Verifying Finite State Machines in Probabilistic Environments

Markus Siegle
Universität der Bundeswehr München, Institut für Technische Informatik

Abstract: Finite state machines are considered in a probabilistic environment that is generated by a Markov chain. An expressive temporal logic is introduced for specifying complex requirements that the FSM should satisfy in the given environment. The corresponding model checking algorithm is described and its symbolic implementation is sketched. Throughout the paper, the method is illustrated by a simple running example.
Keywords: FSM, Markov chain, hardware verification, model checking, BDD.

1 Introduction

Finite state machines (FSM) play an important role as a general model for sequential logic circuits and control circuits. Using the concept of FSMs, such circuits can be specified at an abstract level and then implemented more or less automatically in (programmable) hardware. This paper proposes a method for analysing the behaviour of FSMs by means of model checking. The FSM under investigation is exposed to a probabilistic environment which generates the sequence of input vectors according to a specific stochastic law. The combination of the FSM and its environment is then a stochastic model which serves as basis for verification. A new temporal logic, more expressive than PCTL [5], is presented which enables the user to specify complex requirements, covering both functional and timing properties of the FSM.

The method is illustrated in Fig. 1a. The system model $\mathcal{F}$ and the environment model $\mathcal{M}$ are shown at the top left of the figure. Their combination yields the labelled Markov chain $\mathcal{P}$. The requirement to be checked, specified by means of a temporal logic, is fed – together with the Markov chain $\mathcal{P}$ – into the model checking engine. This engine contains the algorithms (including Boolean evaluation, graph analysis and numerical solution of systems of linear equations) for determining whether a given requirement is satisfied or not. The verification results are then output in the form of truth values augmented by numerical data as indicated at the bottom of the figure.

FSMs with Markovian behaviour have been investigated in different contexts. Marculescu et al. computed the steady-state probabilities of a FSM whose inputs are generated by a so-called stochastic state machine which exhibits Markovian behaviour [10]. The problem of computing the steady-state probabilities for very large FSMs whose inputs are Markovian was addressed by Hachtel et al. [4]. These authors use Algebraic Decision Diagrams (ADDs) as a symbolic data structure, and take the decomposable structure of the FSM into account. On the verification side, the well-known logic PCTL, interpreted over state-labelled Markov chains, was developed by Hansson and Jonsson [5] and has been extended since in various directions. One of the most powerful such extensions is the work by Baier et al. [2], which describes the logic asCSL to be interpreted over state- and action-labelled Continuous-Time Markov Chains, where it is possible to characterise satisfying paths with the help of regular expressions of actions and so-called tests. The verification algorithms for probabilistic logics have been implemented in model checking tools such as ETMCC [6], PRISM [9] and CASPA [8], where the latter two rely on space-efficient symbolic data structures.

The rest of this paper is structured as follows: Sec. 2 formally introduces the system model and the environment model and how they are combined. The temporal logic for requirement specification and the associated model checking algorithm are discussed in Sec. 3, and Sec. 4 concludes the paper. Both Secs. 2 and 3 contain a short discussion of how the method can be implemented symbolically. Throughout the paper, the method is illustrated by means of the running example of a simple bus arbiter.

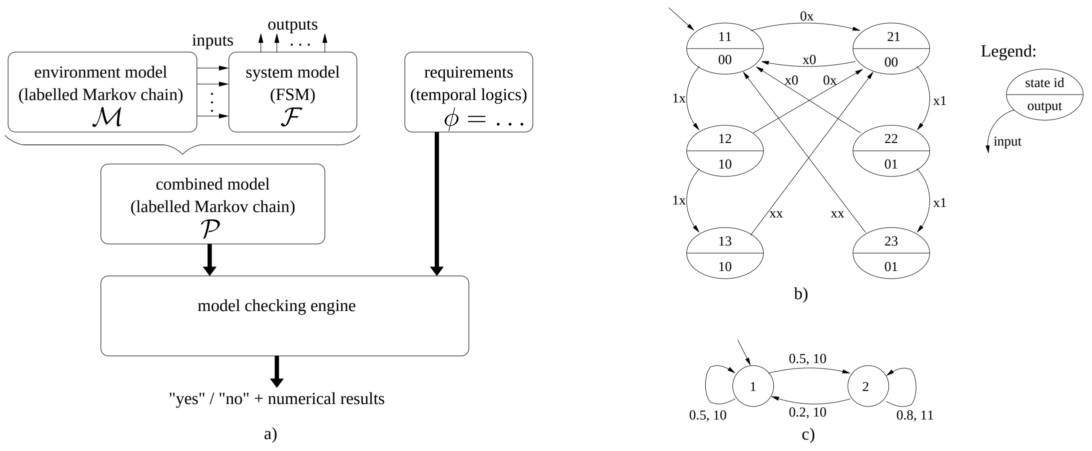

Figure 1: a) Overview of the proposed method. b) FSM model $\mathcal{F}$ of a two cell bus arbiter. c) Environment model $\mathcal{M}$.

2 Model Description

System Model: A FSM in the style of Moore-automata is defined as follows:

Definition 2.1 *Let the input alphabet $I = \mathbb{B}^n$ and the output alphabet $O \subseteq \mathbb{B}^m$ be given. A Finite State Machine (FSM) is a tuple $\mathcal{F} = (S_\mathcal{F}, s^0_\mathcal{F}, n_\mathcal{F}, o_\mathcal{F})$ where $S_\mathcal{F}$ is a finite set of states, $s^0_\mathcal{F} \in S_\mathcal{F}$ is the initial state, $n_\mathcal{F} : (S_\mathcal{F} \times I) \mapsto S_\mathcal{F}$ is the next state function, and $o_\mathcal{F} : S_\mathcal{F} \mapsto O$ is the output function.*

The input to the FSM is a vector of Booleans corresponding to the n primary inputs of the system. The output of the system is again a Boolean vector (of length m). For the purpose of verification, each of the m output signals will be interpreted as an atomic proposition which is either true or false in a given state. The system model is a clocked system, i.e. in every clock cycle the system evaluates its inputs and takes a transition as specified by function $n_\mathcal{F}$, leading to the successor state which may be identical to the current state in the case of self-loops.

Example 1 *The system model $\mathcal{F}$ for a simple two-cell bus arbiter is shown in Fig. 1b. The arbiter has $n = 2$ input signals (req_1 and req_2) and $m = 2$ output signals (gnt_1 and gnt_2). Upon request from cell i, the arbiter will eventually issue a grant to that cell, and the bus holding time for a particular cell is restricted to a maximum of two clock cycles. The bus is granted to the cells in a round-robin fashion. The notation used in the figure is explained in the legend shown on the right of the state graph: The state identifier is given in the top half of each state and the output signals are shown in the lower half, where a "1" means that the grant signal for the corresponding cell is set to one. The arcs are labelled by the vector of input signals, where an "x" means a don't care of the corresponding request signal. In state $i1$ the arbiter polls cell i ($i = 1, 2$), but no grant signal is issued. In states 12 and 13 the bus is granted to cell 1 for a first and second cycle, repectively. States 22 and 23 model the symmetric behaviour for cell 2.*

The arbiter can easily be extended to more than two cells and to longer maximum bus holding times, but we chose this small configuration in order to keep the running example simple.

Environment Model: The environment creating the stimuli for the FSM to be verified is an action-labelled deterministic-time Markov chain (DTMC) defined as follows:

Definition 2.2 *Let an alphabet of actions $I = \mathbb{B}^n$ be given. An action-labelled deterministic-time Markov Chain (al-DTMC) is a tuple $\mathcal{M} = (S_\mathcal{M}, s^0_\mathcal{M}, \delta_\mathcal{M})$ where $S_\mathcal{M}$ is a finite set of* *states, $s^0_{\mathcal{M}} \in S_{\mathcal{M}}$ is the initial state, $\delta_{\mathcal{M}} \subseteq S_{\mathcal{M}} \times [0,1] \times I \times S_{\mathcal{M}}$ is the transition relation, such that*

$$\forall s \in S_{\mathcal{M}} : \sum_{(s,p,i,s') \in \delta_{\mathcal{M}}} p \;=\; 1 \tag{1}$$

If $(s, p, i, s') \in \delta_{\mathcal{M}}$ we also use the notation $s \xrightarrow{p,i} s'$. The value $p \in [0,1]$ denotes the probability with which the al-DTMC moves from state s to state s', thereby generating the signal vector $i \in I$, which is going to be the stimulus for the FSM[1]. Condition (1) states that the sum of all probabilities emanating from a state is equal to one. This also implies that the environment model will keep generating stimuli forever, even if it stays in some “absorbing” state by a self-loop. Similar to the system model, the environment model is also a clocked system, i.e. at every clock cycle it takes a probabilistic decision, produces the vector of stimuli and moves to the next state (which is possibly identical to the current state).

Example 2 *A possible environment model $\mathcal{M}$ for the bus arbiter from Fig. 1b is shown in Fig. 1c. It is a Markov chain with just two states, state 1 being the initial state. Each arc is labelled with a transition probability and a vector of stimuli generated by the Markov chain. $\mathcal{M}$ models a rather special environment, since the only vectors of stimuli generated are 10 and 11.*

Combined Model: The synchronous parallel composition of a FSM as system model and an al-DTMC as environment model yields an action- and state-labelled DTMC (asl-DTMC) which will be used as basis for verification.

Definition 2.3 *Let the FSM $\mathcal{F} = (S_{\mathcal{F}}, s^0_{\mathcal{F}}, n_{\mathcal{F}}, o_{\mathcal{F}})$ and the al-DTMC $\mathcal{M} = (S_{\mathcal{M}}, s^0_{\mathcal{M}}, \delta_{\mathcal{M}})$ be given. Their synchronous product is an action- and state-labelled DTMC (asl-DTMC) which is a tuple $\mathcal{P} = (S_{\mathcal{P}}, s^0_{\mathcal{P}}, \delta_{\mathcal{P}}, o_{\mathcal{P}})$ where $S_{\mathcal{P}} = S_{\mathcal{F}} \times S_{\mathcal{M}}$ is the finite set of states, $s^0_{\mathcal{P}} = (s^0_{\mathcal{F}}, s^0_{\mathcal{M}}) \in S_{\mathcal{P}}$ is the initial state, $\delta_{\mathcal{P}} \subseteq S_{\mathcal{P}} \times [0,1] \times I \times S_{\mathcal{P}}$ is the transition relation, defined as*

$$(((s_{\mathcal{F}}, s_{\mathcal{M}}), p, i, (s'_{\mathcal{F}}, s'_{\mathcal{M}})) \in \delta_{\mathcal{P}}) \quad \textit{iff} \quad (n_{\mathcal{F}}(s_{\mathcal{F}}, i) = s'_{\mathcal{F}} \;\; \textit{and} \;\; (s_{\mathcal{M}}, p, i, s'_{\mathcal{M}}) \in \delta_{\mathcal{M}}),$$

and $o_{\mathcal{P}} : S_{\mathcal{P}} \mapsto O$, defined as $o_{\mathcal{P}}((s_{\mathcal{F}}, .)) = o_{\mathcal{F}}(s_{\mathcal{F}})$, is the output function.

The asl-DTMC $\mathcal{P}$, essentially the product of the FSM $\mathcal{F}$ and the al-DTMC $\mathcal{M}$, inherits the next state function and the output function from $\mathcal{F}$ and the transition probabilities from $\mathcal{M}$. Due to the definition of the transition relation $\delta_{\mathcal{P}}$, not all states of $S_{\mathcal{P}}$ may be reachable from the initial state $s^0_{\mathcal{P}}$. It should be noted that the human user does not have to concern himself with this combined model $\mathcal{P}$, since it is generated automatically in order to be used as the basis of the subsequent verification procedure.

Example 3 *The combined model $\mathcal{P}$ obtained from the synchronous parallel composition of the system $\mathcal{F}$ and the environment $\mathcal{M}$ from Fig. 1 is shown in Fig. 2. The state identifiers, shown in the top half of each state, are now of the form aa/b, where aa denotes the state of the system $\mathcal{F}$ and b that of the environment $\mathcal{M}$. The lower half of each state contains the output vector (i.e. the grant signals). Each arc is labelled by the transition probability and the vector of stimuli (i.e. the request signals). It is noteworthy that out of the $6 \cdot 2 = 12$ states of the product space $S_{\mathcal{F}} \times S_{\mathcal{M}}$ only 10 are actually reachable from the initial state.*

[1] In practice, one usually has $I \subseteq \mathbb{B}^n$ instead of $I = \mathbb{B}^n$ as action set of the environment model $\mathcal{M}$. We chose the latter notation in order to indicate that the stimuli generated by $\mathcal{M}$ correspond exactly to the input alphabet of the system model $\mathcal{F}$.

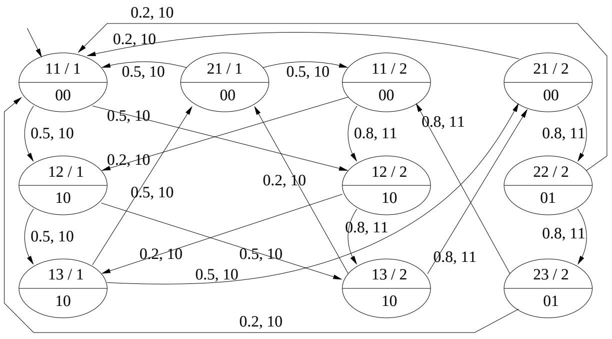

Figure 2: Combined model $\mathcal{P}$ for the bus arbiter.

Implementation with BDDs: Binary Decision Diagrams (BDD), a graph-based data structure for the representation of Boolean functions, have been used with great success for the compact representation and analysis of very large transition systems. In the present context, BDDs are used to represent the FSM under investigation, in particular the next-state function of the FSM is encoded as a BDD. For this purpose, a characteristic function in disjunctive normal form is generated, where each term encodes the source state and the input as well as the target state of a given state-to-state transition. Similarly, we employ multi-terminal BDDs[2] (MTBDD) [3, 1] for representing the transition relation of the Markovian environment model, where the probabilities associated with the individual transitions are stored in the terminal vertices of the decision diagram.

Based on a BDD F representing the FSM $\mathcal{F}$ and an MTBDD M representing the environment $\mathcal{M}$, the MTBDD P representing the combined model $\mathcal{P}$ is constructed by a single call to the standard *apply*-function for MTBDDs [3] as

$$\mathsf{P} = apply(\mathsf{F}, \mathsf{M}, \cdot) \tag{2}$$

By an argument similar to the one used in [7] it can be shown that this product construction leads only to an additive growth of the symbolic data structure, i.e. the number of vertices of the resulting MTBDD P is a constant times the sum of the number of the vertices of F and M. This is a very strong result, since the state space encoded by the symbolic data structures exhibits a multiplicative growth. The MTBDD P constructed according to Eq. (2) encodes all transitions of the combined model $\mathcal{P}$, i.e. not only the ones that are reachable from its initial state. Therefore, P is subsequently restricted to its reachable portion by applying a standard symbolic reachability algorithm.

3 Verification

Logic for Requirement Specification: In this section, a new temporal logic $\mathcal{L}$ is defined, to be used for specifying the requirements that the FSM should satisfy in a given environment.

Definition 3.1 *Let a set of atomic propositions AP (with $|AP| = m$) and a set of actions $I (= \mathbb{B}^n)$ be given. The set of valid state formulas Φ of the temporal logic $\mathcal{L}$ can be constructed according to the following grammar:*

$$\Phi \ ::= \ q \mid \neg\Phi \mid \Phi \wedge \Phi \mid S_{\bowtie p}(\Phi) \mid P_{\bowtie p}(\Phi[\rho]^k\Phi)$$

where $q \in AP$ is an atomic proposition, $\bowtie \in \{<, \leq, \geq, >\}$ is a comparison operator, $p \in [0,1]$ is a probability value, $k \in \mathbb{N} \cup \{\infty\}$ is a step counter, and ρ is a regular expression of actions defined as

$$\rho \ ::= \ \epsilon \mid a \mid \rho;\rho \mid \rho + \rho \mid \rho^*$$

with $a \in I$ being an action.

[2]MTBDDs are also called Algebraic Decision Diagrams (ADD).

The state formulas of $\mathcal{L}$ are interpreted over the states of an asl-DTMC $\mathcal{P}$. The notation $s \models \phi$ denotes that state s satisfies state formula ϕ. The following is an informal explanation of the semantics[3] of $\mathcal{L}$: A state s satisfies $\phi = q \in AP$ iff in state s the output signal corresponding to q is set. The logical operators negation ($\neg$) and conjunction ($\wedge$) have their usual meaning, and all other logical operators (such as disjunction or implication) can be derived from them. State s satisfies the steady-state formula $S_{\bowtie p}(\phi)$ iff starting from s as the initial state the steady-state probability of being in a state satisfying ϕ is within the bound as specified by $\bowtie p$. State s satisfies the quantised path formula formula $P_{\bowtie p}(\phi_1[\rho]^k\phi_2)$ iff the probability measure of the set of satisfying paths emanating from s is within the bound as specified by $\bowtie p$. A path is satisfying, if it is of the form $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots \xrightarrow{a_l} s_l$, such that $l \leq k$ and $s_0 \models \phi_1 \wedge \ldots \wedge s_{l-1} \models \phi_1$ and $s_l \models \phi_2$ and the concatenation of the actions $a_1 \ldots a_l$ is a word in the language defined by the regular expression ρ. In the case $k = \infty$ there is no upper limit on the length l of the path. It is straight-forward to show that standard temporal operators such as "next" and "until" can be derived as special cases of the quantised path formula $P_{\bowtie p}(.)$. Being able to characterise paths by regular expressions, $\mathcal{L}$ is more expressive than PCTL.

Example 4 *We consider again the bus arbiter (system model $\mathcal{F}$ from Fig. 1b) in the environment as given by Markov chain $\mathcal{M}$ from Fig. 1c. Using the temporal logic $\mathcal{L}$, the following requirements can be formulated (note that for requirement specification the user does not need to have explicit knowledge of the product $\mathcal{P}$):*

1. *The arbiter should be safe, i.e. the two grant signals should never be active at the same time: $\phi_1 = \neg(gnt_1 \wedge gnt_2)$*
   Requirement ϕ_1 is an example of a simple propositional logic formula.

2. *The bus utilisation should be at least 50%: $\phi_2 = S_{\geq 0.5}(gnt_1 \vee gnt_2)$*
   Requirement ϕ_2 states that, in equilibrium, one of the two grant signals should be active at least 50% of the time.

3. *If the req_2 input signal persists, cell 2 should be granted the bus after at most 4 clock cycles: $\phi_3 = P_{\leq 0}\left(\neg gnt_2[(01+11)^*]^4 \neg gnt_2\right)$*
   Requirement ϕ_3 is a quantised path formula, where the probability bound ≤ 0 expresses that the formula should hold with probability 0, i.e. not on any path emanating from a state. Persistence of the req_2 signal is formulated by the regular expression $(01+11)^*$ which denotes a sequence of actions (i.e. inputs) of arbitrary length where the second element, corresponding to req_2, is always set to one. Requirement ϕ_3 expresses a fairness property of the arbiter.

4. *At most 50% of the bus holding times of cell 2 should last for 2 clock cycles: $\phi_4 = gnt_2 \rightarrow P_{\leq 0.5}\left(true[any]^1 gnt_2\right)$*
   Requirement ϕ_4 is formulated as an implication. Furthermore, the abbreviation "any" is used to denote an arbitrary action (input). Looking at the environment model from Fig. 1c, one can easily see that requirement ϕ_4 will be violated, since having issued the req_2 signal, cell 2 will reissue this signal with probability 0.8.

5. *The probability of two successive bus grants to cell 1 without granting to cell 2 in between should be at most 20%: $\phi_5 = gnt_1 \rightarrow P_{\leq 0.2}\left(\neg gnt_2[any; any; any^*]^\infty gnt_1\right)$*
   Requirement ϕ_5 states that if the bus is currently granted to cell 1, a new grant to cell 1 without intermediate grant to cell 2 should happen with probability at most 20%. The number of steps between the current and the next grant to cell 1 is left open, but the regular expression enforces that there are at least two steps (with arbitrary input) in between.

[3]The formal semantics of $\mathcal{L}$ is omitted in this paper due to space limitations.

Model Checking Algorithm: The basic model checking algorithm for the logic $\mathcal{L}$ is similar to that for CTL: A given $\mathcal{L}$-formula is checked by checking all of its subformulas in a bottom-up fashion, thereby labelling the states of the model with the valid subformulas which can thereafter be treated like atomic propositions. Checking the Boolean operators of $\mathcal{L}$ is similar to CTL. For checking the steady-state operator $S_{\bowtie p}(\phi)$ two cases have to be distinguished: If the combined model $\mathcal{P}$ consists of a single strongly connected component the steady-state probabilities are independent of the initial state and can be determined by solving the linear system of equations $\vec{\pi} \cdot T = \vec{\pi}$, where T denotes the transition probability matrix of $\mathcal{P}$ and $\vec{\pi}$ denotes the vector of steady-state probabilities. Otherwise the computation of the initial-state-dependent steady-state distribution requires the computation of the bottom strongly connected components (BSCC) of $\mathcal{P}$ by a graph algorithm and then determining the steady-state distribution for each BSCC. Furthermore, for each transient state as initial state, the probabilities of eventually reaching the individual BSCCs have to be computed.

The algorithm for model checking quantised path formulas of the form $P_{\bowtie p}(\phi_1[\rho]^k\phi_2)$ is the most involved. It proceeds along the following steps:

1. From the regular expression ρ a non-deterministic finite automaton $\mathcal{A}_\rho$ is constructed.
2. From the labelled Markov chain $\mathcal{P}$ and the automaton $\mathcal{A}_\rho$ a product is constructed in a way that is analogous (but not identical) to the way described in [2]. The result is an unlabelled DTMC $\mathcal{X}$ whose transitions are of the form

$$(s_i, Z) \xrightarrow{p} (s_j, Z') \tag{3}$$

   where s_i and s_j are states of $\mathcal{P}$, and Z and Z' are sets of states of $\mathcal{A}_\rho$. Furthermore, $\mathcal{X}$ has two special absorbing[4] states: All states (s_i, Z) where the model part satisfies ϕ_2 and where Z includes an accepting state are combined to a single absorbing state *succ*, while all states where the model part satisfies neither ϕ_1 nor ϕ_2 are combined to a single absorbing state *fail*.
3. By standard DTMC analysis of $\mathcal{X}$, for each state the probability $\tilde{p}$ of reaching state *succ* in at most k steps is computed. If $\tilde{p} \bowtie p$ then the quantised path formula is valid, otherwise it is violated.

When model checking formulas of type $S_{\bowtie p}(.)$ or $P_{\bowtie p}(.)$, in addition to returning "yes" or "no" as an answer, the model checker can also return the computed numerical results, such that the user learns how close these values are to the specified probability bound p and thus better understands the results of verification.

The method described in this paper involves a double product construction with the obvious negative implication on the size of the state space. The model $\mathcal{P}$ to be analysed is the product of the FSM $\mathcal{F}$ and the Markovian environment $\mathcal{M}$. The size of its state space is at the most $|S_\mathcal{F}| \cdot |S_\mathcal{M}|$, but in many cases it is much smaller due to reachability conditions. When model checking a formula of type $P_{\bowtie p}(\phi_1[\rho]^k\phi_2)$ a further product, namely that of $\mathcal{P}$ and automaton $\mathcal{A}_\rho$, has to be constructed, which also includes the determinisation of $\mathcal{A}_\rho$. However, here again, experience has shown that the size of the reachable portion of the resulting state space is often far smaller than the theoretical worst case product. The reason is that the regular expression ρ of inputs, together with the conditions ϕ_1 and ϕ_2, may considerably restrict the permissible behaviour of $\mathcal{P}$.

Implementation with BDDs: The model checking algorithm for the logic $\mathcal{L}$ can be implemented in a fully symbolic manner. As described in Sec. 2, the model to be checked is

[4] "Absorbing" means having a self-loop with transition probability one.

represented as an MTBDD P. For each subformula ϕ_i to be checked, a BDD is constructed which represents the corresponding satisfaction set $Sat(\phi_i)$. Propositional logic subformulas are checked by directly applying the Boolean operators on the respective BDDs. For checking formulas of type $S_{\bowtie p}(.)$, the vector of steady-state probabilities is computed with the help of an MTBDD-based iterative numerical algorithm as realised in the tools PRISM and CASPA.

For checking formulas of type $P_{\bowtie p}(\phi_1[\rho]^k\phi_2)$ the automaton $\mathcal{A}_\rho$ is represented symbolically as a BDD A_ρ (which is constructed directly from the regular expression ρ). The next step is the symbolic product construction of P and A_ρ, which includes the determinisation of the automaton $\mathcal{A}_\rho$ by a standard powerset construction and thereby also the generation of the special absorbing states *succ* and $fail$. This step yields the transitions of the resulting DTMC $\mathcal{X}$ (according to Eq. (3)) encoded as an MTBDD X on which the calculation of the k-step transient probabilities is carried out by an efficient MTBDD-based numerical method.

4 Conclusion and Future Work

This paper has introduced a method for verifying finite state machines in a probabilistic environment which is specified by a Markov chain. Requirements are formulated with the help of an expressive temporal logic and checked by a specialised model checking algorithm. The paper also briefly describes how the method can be realised symbolically, using BDDs and MTBDDs as the basic data structures. Based on experience reports from related fields such as [4, 8, 9] it is fair to say that the approach has the potential to handle very large systems.

The next step is an implementation of the method and to carry out case studies based on real-life FSMs. In addition, there exist ideas of how the expressiveness of the logic $\mathcal{L}$ could be further extended, which would lead to extensions of the model checking algorithm.

References

[1] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic Decision Diagrams and their Applications. *Form. Meth. in Sys. Design*, 10(2/3):171–206, 1997.

[2] C. Baier, L. Cloth, B. Haverkort, M. Kuntz, and M. Siegle. Model Checking Action- and State-labelled Markov Chains. In *Int. Conf. on Dependable Systems and Networks: Performance and Dependability Symposium*, pages 701–710. IEEE Computer Society Press, 2004.

[3] M. Fujita, P. McGeer, and J.C.-Y. Yang. Multi-terminal Binary Decision Diagrams: An efficient data structure for matrix representation. *Form. Meth. in Sys. Design*, 10(2/3):149–169, 1997.

[4] G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Markovian Analysis of Large Finite State Machines. *IEEE Trans. on CAD*, 15(12):1479–1493, Dec. 1996.

[5] H. Hansson and B. Jonsson. A logic for reasoning about time and probability. *Formal Aspects of Computing*, (6):512–535, 1994.

[6] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A tool for model checking Markov chains. *Software Tools for Technology Transfer (STTT)*, 4(2):153–172, 2003.

[7] H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, and M. Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *Journal of Logic and Algebraic Programming*, 56(1-2):23–67, 2003.

[8] M. Kuntz, M. Siegle, and E. Werner. Symbolic Performance and Dependability Evaluation with the Tool CASPA. In *Europ. Perf. Engineering Workshop*, pages 293–307. LNCS 3236, 2004.

[9] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking in practice: Case studies with PRISM. *ACM Performance Evaluation Review*, 32(4):16–21, 2005.

[10] D. Marculescu, R. Marculescu, and M. Pedram. Trace-Driven Steady-State Probability Estimation in FSMs with Application to Power Estimation. In *Proc. Design, Automation and Test in Europe*, pages 774–781, IEEE, 1998.