. The use of projection is also mostly beneficial. In most of the cases, the complexity of the learning algorithms as well as the complexity of the resulting models are directly correlated to the the number of distinct observations (the alphabet) of traces. Moreover, under normal considerations, a large alphabet requires a large size for the sample set. Intuitively, the more complex the final model is, the more traces are needed to learn it correctly. Nevertheless, one should mention that a bit of care is needed to meaningfully combine abstraction

and learning. That is, abstraction may change a deterministic model into a non-deterministic one, and henceforth has an impact on the learning algorithms needed for it.

The contributions of the paper are as follows. We propose a general approach to compute abstract stochastic models using learning and projection and discuss conditions under which the obtained model is a correct abstraction of the original system. We provide a first simple definition for a projection operator on execution traces given an LTL property and an implementation of the whole procedure. We finally validate the approach on the Herman's Self Stabilizing protocol. The obtained results show an important reduction of the model size, the SMC time, and accurate probability estimations of the verified properties.

The remainder of this paper is organized as follows. The basic formalisms for probabilistic modeling and learning techniques are briefly recalled in Section 2. In Section 3, we present our contribution, that is the joint use of abstraction and learning as a means to speed-up statistical model checking. We discuss the restrictions needed for convergence and correctness. In Section 4, we present the experimental set-up and concrete results. Related work is discussed in Section 5. Finally, conclusions and directions for future research are presented in Section 6.

2 Background

Let AP be a finite set of atomic propositions. We define the alphabet $\Sigma = 2^{AP}$ and denote the elements of Σ (subsets of AP) as symbols. The empty symbol is denoted by τ. As usual, we denote by Σ^ω (resp. Σ^*) the sets of infinite (resp. finite) words over Σ. For an infinite word $\sigma = \sigma_0\sigma_1...$ and $i \geq 0$, we define the ith suffix (resp. prefix) of σ as $\sigma[i..] = \sigma_i\sigma_{i+1}...$ (resp. as $\sigma[..i] = \sigma_0...\sigma_i$).

2.1 Probabilistic models

Definition 1. *A labeled Markov chain (LMC) M is a tuple $\langle S, \iota, \pi, L\rangle$ where,*

- *S is a finite set of states,*
- *$\iota : S \to [0,1]$ is the initial states distribution such that $\sum_{s\in S} \iota(s) = 1$,*
- *$\pi : S \times S \to [0,1]$ is the probability transition function such that for each $s \in S$, $\sum_{s'\in S} \pi(s,s') = 1$ and*
- *$L : S \to \Sigma$ is a state labeling function.*

A *run* is a possible behavior (infinite execution) of the LMC. A *trace* is the sequence of labels associated to the states of the run. Formally:

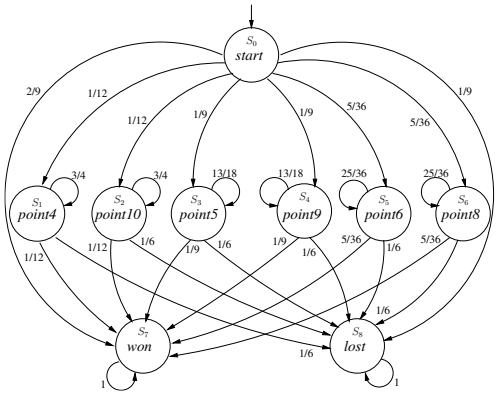
Definition 2. *Let $M = \langle S, \iota, \pi, L\rangle$ be a LMC. A run of M is an infinite sequence of states $s_0s_1...s_ns_{n+1}...$ such that $\iota(s_0) > 0$ and $\pi(s_i, s_{i+1}) > 0$, for all $i \geq 0$. A trace σ associated to a run $s_0s_1...s_ns_{n+1}...$ is the infinite word $L(s_0)L(s_1)...L(s_n)L(s_{n+1})....$ A finite run (resp. finite trace) is any finite prefix of a run (resp. trace).*

We denote by $Runs(M)$ the set of runs and by $Traces(M)$ the set of traces of M. Moreover, we denote by Pr_M the underlying probability measure induced by M on the set of its traces. This measure is well-defined in the context of Markov chains (see [1]). Two LMCs M_1 and M_2 are called equivalent, and denoted $M_1 \approx M_2$ if they have identical probability measures on traces, that is, $Pr_{M_1} = Pr_{M_2}$.

A labeled Markov chain is *deterministic* (DLMC) iff (i) $\exists s_0 \in S$ such that $\iota(s_0) = 1$, and (ii) $\forall s \in S$, $\forall \sigma \in \Sigma$ there exists at most one $s' \in S$ such that $\pi(s, s') > 0$ and $L(s') = \sigma$.

Probabilistic finite automata (PFA) are an alternative model for probabilistic systems. They are defined similarly to LMC with the following modification: π is now defined on $S \times S \cup \{\$\}$ and $\pi(s, \$)$ stands for the probability to terminate execution at state s. Henceforth, the associated notions of runs and traces correspond to finite runs and finite traces for a LMC. The probability of a finite run $\mathbf{s} = s_0 s_1 ... s_n$ of a PFA is $Pr(\mathbf{s}) = \iota(s_0) \cdot (\prod_{i=0}^{n-1} \pi(s_i, s_{i+1})) \cdot \pi(s_n, \$)$. Deterministic PFA are denoted as DPFA.

Example. We consider the Craps Gambling Game [1] as an illustrative example. A player starts by rolling two fair six-sided dice. The outcome of the two dice determines whether he wins or not. If the outcome is 7 or 11, the player wins. If the outcome is $2, 3$, or 12, the player looses. Otherwise, the dice are rolled again taking into account the previous outcome (called point). If the new outcome is 7, the player looses. If it is equal to point, he wins. For all other outcome, the dice are rolled again and the process continue until the player wins or looses. Figure 1 illustrates the DLMC model that describes the game behavior. A possible run of the DLMC below is $r = S_0 S_5 S_5 S_7 S_7 \ldots$ The corresponding trace is $t = start\ point6\ point6\ won\ won \ldots$ and $Pr(t) = 1 \times \frac{5}{36} \times \frac{25}{36} \times \frac{5}{36} \times 1 \times \ldots = 0.0277$.



$S = \{S_0, S_1, , ..., S_8\}$

$L(S_0) = start, \ldots, L(S_8) = lost$

$\iota(S_0) = 1$

$$\pi = \begin{array}{c} \\ S_0 \\ S_1 \\ \vdots \\ S_8 \end{array} \begin{array}{c} \begin{array}{cccc} S_0 & S_1 & \cdots & S_8 \end{array} \\ \begin{pmatrix} 0 & \frac{1}{12} & \cdots & \frac{1}{9} \\ 0 & \frac{3}{4} & \cdots & \frac{1}{12} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix} \end{array}$$

Fig. 1: A DLMC model for the Craps Gambling Game.

2.2 Probabilistic Linear Time Temporal Logic (PLTL)

The linear time temporal logic (LTL) formula φ built over a set of atomic propositions AP is defined by the following syntax:

$$\varphi := true \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid N\varphi \mid \varphi_1 U \varphi_2 \mid \varphi_1 U^i \varphi_2 \quad (p \in AP)$$

N, U and U^i are respectively the next, until and bounded until operators. Additional Boolean operators can be inferred from negation $\neg$ and conjunction $\wedge$. Moreover, temporal operators such as G (*always*) and F (*eventually*) are defined as $F\varphi \equiv true\ U\varphi$ and $G\varphi \equiv \neg F \neg\varphi$. The bounded fragment of LTL (denoted BLTL) restricts the use of the *until* operator U to its bounded variant U^i. LTL formula are interpreted on infinite traces $\sigma = \sigma_0\sigma_1 \ldots \in \Sigma^\omega$ as follows:

- $\sigma \vDash true$; $\sigma \vDash p$ iff $p \in \sigma_0$; $\sigma \vDash \neg\varphi$ iff $\sigma \nvDash \varphi$;
- $\sigma \vDash \varphi_1 \wedge \varphi_2$ iff $\sigma \vDash \varphi_1$ and $\sigma \vDash \varphi_2$; $\sigma \vDash N\varphi$ iff $\sigma[1..] \vDash \varphi$;
- $\sigma \vDash \varphi_1 U \varphi_2$ iff $\exists k \geq 0$ s.t. $\sigma[k..] \vDash \varphi_2$ and $\forall j \in [0, k[$ holds $\sigma[j..] \vDash \varphi_1$;
- $\sigma \vDash \varphi_1 U^i \varphi_2$ iff $\exists k \in [0, i]$ s.t. $\sigma[k..] \vDash \varphi_2$ and $\forall j \in [0, k[$ holds $\sigma[j..] \vDash \varphi_1$.

Definition 3. *Given an LMC M and an LTL property φ, the probability for M to satisfy φ denoted by $Pr(M \vDash \varphi)$ is given by the measure $Pr_M\{\sigma \in Traces(M) \mid \sigma \models \varphi\}$. In addition, we say that M satisfies φ denoted by $M \vDash \varphi$ iff $\forall \sigma \in Traces(M), \sigma \vDash \varphi$.*

Example. Given the Craps Gambling Game model in Figure 1, one could check for instance the following probabilistic (B)LTL properties. The probability to eventually loose is $Pr(F\ lost) = 0.51$, and the probability to win in two steps is $Pr(true\ U^2\ won) = 0.3$.

2.3 Statistical Model Checking (SMC)

Consider a stochastic system $\mathcal{S}$ and a property φ. SMC refers to a series of simulation-based techniques that can be used to answer two questions : (1) **Qualitative :** Is the probability for $\mathcal{S}$ to satisfy φ greater or equal to a certain threshold? and (2) **Quantitative :** What is the probability for $\mathcal{S}$ to satisfy φ? Contrary to numerical approaches, the answer is given up to some correctness precision. In the sequel, we overview two SMC techniques. Let B_i be a discrete random variable with a Bernoulli distribution of parameter p. Such a variable can only take 2 values 0 and 1 with $Pr[B_i = 1] = p$ and $Pr[B_i = 0] = 1 - p$. In our context, each variable B_i is associated with one simulation of the system. The outcome for B_i, denoted b_i, is 1 if the simulation satisfies φ and 0 otherwise.

Qualitative Answer. The main approaches [27, 24] proposed to answer the qualitative question are based on *hypothesis testing*. Let $p = Pr(\varphi)$, to determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$. A test-based solution does not guarantee a correct result but it is possible to bound the probability of error. The *strength* of a test is determined by two parameters, α and β,

such that the probability of accepting K (respectively, H) when H (respectively, K) holds, called a Type-I error (respectively, a Type-II error) is less or equal to α (respectively, β). A test has *ideal performance* if the probability of the Type-I error (respectively, Type-II error) is exactly α (respectively, β). However, these requirements make it impossible to ensure a low probability for both types of errors simultaneously (see [27] for details). A solution is to use an *indifference region* $[p_1, p_0]$ (given some δ, $p_1 = \theta - \delta$ and $p_0 = \theta + \delta$) and to test $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$. We now sketch the Sequential Probability Ratio Test (SPRT). In this algorithm, one has to choose two values A and B ($A > B$) that ensure that the strength of the test is respected. Let m be the number of observations that have been made so far. The test is based on the following quotient: $\frac{p_{1m}}{p_{0m}} = \prod_{i=1}^{m} \frac{Pr(B_i=b_i|p=p_1)}{Pr(B_i=b_i|p=p_0)} = \frac{p_1^{d_m}(1-p_1)^{m-d_m}}{p_0^{d_m}(1-p_0)^{m-d_m}}$, where $d_m = \sum_{i=1}^{m} b_i$. The idea is to accept H_0 if $\frac{p_{1m}}{p_{0m}} \geq A$, and H_1 if $\frac{p_{1m}}{p_{0m}} \leq B$. The algorithm computes $\frac{p_{1m}}{p_{0m}}$ for successive values of m until either H_0 or H_1 is satisfied. This has the advantage of minimizing the number of simulations.

Quantitative Answer. In [12, 17] Peyronnet et al. propose an estimation procedure to compute the probability p for $\mathcal{S}$ to satisfy φ. Given a *precision* δ, Peyronnet's procedure, which we call PESTIM, computes a value for p' such that $|p' - p| \leq \delta$ with *confidence* $1 - \alpha$. The procedure is based on the *Chernoff-Hoeffding bound* [14]. Let m be the number of simulations of the system and $p' = (\sum_{i=1}^{m} b_i)/m$, then Chernoff-Hoeffding bound [14] gives $Pr(|p' - p| > \delta) < 2e^{-\frac{m\delta^2}{4}}$. As a consequence, if we take $m \geq \frac{4}{\delta^2} \log(\frac{2}{\alpha})$, then $Pr(|p' - p| \leq \delta) \geq 1 - \alpha$.

2.4 Probabilistic Learning

Learning probabilities distributions over traces is a hard problem [7] with potential applications in a wide range of domains, far beyond formal verification. Many methods have been proposed in the research literature and are continuously improved and challenged on learning research competitions [26]. The family of state merging techniques is one of the most successful nowadays. Intuitively, these techniques proceed by first constructing some large automata-based representation of the set of input traces and then progressively compacting them, by merging states, into a smaller automaton, while preserving as much as possible trace occurrence frequencies/probabilities. Different algorithms in this family can learn either DPFA models [6, 9, 8] or general PFA models [25, 22, 10].

In this paper, we use AAlergia [19] which is a state merging algorithm that exclusively learn deterministic models. Given a sample of traces, the algorithm proceeds in three steps. It first builds an intermediate representation, a *Frequency Prefix Tree Acceptor* (FPTA), which is a restricted form of DPFA that represents all the traces in the input sample and their corresponding frequencies. Seconds, based on a compatibility criterion parametrized by α_A (automatically computed, as explained in [19]), it iteratively merges states of the FPTA having the same labels and similar probability distributions until reaching a compact DPFA. Finally, it transforms the obtained DPFA into a DLMC model.

AAlergia is proven to converge to the correct model in the limit [19] if the input traces are generated, with random lengths, from an LMC model. A first consequence concerns verification on DLMCs and ensures that, in the limit (with sufficiently big sample set of traces), a given LTL property will hold on the original and the learned model with the same probability. This result is partially extended to LMC. That is, for arbitrary Markov chain models, the algorithm might not converge to the good model in general. In the case of input traces from a non-deterministic LMC model (which moreover, does not have an equivalent deterministic representation), as the sample size increases, AAlergia will build a sequence of DLMCs (usually, of increasing size) tending to approximate the original model. It is however proven that, in the limit, these learned DLMC models provide an increasingly better approximation for the initial (prefix) behavior, and hence preserve the satisfaction of bounded LTL properties.

3 Learning Abstract Models

The verification problem in the stochastic setting amounts to compute $Pr(M \models \varphi)$ for an LMC model M and an LTL property φ. Moreover, M might not be explicitly known, that is, it could be a black-box probabilistic system which can be executed arbitrarily many times in order to produce arbitrarily long traces.

Due to the reasons introduced earlier, we would like to avoid the verification of φ on the original model M. Instead, we would like to perform it on a smaller, abstract model $M^\sharp$ which preserves the satisfaction probability of φ, that is, $Pr(M \models \varphi) = Pr(M^\sharp \models \varphi)$. We propose hereafter a method to compute such an abstraction $M^\sharp$ by combining learning and a projection operator on traces parametrized by the property φ. The idea is based on the simple observation that, when checking a model against a property, only a subset of the atomic propositions is really relevant. In fact, only the atomic propositions mentioned explicitly in the property are useful while the others can be safely ignored.

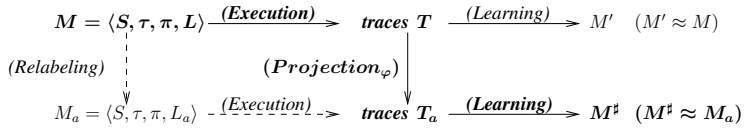


Fig. 2: Learning abstract models: approach overview.

The proposed approach is depicted in Figure 2. It consists of initially generating a finite set of random finite traces T (with random lengths) from M (Sampling). In a second step, a projection is applied on traces T in order to restrict the atomic propositions to the ones needed for the evaluation of the property φ. The projection is detailed below. Third, the set of projected traces is used as an input to a learning algorithm. For experiments, we have used AAler-

gia [19], however, any other algorithm could be used. The output of the learning denoted $M^\sharp$ on Figure 2 will be used to evaluate the property of interest φ.

It is worthwhile to mention that, in our approach, the sampling step (which could be time consuming) is done only once, while the following steps could be repeated given different properties. Our approach ensures a significant time reduction with respect to applying SMC directly on the black-box system since it generally requires trace re-sampling every time. In [24], re-sampling is avoided but raises confidence level issues as discussed in Section 5. In addition, some SMC algorithms, besides their termination guarantee, might potentially need a huge number/length of traces depending on the required confidence level.

The soundness of this approach is however justified only under particular considerations. Note that a projection may potentially introduce non-deterministic behavior at the level of traces. We then need to distinguish several cases. The first one is when the traces are generated from a DLMC and the projection operation does not introduce any non-determinism. In this case any learning algorithm should work, for instance, AAlergia. Another case is when the traces are generated from an LMC and/or the projection introduces non-determinism. This case is divided into two sub-cases depending on the type of non-determinism. If the non-deterministic model has an equivalent deterministic one, then any learning algorithm can be used. Otherwise, one needs to use learning algorithms capable to learn non-deterministic models such as [25, 10]. We detail the main steps of the approach and illustrate them on the running example. The correctness is formally established by Theorem 1.

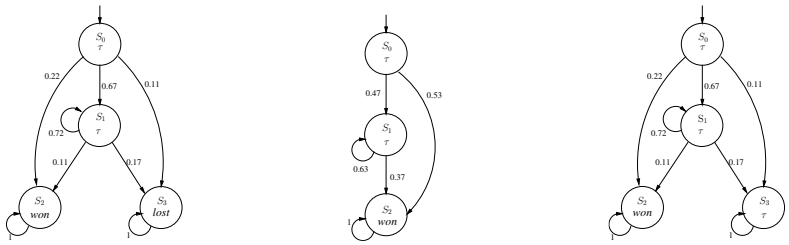
3.1 Main steps

Projection. The projection is defined on traces so as to reduce the number of labels and henceforth, later on, the number of states in the learned model. We introduce a first syntactic definition of a projection operator. It basically consists of ignoring the atomic propositions that are not relevant to the property under verification as formally defined below.

Definition 4. *Let $V_\varphi \subseteq AP$ called the support of φ be the set of atomic propositions occurring explicitly in φ. The projection $\mathcal{P}_\varphi : \Sigma^* \to \Sigma^*$ is defined as $\mathcal{P}_\varphi(\sigma_0\sigma_1...\sigma_n) = \sigma'_0\sigma'_1...\sigma'_n$ where $\sigma'_i = \sigma_i \cap V_\varphi$ for all $i \in [0, n]$.*

Example. Given a set T of traces generated from the Craps Gambling Game model in Figure 1 and the properties $\varphi_1 = F\ won$ and $\varphi_2 = F\ (won \vee lost)$, Definition 4 is applied to compute the corresponding sets of projected traces T_{a_1} and T_{a_2}: $T = \{$*start won, start lost lost, start won won won won won won won won won, start point5, start point10 point10 point10 point10 point10, start point9 point9,* $\ldots\}$; $T_{a_1} = \{\tau$ *won,* $\tau\ \tau\ \tau$, τ *won won won won won won won won won won,* $\tau\ \tau$, $\tau\ \tau\ \tau\ \tau\ \tau\ \tau$, $\tau\ \tau\ \tau$, $\ldots\}$; $T_{a_2} = \{\tau$ *won,* τ *lost lost,* τ *won won won won won won won won won won,* $\tau\ \tau$, $\tau\ \tau\ \tau\ \tau\ \tau\ \tau\ \tau$, $\tau\ \tau\ \tau$, $\ldots\}$

Learning. We briefly illustrate the learning phase using AAlergia on the running example. Figure 3 shows three learned models of the Craps Gambling Game obtained using the set T of 5000 traces generated from the model in Figure 1. One can note, out of this figure, the important reduction of the obtained models sizes with respect to the original one. Figure 3a shows the model learned by AAlergia taking as input the set T_{a_2}, that is, with respect to property $\varphi_2 = F\ (won \vee lost)$. Figure 3b is obtained by applying AAlergia on the set T_{a_1}, that is, projected with respect to $\varphi_1 = F\ won$. Remark that this model is not equivalent but only an approximation of the original model in Figure 1. That is, in the latter there exists some non null probability to never reach the *won* state. Whereas, in the learned model the *won* state is reachable with probability 1. This approximation could however improve if a larger set of traces is used for learning as stated in the previous section. Finally, the third learned model shown in Figure 3c is equally obtained from T_{a_1} but when using an algorithm able to learn non-deterministic models such as the one proposed by Stolcke [25].



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 3: Learned Markov Chains for Craps Gambling Game using 5000 traces.

Statistical Model Checking. The last step evaluates the considered property on the learned model. Table 1 provides results of verifying the property $\varphi_1 = F\ won$ on the Craps Gambling Game models. It shows that the model in Figure 3a exhibits similar probability to the original Craps model, whereas the one in Figure 3b shows different ones. The reason is that the projection introduced a non-determinism in the input sample. In addition, it seems that in this case there is no equivalent deterministic model that could be learned by AAlergia.

Models	$Pr(\varphi_1)$
Scenario 1(Figure 3a)	0.485
Scenario 2 (Figure 3b)	1
Original Model (Figure 1)	0.493

Table 1: Verifying φ_1 on the original and the learned Craps Gambling Game models using SMC (PESTIM).

3.2 Correctness

The correctness of our approach is formally stated as follows.

Theorem 1. *Let M be an LMC model and let φ be an LTL property. Let $M^\sharp$ be the learned model from a sample set of traces generated from M and projected according to φ as in definition 4. Then, in the limit, $M^\sharp$ is a correct abstraction for the verification of φ, that is $Pr(M \models \varphi) = Pr(M^\sharp \models \varphi)$ if either*

i) φ belongs to the bounded fragment BLTL and the learning algorithm converges for DLMC models, or
ii) the learning algorithm converges for arbitrary LMC models

Proof. First, let us remark that $M^\sharp$ is constructed as illustrated by the thick line in Figure 2. Let us moreover observe that any sample set of projected traces T_a obtained from M is equally obtained from M_a, that is, the "abstracted" version of M where only the labeling function has changed from L into L_a by taking $L_a(s) = L(s) \cap V_\varphi$, for all $s \in S$. In other words, the left-hand side of the diagram shown in Figure 2 commutes. Henceforth, M and M_a are identical with respect to the satisfaction of φ. The underlying set of runs and their associated probabilities are the same in M and M_a. As the atomic propositions occurring in φ are preserved by relabeling, it obviously holds that $Pr(M \models \varphi) = Pr(M_a \models \varphi)$.

Moreover, learning from the sample set T_a leads eventually to M_a. That is, under particular restrictions specific to the learning algorithms and limit conditions, the learned model $M^\sharp$ will be an equivalent representation of M_a, that is, $M^\sharp \approx M_a$. We distinguish two cases depending on the learning algorithm:

i) In the case of deterministic models learning (e.g., AAlergia), the learned model $M^\sharp$ is provable equivalent only for a deterministic input model M_a. But, in addition, for the general case, this models is also providing good approximations for the initial (prefix) behavior of M_a and hence preserve the probability of satisfaction for properties in the BLTL fragment (see Theorem 3 in [19]). Thereof, by using AAlergia or a similar learning algorithm, it holds that $Pr(M_a \models \varphi) = Pr(M^\sharp \models \varphi)$ whenever φ belongs to BLTL.
ii) In the general case of non-deterministic models learning, it is guaranteed in the limit that $M_a \approx M^\sharp$. Thereof, one can safely conclude that $Pr(M_a \models \varphi) = Pr(M'_a \models \varphi)$ for any φ.

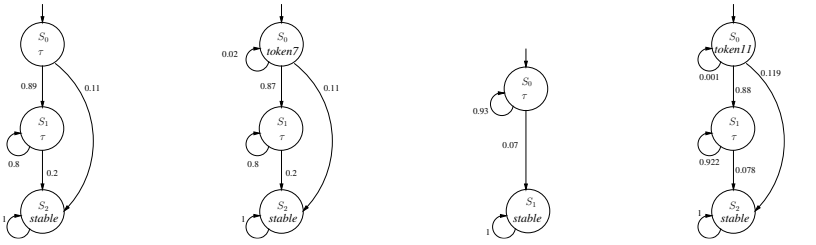
Henceforth, in both cases it holds $Pr(M \models \varphi) = Pr(M^\sharp \models \varphi)$. □

4 Case study: Herman's Self Stabilizing Protocol

To experiment our approach, we use the Herman's Self Stabilizing Protocol [13]. The goal of such a protocol is to perform fault tolerance by enabling a distributed system starting in an arbitrary state to converge to a legitimate one in a finite time. Given a token ring network where the processes are indexed from 1 to N (N must be odd) and ordered anticlockwise, the algorithm operates synchronously.

Processes can possess tokens, which circulate in one direction around the ring. At every step, each process with a token tosses a coin. Depending on the outcome, it decides to keep it or to pass it on to the right neighbor. When a process holds two tokens, they are eliminated. Thus, the number of tokens remains odd. The network is said to be stable if exactly one process has a token. Once such a configuration is reached, the token circulate forever, fairly, around the ring.

We apply our abstraction approach to several configurations ($N = 7, 11, 19, 21$) of the protocol. Note that as the number of processes increases, the state space becomes very large and makes the verification quite heavy even using simulation-based methods such as SMC. We use AAlergia for learning and show that we are able to reduce the state space while still accurate for several properties. We consider the bounded properties $\varphi^L = Pr(true\ U^L\ stable)$ and $\psi_N^L = Pr(tokenN\ U^L\ stable)$ where N is the number of processes in the network and L is a bound. The first property states that the protocol reaches the *stable* state in L steps whatever the intermediate ones are. The second specifies in addition that the protocol directly moves from N tokens to the *stable* state (1 token), that is, all the states before *stable* are *tokenN*. We first apply the projection on the traces generated from the different configurations using the properties supports $V_{\varphi^L} = \{stable\}$ and $V_{\psi_N^L} = \{tokenN, stable\}$. Then, we use AAlergia to learn the corresponding models shown in Figure 4. The models for $N = 19, 21$ are similar to $N = 11$ and are omitted for space constraints.



(a) Given φ^L (b) Given ψ_7^L (c) Given φ^L (d) Given ψ_{11}^L

Fig. 4: Learned (AAlergia) Herman's models for $N = 7$ (a,b) and $N = 11$ (c,d).

Table 2 summarizes the learned models characteristics, AAlergia performance when combined with projection, and properties verification results using PRISM [16]. In this table, the first two columns list the used configurations and their corresponding sizes. The third column depicts the properties under consideration. Information about the learning process are then detailed: α_A is the AAlergia compatibility criterion parameter, *Size* is the learned model size, and *Time* is the learning time in seconds. The last part concerns the comparison of the original and the learned model in term of properties probabilities and verification time. The verification part relies on the PESTIM algorithm which is parametrized by two confidence parameters δ and α.

The results in Table 2 point out two important facts. The first is the drastic reduction of the learned models sizes and SMC time compared to the original Herman's Self Stabilizing protocol. Figure 5 summarizes the SMC time of φ^{10} for the learned and the original models when increasing N. The figure and the table allow us to see how big is the SMC time of the original model with respect to the learned one. Figure 5 shows in addition the learning time which is also far below the SMC time of the original model for $N > 11$. Moreover, one can see that the time to learn plus SMC the learned model is below the SMC time of the original model for $N > 11$ which confirms the pertinence of our approach for big models. For instance, for $N = 19$, the learning took about 83 seconds and SMC the learned model about 0.307 seconds while SMC the original one took about 13 hours. Furthermore, since the sampling step is done only once in our approach, its time impact is reduced when considering many properties. The second fact is that, besides this reduction, the models are quite accurate in terms of probability measures as clearly shown in the table and Figures 6a,6b and 6c. These figures show the verification results of φ^L (for different L) on the original protocol versus the learned model for all the considered configurations.

	Size	*Prop.*	*Learning*			*SMC*				
						δ, α	*Learned Model*		*Original Model*	
			α_A	*Size*	*Time(s)*		*Pr*	*Time(s)*	*Pr*	*Time*
$N = 7$	2^7	φ^{10}	$[2^{-9}, 2^0]$	3	69.70	$10^{-2}, 10^{-1}$	0.874	0.180	0.874	3.40 *s*
						$10^{-2}, 10^{-2}$	0.880	0.320	0.873	5.44 *s*
		ψ^{30}	$[2^{-6}, 2^6]$	3	45.98	$10^{-2}, 10^{-1}$	0.112	0.050	0.112	0.93 *s*
						$10^{-2}, 10^{-2}$	0.109	0.111	0.111	1.51 *s*
		ϕ	$[2^{-8}, 2^0]$	4	167.50	–	0.160	0.005	0.167	0.02 *s*
		Sample Size = 5000								
$N = 11$	2^{11}	φ^{10}	$[2^{-4}, 2^6]$	2	54.67	$10^{-2}, 10^{-1}$	0.517	0.250	0.543	33.1 *s*
						$10^{-2}, 10^{-2}$	0.518	0.440	0.543	58.3 *s*
		ψ^{30}	$[2^{-6}, 2^6]$	3	60.22	$10^{-2}, 10^{-1}$	0.011	0.039	0.012	12.1 *s*
						$10^{-2}, 10^{-2}$	0.012	0.070	0.011	21.7 *s*
		Sample Size = 5000								
$N = 19$	2^{19}	φ^{10}	$[2^{-4}, 2^6]$	2	82.95	$10^{-2}, 10^{-1}$	0.197	0.180	0.148	8.1 *h*
						$10^{-2}, 10^{-2}$	0.191	0.307	0.151	13.3 *h*
		ψ^{30}	$[2^{-6}, 2^6]$	3	172.58	$10^{-2}, 10^{-1}$	0.000	0.040	0.0001	5.7 *h*
						$10^{-2}, 10^{-2}$	0.000	0.074	0.0008	10.1 *h*
		Sample Size = 10000								
$N = 21$	2^{21}	φ^{10}	$[2^{-10}, 2^0]$	3	253.71	$10^{-2}, 10^{-1}$	0.169	0.355	0.172	34 *h*
						$10^{-2}, 10^{-2}$	0.163	0.616	–	> 5 *d*
		Sample Size = 10000								

Table 2: Abstraction and verification results of φ^{10} and ψ^{30} using PESTIM.

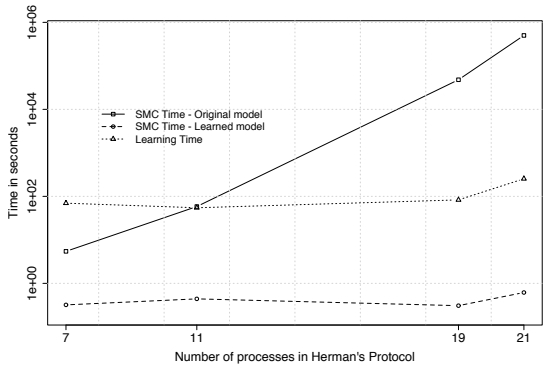
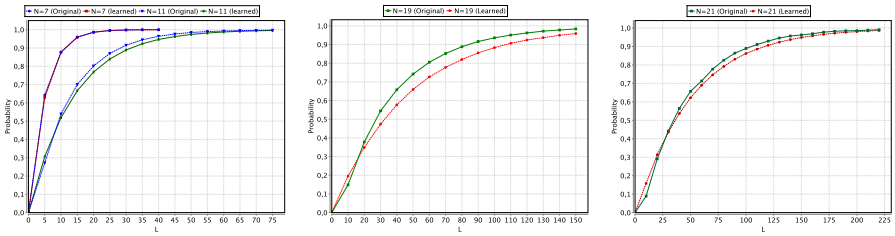


Fig. 5: PESTIM $(10^{-2}, 10^{-2})$ time: original vs. learned Herman's model for φ^{10}.



(a) $N = 7, 11$. (b) $N = 19$. (c) $N = 21$.

Fig. 6: φ^L verification results using PESTIM for $N = 7, 11, 19$, and 21. The results for $N = 21$ are obtained with PESTIM $(5.10^{-2}, 5.10^{-3})$.

	L	*Original Model*			*Learned Model*		
		θ	*Traces*	*Time(s)*	θ	*Traces*	*Time(s)*
$N = 7$	$L = 1$	$[0.109, 0.110[$	622018	25.643	$[0.107, 0.108[$	588357	1.363
	$L = 30$	$[0.111, 0.112[$	622834	25.749	$[0.108, 0.109[$	533885	1.282
	$L = 65$	$[0.111, 0.112[$	651434	26.756	$[0.108, 0.109[$	476883	1.118
$N = 11$	$L = 1$	$[0.011, 0.012[$	147178	85.135	$[0.012, 0.013[$	163600	0.411
	$L = 30$	$[0.011, 0.012[$	105362	60.206	$[0.013, 0.014[$	098493	0.262
	$L = 65$	$[0.011, 0.012[$	137469	80.648	$[0.013, 0.014[$	248300	0.564

Table 3: SPRT $(10^{-3}, 10^{-3})$ of ψ_N^L on the original and the learned models.

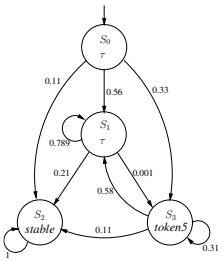


Fig. 7: Learned Herman's protocol model ($N = 7$) using AAlergia given ϕ.

In addition to PESTIM, we used the SPRT technique to validate with more confidence the results of the property $\psi_N^L = Pr(tokenN\ U^L\ token1) >= \theta$ for $N = 7, 11$. We fixed the confidence parameters to $\alpha = \beta = 10^{-3}$ and $\delta = 10^{-3}$. Table 3 shows the verification results and performance (verification time and number of traces) for different L values. Note that for this experiment, we used the same model learned previously. In this table, θ is the probability range to satisfy ψ_N^L, *Traces* is the number of traces used by SPRT, and *Time* is the SMC time. This table confirms the observation made in the previous experiment, that is, the reduction of the SMC time when using the abstract model while the probability estimation still accurate.

We did an additional property $\phi = Pr(X(token5\ U\ stable))$ for Herman's protocol with $N = 7$ in order to investigate the usability of this instance of the approach for unbounded properties (all the considered properties so far where bounded). The corresponding learned model is shown in Figure 7 and the verification results are depicted in Table 2. The obtained results show that the probability of satisfying ϕ is almost the same for the learned and the original protocol. This is possible (to check unbounded LTL properties on a learned model with a good accuracy) because, in this case, there exist an equivalent deterministic model to the original Herman's protocol that AAlergia succeed to learn. Since ϕ is unbounded, we rely on classical probabilistic model checking using PRISM.

5 Related Work

We first review some applications of learning techniques for systems verification. For more details, we refer the reader to the literature survey from Martin Leucker [18]. Pena et al. propose to use learning for the purpose of state reduction in incompletely specified finite state machines [21]. Based on Angluin's L* algorithm, which computes the minimal DFA in polynomial time, the authors propose a learning technique that produces an equivalent, reduced finite state machine. In contrast, our work relies on the AAlergia algorithm and assumes that the input data is generated from an LMC. Peled et al. propose to combine model checking, testing, and learning to automatically check properties of systems whose structure is unknown [20]. This paper motivates black-box checking where a user performs acceptance tests and does not have access to the design,

nor to the internal structure of the system. The authors, however, conclude that the complexity of their algorithms could be reduced if an abstract model of the system would be available. Additionally, the authors pointed out the need to take into account the property of interest to tackle verification complexity.

Among the works aiming to improve SMC applicability, we mention Sen et al. SMC algorithm for black-box systems [24]. In this work, systems are assumed to be uncontrolled, that is, traces can not be generated on demand. Hence, the approach cannot guarantee a correct answer within required error bounds. It computes instead a p-value as a confidence measure. While our approach is not making such an assumption, it also uses a pre-generated set of traces to learn an abstract model which is given as input to SMC. In contrast, [24] uses the pre-generated traces as direct input to their SMC algorithm. This raises the confidence issue but makes it faster since no learning is performed.

6 Conclusion

Reducing the SMC time of a given LTL property on a large stochastic system is the primary benefit of our abstraction approach. This gain is achieved through the combined use of projection on traces and learning. Projection is performed by considering the support of the property of interest, that is, the set of symbols explicitly appearing in that property. The approach could be instantiated with any learning algorithm. Although, this must respect the conditions discussed earlier to produce accurate models preserving the probability of the property under verification. Experimental results show that (1) verifying the properties of interest on the abstract model is faster than the original one, and that (2) the estimation of the probability of satisfying these properties is accurate with respect to the one obtained on the original system.

The proposed projection definition is currently quite simple. It allowed us to instantiate our methodology and to implement it for validation. As future work, we plan to improve it such that to obtain coarser abstractions, yet preserving the probability of the underlying property (as opposed to a class of properties currently). This could be potentially achieved by taking into account the LTL operators semantics. We shall also apply the approach to other real-life systems and consider using other algorithms able to learn non-deterministic models. Furthermore, our proposed approach is applicable to discrete stochastic systems. An interesting direction to investigate is its extension to continuous systems, such as continuous time Markov chains [23] or probabilistic timed automata.

References

1. C. Baier and J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.
2. A. Basu, S. Bensalem, M. Bozga, P. Bourgos, M. Maheshwari, and J. Sifakis. Component assemblies in the context of manycore. In *FMCO'11*, pages 314–333.
3. S. Bensalem, M. Bozga, B. Delahaye, C. Jégourel, A. Legay, and A. Nouri. Statistical Model Checking QoS Properties of Systems with SBIP. In *ISoLA (1)*, pages 327–341, 2012.

4. S. Bensalem, B. Delahaye, and A. Legay. Statistical model checking: Present and future. In *RV*, volume 6418 of *LNCS*. Springer, 2010.
5. P. E. Bulychev, A. David, K. G. Larsen, M. Mikucionis, D. B. Poulsen, A. Legay, and Z. Wang. Uppaal-smc: Statistical model checking for priced timed automata. In *QAPL'12*, pages 1–16, 2012.
6. R. C. Carrasco and J. Oncina. Learning Stochastic Regular Grammars by Means of a State Merging Method. In *ICGI*, pages 139–152, 1994.
7. C. de la Higuera. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, New York, NY, USA, 2010.
8. C. de la Higuera and J. Oncina. Identification with Probability One of Stochastic Deterministic Linear Languages. In *ALT*, pages 247–258, 2003.
9. C. de la Higuera, J. Oncina, and E. Vidal. Identification of DFA: data-dependent vs data-independent algorithms. In *ICGI*, pages 313–325, 1996.
10. F. Denis, Y. Esposito, and A. Habrard. Learning rational stochastic languages. COLT'06, 2006.
11. R. Grosu and S. A. Smolka. Monte carlo model checking. In *TACAS*, volume 3440 of *LNCS*, pages 271–286. Springer, 2005.
12. T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate Probabilistic Model Checking. In *VMCAI*, pages 73–84, 2004.
13. T. Herman. Probabilistic self-stabilization. *Information Processing Letters*, 35(2):63–67, 1990.
14. W. Hoeffding. Probability inequalities. *Journal of the American Statistical Association*, 58:13–30, 1963.
15. C. Jégourel, A. Legay, and S. Sedwards. A platform for high performance statistical model checking - plasma. In *TACAS*, LNCS, pages 498–503, 2012.
16. M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: verification of probabilistic real-time systems. CAV, pages 585–591, 2011.
17. S. Laplante, R. Lassaigne, F. Magniez, S. Peyronnet, and M. de Rougemont. Probabilistic abstraction for model checking: An approach based on property testing. *ACM TCS*, 8(4), 2007.
18. M. Leucker. Learning Meets Verification. In *FMCO*, pages 127–151, 2006.
19. H. Mao, Y. Chen, M. Jaeger, T. D. Nielsen, K. G. Larsen, and B. Nielsen. Learning Probabilistic Automata for Model Checking. In *QEST*, pages 111–120, 2011.
20. D. Peled, M. Y. Vardi, and M. Yannakakis. Black box checking. *J. Autom. Lang. Comb.*, 7(2):225–246, Nov. 2001.
21. J. M. Pena and A. L. Oliveira. A new algorithm for exact reduction of incompletely specified finite state machines. *TCAD*, 18(11):1619–1632, 2006.
22. D. Ron, Y. Singer, and N. Tishby. On the learnability and usage of acyclic probabilistic finite automata. COLT, pages 31–40, 1995.
23. K. Sen, M. Viswanathan, and G. Agha. Learning continuous time markov chains from sample executions. In *QEST*, pages 146–155, 2004.
24. K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. In *CAV*, LNCS 3114, pages 202–215. Springer, 2004.
25. A. Stolcke. *Bayesian Learning of Probabilistic Language Models*. PhD thesis, Berkeley, CA, USA, 1994. UMI Order No. GAX95-29515.
26. S. Verwer, R. Eyraud, and C. de la Higuera. Results of the pautomac probabilistic automaton learning competition. In *ICGI*, pages 243–248, 2012.
27. H. L. S. Younes. *Verification and Planning for Stochastic Processes with Asynchronous Events*. PhD thesis, Carnegie Mellon, 2005.