Title:	**A Modest Approach to Modelling and Checking Markov Automata**
Authors:	Yuliya Butkova, Arnd Hartmanns, Holger Hermanns
Report Number:	2019-11
ERC Project:	Power to the People. Verified.
ERC Project ID:	695614
Funded Under:	H2020-EU.1.1. – EXCELLENT SCIENCE
Host Institution:	Universität des Saarlandes, Dependable Systems and Software Saarland Informatics Campus
Published In:	QEST 2019

A Modest Approach to Modelling and Checking Markov Automata*

Yuliya Butkova[1], Arnd Hartmanns[2], and Holger Hermanns[1,3]

[1] Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
[2] University of Twente, Enschede, The Netherlands
[3] Institute of Intelligent Software, Guangzhou, China

Abstract. Markov automata are a compositional modelling formalism with continuous stochastic time, discrete probabilities, and nondeterministic choices. In this paper, we present extensions to the Modest language and the mcsta model checker to describe and analyse Markov automata models. Modest is an expressive high-level language with roots in process algebra that allows large models to be specified in a succinct, modular way. We explain its use for Markov automata and illustrate the advantages over alternative languages. The verification of Markov automata models requires dedicated algorithms for time-bounded probabilistic reachability and long-run average rewards. We describe several recently developed such algorithms as implemented in mcsta and evaluate them on a comprehensive set of benchmarks. Our evaluation shows that mcsta improves the performance and scalability of Markov automata model checking compared to earlier and alternative tools.

1 Introduction

Studying dependability and performance aspects of critical designs or implementations [4] requires a formal mathematical model that captures the core quantitative aspects of such systems. In particular, we need *stochastic continuous time* to model delays of which we only know averages, e.g. the mean time to failure, *discrete probabilistic choices* to describe instantaneous uncertain decisions, as in e.g. randomised algorithms, and *nondeterminism* to be able to deal with underspecification, abstraction, unquantified uncertainty, and concurrency. Markov automata (MA, [18, 20]) extend the classical formalisms of continuous-time Markov chains and discrete-time Markov decision processes (MDP) to encompass all three of these aspects. In contrast to continuous-time MDP (CTMDP), they are compositional: there is a natural parallel composition operator for networks of MA that provides for both interleaved and synchronising transitions without the need for ad-hoc operations to combine transition rates.

MA are the semantic basis for generalised stochastic Petri nets [19] and dynamic extensions of fault trees [6, 31]. Several publications studied algorithmic

* Authors are listed alphabetically. This work has received financial support by DFG grant 389792660 as part of TRR 248 (see perspicuous-computing.science) by ERC Advanced Grant 69561 (POWVER), and by NWO VENI grant 639.021.754.

problems related to the efficient analysis of MA [2, 12–15, 23, 24, 30]. In this light, it is disappointing that tool support for MA is thus far rather brittle. The one dedicated tool for compositional modelling with MA, Scoop [38], is unmaintained, as is the corresponding lower-level MA model checker Imca [22]. The one other actively developed tool with comprehensive MA support is Storm [17], which however lacks built-in support for high-level compositional modelling.

Using the mathematical formalism of MA directly to build complex models is cumbersome. For their use to be practical, we need a higher-level *modelling language*. Aside from a parallel composition operator, such languages typically provide variables over finite domains that can be used in expressions to e.g. enable or disable transitions. Their semantics is then an MA whose states are the valuations of the variables, allowing to compactly describe very large MA. In this paper, we present recent extensions to Modest [27], a high-level modelling language for stochastic hybrid systems, that add support for expressing MA models. Rooted in process algebra, Modest provides various composition operators that allow large models to be assembled from small, easy-to-understand components. In Sect. 3, we illustrate the use of Modest for MA, and we compare its succinctness, expressivity, and readability with alternative languages.

We build MA models to compute quantitative properties of systems such as safety (the probability to reach an unsafe state), reliability (doing so within a time bound), or throughput (the long-run average amount of work completed per time unit). Probabilistic model checking techniques [3] can be applied to MA to effectively compute or approximate such values. While the computation of *unbounded* reachability probabilities and expected accumulated rewards can be reduced to checking the MA's embedded MDP, *time-bounded* probabilities and *long-run average* rewards require dedicated algorithms. We summarise the currently available algorithms, their particular characteristics, and notable implementation considerations, in Sect. 4. To complement our extension of the Modest language with suitable analysis facilities, we have implemented the most promising of these algorithms in the mcsta model checker of the Modest Toolset [28]. We use the MA models of the Quantitative Verification Benchmark Set [29] to evaluate the performance of our implementation and of the different algorithms in Sect. 5. We compare the results with Imca and Storm.

2 Markov Automata

The mathematical formalism of Markov automata provides nondeterministic choices as in labelled transition systems (LTS, or Kripke structures or finite automata), discrete probabilistic decisions as in discrete-time Markov chains (DTMC), and states with exponentially distributed residence times as in continuous-time Markov chains (CTMC). The relationships between these formalisms are visualised in Fig. 1. We now define MA formally and describe their semantics.

Preliminaries. We write $\{ x_1 \mapsto y_1, \dots \}$ to denote the function that maps all x_i to y_i, and if necessary in the respective context, implicitly maps to 0 all x

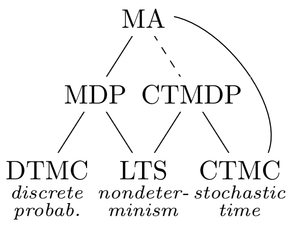


Fig. 1. The MA family tree

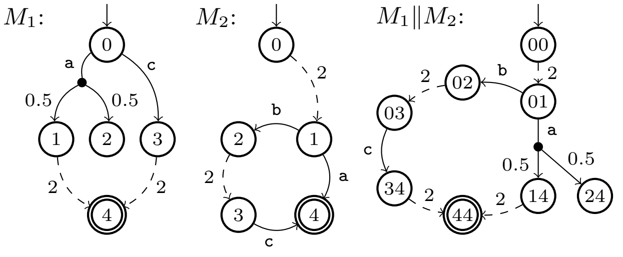


Fig. 2. Example Markov automata

for which no explicit mapping is specified. Given a set S, its powerset is 2^S. A (discrete) probability distribution over S is a function $\mu \in S \to [0,1]$ such that $spt(\mu) \overset{\text{def}}{=} \{\, s \in S \mid \mu(s) > 0 \,\}$ is countable and $\sum_{s \in spt(\mu)} \mu(s) = 1$. $Dist(S)$ is the set of all probability distributions over S, and $\mu_1 \otimes \mu_2$ is the product distribution of μ_1 and μ_2 defined by $(\mu_1 \otimes \mu_2)(s) = \mu_1(s) \cdot \mu_s(s)$. We refer to discrete random choices as *probabilistic* and to continuous ones as *stochastic*.

Definition 1. *A* Markov automaton *(MA) is a tuple*

$$M = \langle S, s_0, A, P, Q, rr, br \rangle$$

where

- S *is a finite set of* states*, with* $s_0 \in S$ *being the* initial state*,*
- A *is a finite set of* actions*,*
- $P \in S \to 2^{A \times Dist(S)}$ *is the* probabilistic transition *function,*
- $Q \in S \to 2^{\mathbb{Q} \times S}$ *the* Markovian transition *function,*
- $rr \in S \to [0, \infty)$ *is the* rate reward *function, and*
- $br \in S \times Tr(M) \times S \to [0, \infty)$ *is the* branch reward *function*

with $Tr(M) \overset{\text{def}}{=} \bigcup_{s \in S} P(s) \cup \bigcup_{s \in S} Q(s)$*.* $P(s)$ *and* $Q(s)$ *must be finite sets for all* $s \in S$*. We define the* exit rate *of* $s \in S$ *as* $E(s) = \sum_{\langle \lambda, s' \rangle \in Q(s)} \lambda$*.*

Example 1. Fig. 2 shows two MA M_1 and M_2 without rewards. We draw probabilistic transitions as solid, Markovian ones as dashed lines. If a transition leads to a single target state, we omit the intermediate probabilistic branching node.

The semantics of an MA is that, in state s, (1) the probability to take Markovian transition $\langle \lambda, s' \rangle \in Q(s)$ and move to state s' within t time units is

$$\lambda / E(s) \cdot (1 - \mathrm{e}^{-E(s) \cdot t}),$$

i.e. the residence time follows the exponential distribution with rate $E(s)$ and the choice of transition is weighted by their rates; and (2) at any point in time, a probabilistic edge $\langle a, \mu \rangle \in P(s)$ can be taken with the successor state being chosen according to μ. MA thus separate interaction from timing: the former is represented by the action-labelled probabilistic transitions, and the latter is governed by the rates of the Markovian transitions. This is the key difference to CTMDP, which have one kind of transitions with both actions and rates. It enables parallel composition operators with action synchronisation for MA without any need to prescribe an ad-hoc operation for combining rates.

Definition 2. *Given two MA $M_i = \langle S_i, s_{0_i}, A_i, P_i, M_i \rangle$, $i \in \{ 1, 2 \}$, their parallel composition is $M_1 \parallel M_2 \overset{\text{def}}{=} \langle S_1 \times S_2, \langle s_{0_1}, s_{0_2} \rangle, A_1 \cup A_2, P, M \rangle$ with P the smallest function such that*

$$\begin{aligned}
&(\langle a, \mu \rangle \in P_1(s_1) \land a \notin A_2 \Rightarrow \langle a, \mu \otimes \{ s_2 \mapsto 1 \} \rangle \in P(\langle s_1, s_2 \rangle)) \\
\land\, &(\langle a, \mu \rangle \in P_2(s_2) \land a \notin A_1 \Rightarrow \langle a, \{ s_1 \mapsto 1 \} \otimes \mu \rangle \in P(\langle s_1, s_2 \rangle)) \\
\land\, &(\langle a, \mu_1 \rangle \in P_1(s_1) \land \langle a, \mu_2 \rangle \in P_2(s_2) \land a \in A_1 \cap A_2 \Rightarrow \langle a, \mu_1 \otimes \mu_2 \rangle \in P(\langle s_1, s_2 \rangle))
\end{aligned}$$

and Q is the smallest function s.t. $(\langle \lambda, s_1' \rangle \in Q_1(s_1) \Rightarrow \langle \lambda, \langle s_1', s_2 \rangle \rangle \in Q(\langle s_1, s_2 \rangle))$ and vice-versa for Q_2.

The operator above uses multi-way synchronisation on the shared alphabet of the two automata; similar operators could be defined for other synchronisation mechanisms, e.g. to define input-output MA. Fig. 2 includes the parallel composition of the example M_1 and M_2, where we write nm for state $\langle n, m \rangle$. The two automata synchronise on the shared actions $\mathtt{a}$ and $\mathtt{c}$.

We defined MA as *open* systems [8]: probabilistic transitions can interact with, wait for, and be blocked by other MA in parallel composition. For verification, we make the usual *closed system* and *maximal progress* assumptions, i.e. we assume that probabilistic transitions face no further interference and take place without delay. If multiple probabilistic transitions are available in a state, however, the choice between them remains nondeterministic. Since the probability that a Markovian transition is taken in zero time is 0, the maximal progress assumption allows us to remove all Markovian transitions from states that also have a probabilistic transition. In such *closed MA*, we can thus distinguish between Markovian states (where $P(s) = \varnothing$) and probabilistic states (where $Q(s) = \varnothing$). The behaviour of a closed, deadlock-free MA M is defined via its paths:

Definition 3. *A* path $\pi \in \mathit{\Pi}(M)$ *is an infinite sequence*

$$\pi = s_0\, t_0\, tr_0\, s_1 \ldots \in (S \times [0, \infty) \times Tr(M))^\omega$$

such that $Q(s_i) = \varnothing \Rightarrow t_i = 0$ and $tr_i \in P(s_i) \cup Q(s_i)$. We write $\mathit{\Pi}_f(M)$ for the set of all path prefixes π_f *ending in a state. Let $\pi_{\leq i} \overset{\text{def}}{=} s_0\, t_0 \ldots s_i$. The* duration $\mathrm{dur}(\pi_f)$ *of a path prefix is the sum of its residence times t_i. A path's* reward *is*

$$\mathrm{rew}(\pi) = \textstyle\sum_{i=0}^{\infty} t_i \cdot rr(s_i) + br(s_i, tr_i, s_{i+1})$$

and is analogously defined for prefixes.

A path prescribes a resolution of all nondeterministic, probabilistic, and stochastic choices. To define a probability measure, we resolve nondeterminism only:

Definition 4. *Given an MA M as above, a* scheduler *in $\mathfrak{S}(M)$ is a function $\sigma \in \mathit{\Pi}_f(M) \to Tr(M)$ s.t. $\forall s \in S \colon \sigma(s) = tr \Rightarrow tr \in P(s) \cup Q(s)$. A* time-dependent *scheduler is in $S \times [0, \infty) \to Tr(M)$; a* memoryless *one in $S \to Tr(M)$.*

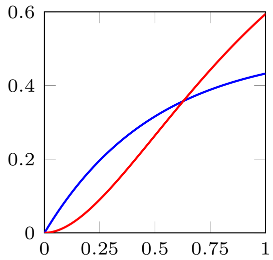
We define deterministic schedulers only since randomised schedulers are in practice only needed for multi-objective problems [34]. We note that CTMDP with early schedulers [36] can be encoded as closed MA. A scheduler induces a probability measure over sets of measurable paths in the usual way [30]. For all of the following types of properties, we are interested in the maximum (supremum) and minimum (infimum) values when ranging over all schedulers $\sigma \in \mathfrak{S}(M)$:

Reachability probabilities: Given a set of goal states $G \subseteq S$, compute the probability of the set of paths that include a state in G. Memoryless schedulers suffice to achieve optimal results (i.e. the max. and min. probabilities).

Time-bounded reachability: Additionally restrict to paths where the sum of delays up to reaching the first state in G is below a bound $b \in [0, \infty)$. Here, time-dependent schedulers with input $b - \mathrm{dur}(\pi_f)$ suffice.

Expected accumulated rewards: For $G \subseteq S$, compute the expected value of the random variable[4] that assigns to path π the value $\mathrm{rew}(\pi_f)$ where π_f is the shortest prefix of π with a state in G. Memoryless schedulers suffice.

Long-run average rewards: Compute the expected value of the random variable that assigns to path π the value $\lim_{i \to \infty} \mathrm{rew}(\pi_{\leq i})/\mathrm{dur}(\pi_{\leq i})$. Memoryless schedulers suffice.



Example 2. Consider MA $M_1 \parallel M_2$ of Fig. 2 and the probability to reach state $\langle 4, 4 \rangle$ within 1 time unit. In state $\langle 0, 1 \rangle$, we have to decide whether to choose action `a` or `b`. The optimal decision depends on the amount of time t that has passed in state $\langle 0, 0 \rangle$. In the plot on the right, we show the probability of reaching state $\langle 4, 4 \rangle$ (y-axis) depending on $1-t$ (x-axis). The blue line represents the reachability probability for the memoryless scheduler that always chooses `a` and the red one is for the scheduler that always takes action `b`. A time-dependent scheduler can make better decisions than either of these two by determining the values of t for which `a` results in a higher probability than `b` and vice-versa. The optimal scheduler thus chooses `a` if and only if $1 - t \leq 0.63$ approximately.

3 Modelling

Tools for the automated analysis of MA need a syntax in which the model and the properties of interest are specified. As noted in Sect. 1, such a modelling language needs to provide a parallel composition operator such that large MA can be built from small specifications, and will typically support modelling with variables that can be used in guards and assignments. In the context of such symbolic formalisms, we have *locations* and *edges* that each induce (many) states and transitions, respectively, in the formalism's plain-MA semantics.

3.1 Modest for Markov Automata

As part of implementing the Jani [10] model exchange format, we recently introduced support for MA into the syntax and semantics of the Modest modelling language [27]. Modest previously supported MDP and more complex

[4] This is well-defined if the maximum (minimum) probability to reach G is 1; otherwise, we define the minimum (maximum) expected accumulated reward to be ∞.

```
const real B;
int(0..2) succ = 0;
action a, b, c;
property P_Min = Pmin(<>[T<=B] (succ == 2));
property P_Max = Pmax(<>[T<=B] (succ == 2));
process M1()
{
   bool fail = false;
   alt {
   :: a palt {
      :1: {==}
      :1: {= fail = true =}
      }
   :: c
   };
   when(!fail) rate(2) {= succ++ =}
}
process M2()
{
   rate(2) tau;
   alt {
   :: a {= succ++ =}
   :: b; rate(2) tau; c {= succ++ =}
   }
}
par {
:: M1()
:: M2()
}
```

Fig. 3. Modest for MA

```
global succ:{0..2} = 0
DONE = done.DONE[]
M1  = a.psum(0.5 -> M1a[] ++ 0.5 -> DONE[])
      ++ c.M1a[]
M1a = <2>.setGlobal(succ, succ + 1)
      .DONE[]
M2  = <2>.(a.M2a[] ++ b.<2>.c.M2a[])
M2a = setGlobal(succ, succ + 1).DONE[]
init M1[] || M2[]
comm (a, a, a), (c, c, c)
reachCondition (succ = 2)
```

Fig. 4. MAPA process algebra

```
ma
const double B
module M1
  s1: [0..4];
  [a] s1=0 -> 0.5:(s1'=1) + 0.5:(s1'=2);
  [c] s1=0 -> 1:(s1'=3);
  <>  s1=1 | s1=3 -> 2:(s1'=4);
endmodule
module M2
  s2: [0..4];
  <>  s2=0 -> 2:(s2'=1);
  [a] s2=1 -> 1:(s2'=4);
  [b] s2=1 -> 1:(s2'=2);
  <>  s2=2 -> 2:(s2'=3);
  [c] s2=3 -> 1:(s2'=4);
endmodule
"P_Min": Pmin=? [F<=B (s1=4 & s2=4)];
"P_Max": Pmax=? [F<=B (s1=4 & s2=4)];
```

Fig. 5. Prism dialect supporting MA

```
#INITIALS
s00
#GOALS
s44
#TRANSITIONS
s00 !
* s01 2
s01 a
* s02 1
s01 b
* s14 0.5
* s24 0.5
s14 !
* s44 2
s02 !
* s03 2
s03 c
* s34 1
s34 !
* s44 2
```

Fig. 6. Imca state space format

continuous-time formalisms such as stochastic hybrid automata, but did not have provisions for succinctly annotating edges with rates. We added the `rate`(e) construct for this purpose, which behaves analogously to the existing `when`(e) construct for specifying the enabling condition of an edge. Modest enforces the separation of probabilistic and Markovian transitions by requiring edges for which a rate is specified to have the predefined and non-synchronising τ action label. If this restriction is not met, the model is recognised as a CTMDP.

At its core, Modest is a process algebra: it provides various operations such as parallel composition (`par`), sequential composition (`;`), parameterised process definitions, process calls, and guards (`when`) to flexibly construct complex models out of small and reusable components. Its syntax however borrows heavily from commonly used programming languages, and it provides high-level conveniences such as `do` loops and a full-fledged mechanism for throwing (`throw`) and handling (`try-catch`) exceptions. As such, Modest tends to be more verbose than classic process algebras, but also more readable and beginner-friendly. To specify complex behaviour in a succinct manner, Modest also provides variables of standard basic types (e.g. `bool`, `int`, or bounded `int`), arrays, and user-defined recursive datatypes akin to functional programming languages. Its syntax for expressions again is aligned with `C`-like programming languages for ease of use.

In Fig. 3, we show a Modest representation of the parallel composition of MA M_1 and M_2 of Fig. 2. M_1 has been slightly optimised by merging states 1 and 3 into the last line of process `M1`; this actually came naturally when modelling due to the ease in which behaviours can be combined and shared in Modest. The model also includes the declaration of two properties of interest for verification, `P_Min` and `P_Max`, which ask for the probability to reach state $\langle 4, 4\rangle$—made observable via the global variable `succ`—within time `B` akin to Example 2. `B` is an open parameter for which values can be specified at verification time. There are many features of Modest not used in this small model; the interested reader may find more complex Modest MA models, in particular with arrays and rewards, in the Quantitative Verification Benchmark Set [29] at qcomp.org.

Tool support. The Modest Toolset [28] is a comprehensive suite of tools for quantitative modelling and verification. Its primary input languages are Modest and Jani. MA are supported in its mosta, moconv[5], mcsta, and modes tools. mosta visualises the symbolic semantics of models and is useful for model debugging. moconv transforms models between modelling languages (it can e.g. convert Modest to Jani) and performs syntactic rewriting and optimisations. mcsta is an explicit-state model checker; we present and evaluate its MA-specific algorithms in sects. 4 and 5. modes [9] is a statistical model checker with automated rare event simulation capabilities. It implements the lightweight scheduler sampling approach [32] for nondeterministic models, including MA [16]. The Modest Toolset is written in `C#`, works cross-platform on Linux, Mac OS, and Windows, and is freely available at modestchecker.net. All its tools share a common infrastructure for parsing and syntactic transformations. mcsta and modes additionally build on the same state space exploration engine that compiles models to bytecode at runtime for memory efficiency and performance.

3.2 Alternative Modelling Languages

Modest is not the only modelling language for MA. These are the alternatives:

State space files for Imca. The first MA-specific algorithms were implemented in the Imca tool [22]. Its only input language is a text-based explicit state space format as illustrated for our example of $M_1 \parallel M_2$ in Fig. 6. This is clearly not a useful modelling language, but a format to be automatically generated by tools.

[5] moconv can also export CTMDP to Jani, but due to their lack of a natural parallel composition operator, the analysis of CTMDP is not supported in the other tools.

Guarded commands with Storm*.* Of the alternative tools for MA, Storm [17] is the only one that is actively maintained. It provides many input languages, with MA being supported through a state space format similar to Imca's, via Jani, as the semantics of generalised stochastic Petri nets [19] in GreatSPN format [1], and through an extension of the Prism guarded command language. We show our example in the latter in Fig. 5. It is a very simple, small language that is easy to learn, however it completely lacks higher-level constructs to structure and compose models aside from the implicit parallel composition of its *modules*.

Process algebra with Scoop*.* Mapa [38] is a dedicated process algebra for MA. It is supported by Scoop [38], which can linearise, reduce, and finally export Mapa models to Imca for verification. We show the example of M_1 and M_2 in Mapa in Fig. 4. As a classic concise process algebra, Mapa tends to be very succinct, but also difficult to read. Mapa models can be much more flexibly composed than Prism models, yet there is less syntactic structure than in Modest—although the languages conceptually share many operators. Mapa notably has a predefined *queue* datatype, and users can specify custom non-recursive datatypes.

Jani [10] is a model interchange format designed to ease tool development and interoperation. It is Json-based and thus human-debuggable, but not intended as human-writable. It represents networks of automata with variables symbolically. Since both the Modest Toolset and Storm support Jani, it is possible to e.g. build MA models in the Modest language, export them to Jani with moconv, and then verify them with Storm. Likewise in the other direction, we can e.g. create a Petri net with GreatSPN, convert to Jani with Storm, and analyse it with mcsta or modes. In this way, the most appropriate modelling language can be combined with the best analysis method and tool for every specific scenario.

4 Algorithms

While the values for some classes of properties can be computed by checking the embedded MDP of an MA, most need dedicated MA-specific algorithms. We briefly describe the algorithms implemented for MA in mcsta, Storm and Imca.

4.1 Untimed and Expected-Reward Properties

Like for CTMC, properties that do not refer to time, or that only refer to expected times, can be computed on the embedded MDP of the Markov automaton. These properties include unbounded as well as branch reward-bounded reachability probabilities and expected accumulated rewards. For simplicity, we will refer to all of these as "unbounded properties". The available algorithms include all the standard exhaustive model checking algorithms for MDP [33], in particular using linear programming (LP), policy iteration, value iteration, interval iteration [5, 25], and sound value iteration [35]. Standard "unsound" value iteration and typical LP solvers do not provide any guarantees (such as ϵ-closeness to the true probability or value) on their results, while interval iteration and sound value iteration do. To combat the state space explosion problem of the exhaustive methods, the BRTDP learning-based approach [7] can be used for probabilities. It attempts to explore only a small part of the state space that is sufficient to provide a lower and an upper bound on the result that are close enough. Its efficiency both in terms of runtime and in terms of memory reduction highly depend on the structure of the model, though.

Tool support. mcsta implements value iteration, LP, and interval iteration for expected rewards and unbounded reachability probabilities. It is being extended to support sound value iteration. It also provides BRTDP as in [2] where simulations with the uniform probabilistic scheduler are used to explore a part of the state space. After every batch of simulation runs, interval iteration is used to compute bounds. STORM implements value and policy iteration, LP, interval iteration, sound value iteration, and a variant of BRTDP. It also provides algorithms to compute exact (rational) solutions using exact arithmetic, but they are currently limited to small models. IMCA supports value iteration only.

4.2 Time-Bounded Reachability

Time-bounded properties pose one of the most challenging problems in MA model checking. Several algorithms with rather different characteristics are currently available for approximating time-bounded reachability probabilities: The **discretisation** approach [23] discretises the time horizon into small intervals, such that the MA will likely perform at most one Markovian transition within each interval. **Unif+** was first presented for CTMDP [13] and later extended to MA [21] in the straightforward way. It is based on an approximation of the optimal time-bounded reachability probability over timed schedulers with that same value but ranging over untimed schedulers. The **switch-step** algorithm [12] attempts to compute *switching points*: the points at which the optimal scheduler changes the action for at least one state, as illustrated in Example 2. Finally, the **BRTDP** idea for time-bounded reachability properties on CTMDP [2] can be extended to MA straightforwardly: the simulation phase performs CTMC-style simulation for Markovian states and MDP-style simulation over probabilistic states. Time progresses only over Markovian states and the simulation stops whenever the time bound expires or a target state is reached. Resolution of nondeterminism is performed via the randomised scheduler that samples the next action uniformly at random from the enabled actions. The analysis phase can be performed by any of the other algorithms for time-bounded analysis on MA.

Tool support. mcsta implements Unif+ and switch-step while STORM supports Unif+ and the discretisation approach. Both provide sound implementations of these algorithms (i.e. they guarantee ϵ-correct results). IMCA implements only discretisation and uses unsound techniques for certain subproblems.

4.3 Long-Run Average Rewards

There exist two approaches for computing long-run average rewards: one based on a reduction to a linear program [24], and a value iteration-based algorithm [14] that approximates the reward up to a user-specified (and guaranteed) precision. In both cases, first the long-run average reward is determined for each maximal end component, then the end components are collapsed, and the overall result is computed as an expected reward value on the collapsed state space.

Tool support. mcsta and Storm implement both of the algorithms while Imca implements only the linear programming-based approach.

4.4 Other Verification Problems

We now briefly summarise other MA verification problems, name the corresponding available algorithms, and mention where they are implemented.

Time-bounded expected rewards extend the time-bounded reachability problem to rewards. The property represents the expected accumulated reward until a time bound is reached. Algorithmic support for this property is limited to the discretisation-based approach of [24], which is implemented in Imca.

Resource-bounded rewards generalise both time-bounded reachability and time-bounded expected rewards. A resource-bounded reward property represents the expected accumulated reward within a finite resource budget. The resource is formally represented by a second type of (branch or rate) reward in the model. The only algorithm available to date is presented in [30], with no tool support.

Discounted rewards. Expected discounted reward properties ask for the expected total reward where rewards collected at a certain time point are discounted with a value, depending on this time point. For example, when dealing with income, discounted rewards allow to take inflation into account. Iterative algorithms for computing and approximating the value exist, such as policy and value iteration [15]. There is however no tool support so far.

Multi-objective tradeoffs. Multi-objective MA model checking allows finding a scheduler that is optimal for several objectives, rather than only one. The only algorithm available to date and implemented in Storm is presented in [34]. It does not support the full range of properties, in particular excluding long-run average and discounted rewards. For the underlying time-bounded analysis, it resorts to discretisation, which tends to not scale well (see Sect. 5 below).

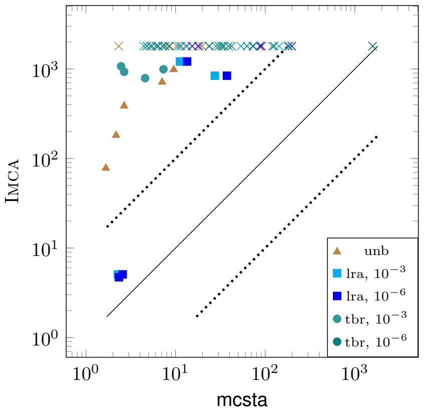
5 Experiments

The Quantitative Verification Benchmark Set (QVBS, [29]) currently contains 18 MA models, specified in Modest, Storm's extension of the Prism language for

MA (cf. Sect. 3.2), as GreatSPN Petri nets, and as fault trees in the Galileo format [37]. For every model, there is also a Jani version. All models have open parameters (like B in our Modest example of Fig. 3) to be scaled up from small to huge state spaces. We use most of these models, selecting parameters that make for challenging, but not impossible, state space sizes (up to a few millions of states), to compare the performance and scalability of the algorithms implemented in mcsta with Imca and Storm. The models include variations of queueing systems, dependability models, scheduling problems, and security case studies. We excluded those models that only have spurious non-determinism (i.e. they are equivalent to a CTMC), and those that can be fully checked in just a few seconds for the given parameter valuations. Due to the absence of long-run average reward properties in most MA models of the benchmark set, we added sensible long-run average properties to most of the Modest models (which are easy to modify by hand, in contrast to Jani) in order to be able to do a meaningful performance comparison. Those are mainly steady-state probabilities (i.e. the special case of a rate reward of 1 in some states and of 0 in all others), or properties describing long-run average costs of running the modelled system.

All experiments were conducted on two servers with Intel Core i7-4790 processors and 16 resp. 32 GB of RAM running 64-bit Ubuntu Linux 18.04. We keep the default values for all the command line arguments of the tools, unless we explicitly mention specific parameters being used. When we request a certain precision for results (with sound methods), we request absolute, not relative, precision. We show all results as scatter plots like the one below, with log-log axes. Every benchmark *instance*—a model, a valuation for its parameters, and a property to check—results in one point in these plots. A point $\langle x, y \rangle$ states that the runtime of the tool noted on the x-axis on one instance was x seconds while the runtime of the tool noted on the y-axis was y seconds. Thus points above the solid diagonal line indicate instances where the x-tool was the fastest; it was more than ten times faster (slower) on points above (below) the dotted line. We set the timeout to 30 minutes; a timeout is denoted by an "x" dot in the plots.

5.1 mcsta and Imca



The plot on the right compares the runtime of mcsta and Imca on time-bounded ("tbr"), long-run average ("lra"), and unbounded properties ("unb"). The input of Imca is an explicit representation of a state space (cf. Sect. 3.2). Thus, before a model can be analysed with Imca, the state space has to be fully explored, transformed into this format, and saved to disk. This takes additional time and memory. Models of a few kB in Modest lead to Imca files of several GB. We use mcsta to perform this transformation, which took up to 200 seconds on each of the benchmarks we selected for our experiments. The runtime presented for Imca does not include the time to generate input models, but only the time it takes to load them into memory and analyse them. For mcsta, we do include the time for state space exploration (from Modest or Jani input). For all experiments, we chose the best runtime among all algorithms provided in each tool. For time-bounded properties we set the precision to 10^{-3} and 10^{-6}. The same holds for long-run properties for mcsta, but not for Imca since its command-line interface does not support setting the precision for these properties. For unbounded properties we use the default parameters of both tools, including precision, since this once again cannot be changed for Imca.

We see that Imca performs far worse than mcsta. This is despite the fact that the considered runtime does not include time for model generation and that its only algorithm for time-bounded properties is unsound (with unsound methods tending to be faster than sound ones [35]), while the one of mcsta is sound. The performance gap is likely due to Imca only implementing the discretisation-based approach, which is known to be inefficient [12, 13], and not providing the most recent model checking algorithms for any of the property types.

5.2 mcsta and Storm

Storm, like mcsta, implements multiple and current algorithms. We thus present the results of this comparison in more detail. The runtimes for both tools include the time for state space exploration and for the numeric computations.

Time-bounded properties. Figure 7 summarises the comparison of time-bounded solvers in mcsta and Storm. Once again we run experiments with precision values 10^{-3} and 10^{-6} and configure the tools to produce sound results. In the top-left plot we compare the best runtime for each tool among the algorithms that it implements; in the bottom-left plot, we compare mcsta's and Storm's implementations of Unif+. In both comparisons, mcsta achieves better runtimes than Storm. In particular, mcsta has no timeouts in the best-algorithm comparison. In the Unif+ comparison, mcsta and Storm both time out in some cases, yet whenever mcsta times out on a model, Storm does so, too (the "x" dot on the 45° line is actually a superposition of several such dots here). The two plots on the right compare the runtime of the switch-step implementation in mcsta with Unif+ in both tools. We do not compare to the discretisation algorithm for time-bounded properties implemented in Storm due to the consistent reports [12, 13] of its inefficiency (which we confirmed in Sect. 5.1 with Imca). We observe that neither Unif+ nor switch-step dominates the other, no matter which tool is used. This is because none of the two algorithms is strictly better than the other. Consider the top-right plot: it compares switch-step and Unif+ in mcsta and confirms the results presented in [12] that the algorithms are good in complementary scenarios. There are cases where one of them times out while the other finishes quite fast, and vice-versa. In particular, Unif+ performs somewhat better when a lower precision is required. Overall, the individual algorithms for

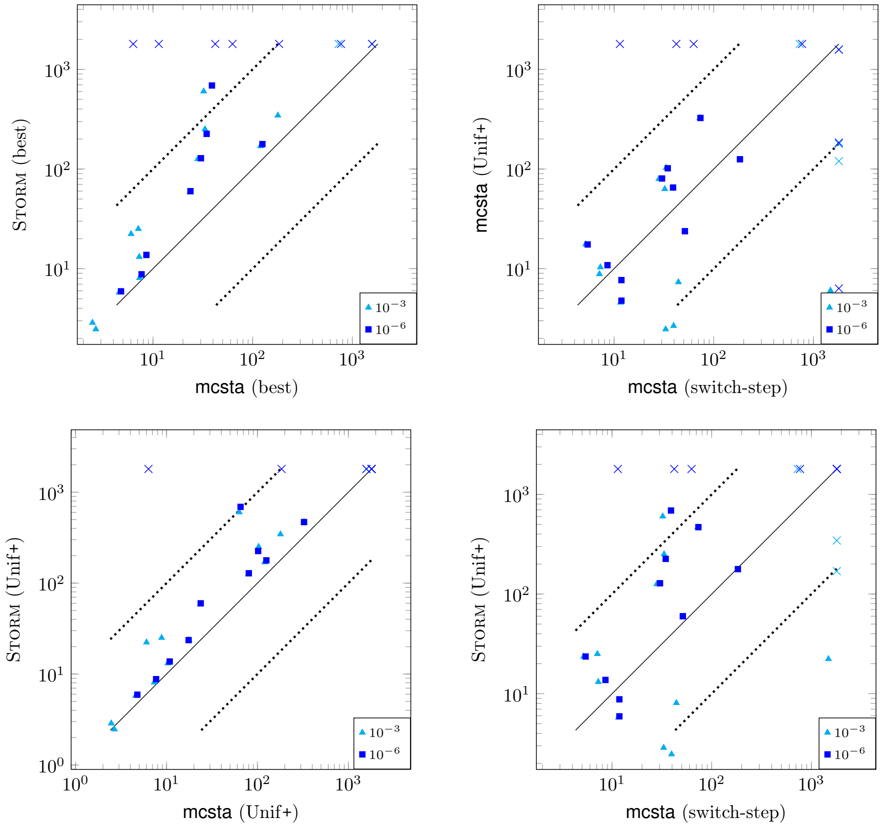


Fig. 7. Runtime of mcsta and Storm on time-bounded properties

time-bounded reachability in mcsta perform competitively, and especially when combined in a portfolio approach (i.e. using the best for each model, which could practically be done by running both concurrently on a multi-core system), offer noticeably better performance and scalability than Storm overall.

Long-run average properties. Figure 8 summarises the comparison of algorithms for model checking long-run average properties in mcsta and Storm. For value iteration-based algorithms ("VI"), we run experiments on precision values 10^{-3} and 10^{-6}, and use only its sound variations. For the linear programming-based approaches ("LP"), we set mcsta and Storm to use linear programming at *all* steps of the algorithm. The LP-based algorithms run with default parameters in both tools. For the top-left plot, we again chose the best runtime over the two algorithms for each tool. The LP-based approaches are not competitive: this can be seen from the three other plots. Here, the bottom-right plot shows that the LP-based algorithms in both mcsta and Storm run out of time on most of the benchmarks. In contrast, the VI-based solutions in both tools finish the

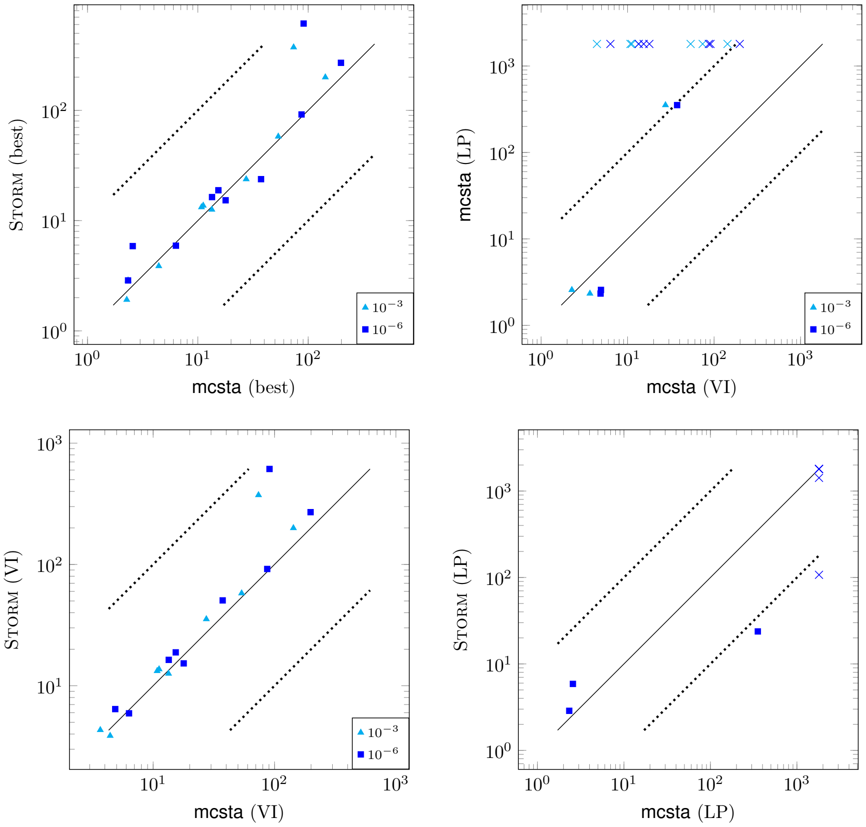


Fig. 8. Runtime of mcsta and Storm on long-run average reward properties

computations on the same benchmarks within the given time bound, as can be seen from the bottom-left and top-right plots. The exact reason for this is hard to extract. It may be possible that, when dealing with long-run properties, the LP-based approach itself is not as efficient as the one using VI, at least on existing benchmarks. Alternatively, it may be that the underlying LP algorithms or their implementations are not efficient. Overall, mcsta and Storm are roughly on par, albeit with mcsta having a few instances where it is significantly faster. The overall similarity is likely due to the set of implemented algorithms being exactly the same. We do notice, though, that specifically Storm's LP method appears to work better than mcsta's.

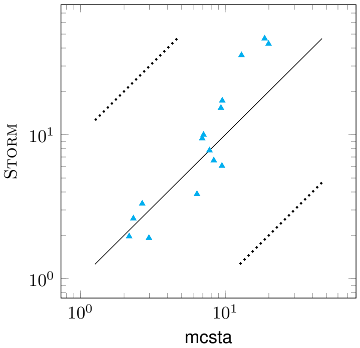
BRTDP. We compared exhaustive algorithms, i.e. those that perform computations on the full state space, with their BRTDP extensions in mcsta on a few benchmarks for time-bounded and unbounded properties. Table 1 summarises the results. BRTDP is useful in cases where the property under consideration does not require the full state space to be explored in order to achieve results

Table 1. Runtime of BRTDP vs. exhaustive algorithms on time-bounded properties

	vgs $(5, 10000)$	stream (20000)	ftwc $(512, 10)$	hecs $(\text{false}, 4, 3)$
BRTDP	6.07 sec	1.91 sec	6.69 sec	5.78 sec
exhaustive	$>$ 30 min	$>$ 30 min	1077 sec	$>$ 30 min

with specified precision. In fact, the explored state space might be only a few percent of the full state space. Table 1 confirms that, in certain cases, these approaches can perform substantially better than their exhaustive counterparts. Precision is set to 10^{-3} here.

Unbounded properties. We finally add a small evaluation for model checking unbounded properties. These properties can be checked via standard MDP algorithms and are thus not the focus of this paper. An extensive evaluation of such properties for both mcsta and Storm was done for the QComp 2019 tool competition [26]. The plot on the right confirms the QComp results of the two tools being competitive with no absolute winner.



6 Conclusion

We have presented a fully integrated toolchain to create and model check Markov automata models based on the high-level compositional modelling language Modest and the mcsta model checker of the Modest Toolset. Other tools of the Modest Toolset complement the approach, such as the modes simulator that helps deal with models too large for traditional model checking, or the moconv tool that can export Modest models to Jani. We have compared the performance of the dedicated MA model checking algorithms in mcsta with Imca and Storm. We found mcsta to significantly outperform Imca, and to be faster than Storm in many cases. The Jani support in both the Modest Toolset and Storm allows the user to choose the most appropriate tool in every instance, thus mcsta and Storm ought to be seen as complementary tools for a common goal. Overall, Markov automata now have user-friendly modelling and efficient verification support in tools that are actively maintained.

Data availability. The data generated in our experimental evaluation as well as instructions to replicate the experiments are archived and available at DOI 10.4121/uuid:98d571be-cdd4-4e5a-a589-7c5b1320e569 [11].

References

1. Amparore, E.G., Balbo, G., Beccuti, M., Donatelli, S., Franceschinis, G.: 30 years of GreatSPN. In: Principles of Performance and Reliability Modeling and Evaluation. pp. 227–254. Springer (2016)
2. Ashok, P., Butkova, Y., Hermanns, H., Kretínský, J.: Continuous-time Markov decisions based on partial exploration. In: ATVA. LNCS, vol. 11138, pp. 317–334. Springer (2018)
3. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Handbook of Model Checking., pp. 963–999. Springer (2018)
4. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Performance evaluation and model checking join forces. Commun. ACM 53(9), 76–85 (2010)
5. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: Interval iteration for Markov decision processes. In: CAV. LNCS, vol. 10426, pp. 160–180. Springer (2017)
6. Boudali, H., Crouzen, P., Stoelinga, M.: A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE Trans. Dependable Sec. Comput. 7(2), 128–143 (2010)
7. Brázdil, T., Chatterjee, K., Chmelik, M., Forejt, V., Kretínský, J., Kwiatkowska, M.Z., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: ATVA. LNCS, vol. 8837, pp. 98–114. Springer (2014)
8. Brázdil, T., Hermanns, H., Krcál, J., Kretínský, J., Rehák, V.: Verification of open interactive Markov chains. In: FSTTCS. LIPIcs, vol. 18, pp. 474–485. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik (2012)
9. Budde, C.E., D'Argenio, P.R., Hartmanns, A., Sedwards, S.: A statistical model checker for nondeterminism and rare events. In: TACAS. LNCS, vol. 10806, pp. 340–358. Springer (2018)
10. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: Quantitative model and tool interaction. In: TACAS. LNCS, vol. 10206, pp. 151–168 (2017)
11. Butkova, Y.: A Modest approach to modelling and checking Markov automata (artifact). 4TU.Centre for Research Data (2019), `https://doi.org/10.4121/uuid:98d571be-cdd4-4e5a-a589-7c5b1320e569`
12. Butkova, Y., Fox, G.: Optimal time-bounded reachability analysis for concurrent systems. In: TACAS. LNCS, vol. 11428, pp. 191–208. Springer (2019)
13. Butkova, Y., Hatefi, H., Hermanns, H., Krcál, J.: Optimal continuous time Markov decisions. In: ATVA. LNCS, vol. 9364, pp. 166–182. Springer (2015)
14. Butkova, Y., Wimmer, R., Hermanns, H.: Long-run rewards for Markov automata. In: TACAS. LNCS, vol. 10206, pp. 188–203 (2017)
15. Butkova, Y., Wimmer, R., Hermanns, H.: Markov automata on discount! In: MMB. LNCS, vol. 10740, pp. 19–34. Springer (2018)
16. D'Argenio, P.R., Hartmanns, A., Sedwards, S.: Lightweight statistical model checking in nondeterministic continuous time. In: ISoLA. LNCS, vol. 11245, pp. 336–353. Springer (2018)
17. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A Storm is coming: A modern probabilistic model checker. In: CAV. LNCS, vol. 10427, pp. 592–600. Springer (2017)
18. Eisentraut, C.: Principles of Markov automata. Ph.D. thesis, Saarland University, Germany (2017)

19. Eisentraut, C., Hermanns, H., Katoen, J.P., Zhang, L.: A semantics for every GSPN. In: Petri Nets. LNCS, vol. 7927, pp. 90–109. Springer (2013)
20. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS. pp. 342–351. IEEE Computer Society (2010)
21. Gros, T.P.: Markov automata taken by Storm. Master's thesis, Saarland University, Germany (2018)
22. Guck, D., Han, T., Katoen, J.P., Neuhäußer, M.R.: Quantitative timed analysis of interactive Markov chains. In: NFM. LNCS, vol. 7226, pp. 8–23. Springer (2012)
23. Guck, D., Hatefi, H., Hermanns, H., Katoen, J.P., Timmer, M.: Analysis of timed and long-run objectives for Markov automata. Logical Methods in Computer Science 10(3) (2014)
24. Guck, D., Timmer, M., Hatefi, H., Ruijters, E., Stoelinga, M.: Modelling and analysis of Markov reward automata. In: ATVA. LNCS, vol. 8837, pp. 168–184. Springer (2014)
25. Haddad, S., Monmege, B.: Interval iteration algorithm for MDPs and IMDPs. Theor. Comput. Sci. 735, 111–131 (2018)
26. Hahn, E.M., Hartmanns, A., Hensel, C., Klauck, M., Klein, J., Kretínský, J., Parker, D., Quatmann, T., Ruijters, E., Steinmetz, M.: The 2019 comparison of tools for the analysis of quantitative formal models. In: 25 Years of TACAS: TOOLympics. LNCS, vol. 11429, pp. 69–92. Springer (2019)
27. Hahn, E.M., Hartmanns, A., Hermanns, H., Katoen, J.P.: A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods in System Design 43(2), 191–232 (2013)
28. Hartmanns, A., Hermanns, H.: The Modest Toolset: An integrated environment for quantitative modelling and verification. In: TACAS. LNCS, vol. 8413, pp. 593–598. Springer (2014)
29. Hartmanns, A., Klauck, M., Parker, D., Quatmann, T., Ruijters, E.: The quantitative verification benchmark set. In: TACAS. LNCS, vol. 11427, pp. 344–350. Springer (2019)
30. Hatefi, H.: Finite horizon analysis of Markov automata. Ph.D. thesis, Saarland University, Germany (2017), scidok.sulb.uni-saarland.de/volltexte/2017/6743/
31. Krcál, J., Krcál, P.: Scalable analysis of fault trees with dynamic features. In: DSN. pp. 89–100. IEEE Computer Society (2015)
32. Legay, A., Sedwards, S., Traonouez, L.M.: Scalable verification of Markov decision processes. In: WS-FMDS at SEFM. LNCS, vol. 8938, pp. 350–362. Springer (2014)
33. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley & Sons, Inc. (1994)
34. Quatmann, T., Junges, S., Katoen, J.P.: Markov automata with multiple objectives. In: CAV. LNCS, vol. 10426, pp. 140–159. Springer (2017)
35. Quatmann, T., Katoen, J.P.: Sound value iteration. In: CAV. LNCS, vol. 10981, pp. 643–661. Springer (2018)
36. Rabe, M.N., Schewe, S.: Finite optimal control for time-bounded reachability in CTMDPs and continuous-time Markov games. Acta Inf. 48(5-6), 291–315 (2011)
37. Sullivan, K.J., Dugan, J.B., Coppit, D.: The Galileo fault tree analysis tool. In: FTCS-29. pp. 232–235. IEEE Computer Society (1999)
38. Timmer, M., Katoen, J.P., van de Pol, J., Stoelinga, M.: Efficient modelling and generation of Markov automata. In: CONCUR. LNCS, vol. 7454, pp. 364–379. Springer (2012)