Modelling and Analysis of Markov Reward Automata

Dennis Guck[1], Mark Timmer[1], Hassan Hatefi[2],
Enno Ruijters[1] and Mariëlle Stoelinga[1]

[1] Formal Methods and Tools, University of Twente, The Netherlands
[2] Dependable Systems and Software, Saarland University, Germany

Abstract. Costs and rewards are important ingredients for many types of systems, modelling critical aspects like energy consumption, task completion, repair costs, and memory usage. This paper introduces Markov reward automata, an extension of Markov automata that allows the modelling of systems incorporating *rewards* (or *costs*) in addition to nondeterminism, discrete probabilistic choice and continuous stochastic timing. Rewards come in two flavours: action rewards, acquired instantaneously when taking a transition; and state rewards, acquired while residing in a state. We present algorithms to optimise three reward functions: the expected cumulative reward until a goal is reached, the expected cumulative reward until a certain time bound, and the long-run average reward. We have implemented these algorithms in the SCOOP/IMCA tool chain and show their feasibility via several case studies.

1 Introduction

The design of computer systems involves many trade offs: Is it cost-effective to use multiple processors to increase availability and performance? Should we carry out preventive maintenance to save future repair costs? Can we reduce the clock speed to save energy, while still meeting the required performance bounds? How can we best schedule a task set so that the operational costs are minimised? Such optimisation questions typically involve the following ingredients: (1) rewards or costs, to measure the quality of the solution; (2) (stochastic) timing to model speed or delay; (3) discrete probability to model random phenomena like failures; and (4) nondeterminism to model the choices in the optimisation process.

This paper introduces Markov reward automata (MRAs), a novel model that combines the ingredients mentioned above. It is obtained by adding rewards to the formalism of Markov automata (MAs) [15]. We support two types of rewards: *Action rewards* are obtained directly when taking a transition, and *state rewards* model the reward per time unit while residing in a state. Such reward extensions have shown valuable in the past for less expressive models, for instance leading to the tool MRMC [24] for model checking reward-based properties over CTMCs [21] and DTMCs [1] with rewards. With our MRA model we provide a natural combination of the EMPA [3] and PEPA [9] reward formalisms.

By generalising MAs, MRAs provide a well-defined semantics for generalised stochastic Petri nets (GSPNs) [13], dynamic fault trees [4] and the domain-specific language AADL [5]. Recent work also demonstrated that MAs (and hence

MRAs as well) are suitable for modelling and analysing distributed algorithms such as a leader election protocol, performance models such as a polling system and hardware models such as a processor grid [31].

Model checking algorithms for MAs against Continuous Stochastic Logic (CSL) properties were discussed in [20]. Notions of strong, weak and branching bisimulation were defined to equate behaviourally equivalent MAs [15,28,12,31], and the process-algebraic language MAPA was introduced for easily specifying large MAs in a concise manner [32]. Several types of reduction techniques [34,33] have been defined for the MAPA language and implemented in the tool SCOOP, optimising specifications to decrease the state space of the corresponding MAs while staying bisimilar [30,18]. This way, MAs can be generated efficiently in a direct way (as opposed to first generating a large model and then reducing), thus partly circumventing the omnipresent state space explosion. Additionally, the game-based abstraction refinement technique developed in [6] provides a sound approximation of time-bounded reachability over a substantially reduced abstract model. The tool IMCA [17,18] was developed to analyse the concrete MAs that are generated by SCOOP. It includes algorithms for computing time-bounded reachability probabilities, expected times and long-run averages for sets of goal states within an MA.

While the framework in place already works well for computing probabilities and expected durations, it did not yet support rewards or costs. Therefore, we extend the MAPA language from MAs to the realm of MRAs and extend most of SCOOP's reduction techniques to efficiently generate them. Further, we present algorithms for three optimisation problems over MRAs. That is, we resolve the nondeterministic choices in the MRA such that one of three optimisation criteria is minimised (or maximised): (1) the expected cumulative reward to reach a set of goal states, (2) the expected cumulative reward until a given time bound, and (3) the long-run average reward.

The current paper is a first step towards a fully quantitative system design formalism. As such, we focus on positive rewards. Negative rewards, more complex optimisation criteria, as well as the handling of several rewards as multi-optimisation problem are important topics for future research. For a more detailed version of this paper with extended proofs we refer to [19].

2 Markov reward automata

MAs were introduced as the union of Interactive Markov Chains (IMCs) [23] and Probabilistic Automata (PAs) [27]. Hence, they feature nondeterminism, as well as Markovian rates and discrete probabilistic choice. We extend this model with reward functions for both the states and the transitions.

Definition 1 (Background). *A* probability distribution *over a countable set S is a function $\mu\colon S \to [0,1]$ such that $\sum_{s\in S} \mu(s) = 1$. For $S' \subseteq S$, let $\mu(S') = \sum_{s\in S'} \mu(s)$. We write $\mathbb{1}_s$ for the* Dirac distribution *for s determined by $\mathbb{1}_s(s) = 1$. We use $\mathsf{Distr}(S)$ to denote the set of all probability distributions over S.*

Given an equivalence relation $R \subseteq S \times S$, we write $[s]_R$ for the equivalence class of s induced by R, i.e., $[s]_R = \{s' \in S \mid (s, s') \in R\}$. Given two probability distributions $\mu, \mu' \in \mathsf{Distr}(S)$ and an equivalence relation R, we write $\mu \equiv_R \mu'$ to denote that $\mu([s]_R) = \mu'([s]_R)$ for every $s \in S$.

2.1 Markov reward automata

Before defining MRAs, we recall the definition of MAs. It assumes a countable universe of actions *Act*, with $\tau \in Act$ the invisible internal action.

Definition 2 (Markov Automata). *A* Markov automaton (MA) *is a tuple $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$, where*

- *S is a countable set of* states*, of which $s^0 \in S$ is the* initial state*;*
- *$A \subseteq Act$ is a countable set of* actions*, including τ;*
- *$\hookrightarrow \subseteq S \times A \times \mathsf{Distr}(S)$ is the* probabilistic transition relation*;*
- *$\rightsquigarrow \subseteq S \times \mathbb{R}_{>0} \times S$ is the* Markovian transition relation*;*

If $(s, \alpha, \mu) \in \hookrightarrow$, we write $s \xhookrightarrow{\alpha} \mu$ and say that action α can be executed *from state s, after which the probability to go to each $s' \in S$ is $\mu(s')$. If $(s, \lambda, s') \in \rightsquigarrow$, we write $s \overset{\lambda}{\rightsquigarrow} s'$ and say that s moves to s' with rate λ.*

A state $s \in S$ that has at least one transition $s \xhookrightarrow{a} \mu$ is called *probabilistic*. A state that has at least one transition $s \overset{\lambda}{\rightsquigarrow} s'$ is called *Markovian*. Note that a state could be both probabilistic and Markovian.

The *rate* between two states $s, s' \in S$ is $\mathbf{R}(s, s') = \sum_{(s,\lambda,s') \in \rightsquigarrow} \lambda$, and the *outgoing rate of s* is $E(s) = \sum_{s' \in S} \mathbf{R}(s, s')$. We require $E(s) < \infty$ for every state $s \in S$. If $E(s) > 0$, the *branching probability distribution* after this delay is denoted by $\mathbb{P}_s$ and defined by $\mathbb{P}_s(s') = \frac{\mathbf{R}(s,s')}{E(s)}$ for every $s' \in S$. By definition of the exponential distribution, the probability of leaving a state s within t time units is given by $1 - e^{-E(s) \cdot t}$ (given $E(s) > 0$), after which the next state is chosen according to $\mathbb{P}_s$. Further, we denote by $A(s)$ the set of all enabled actions in state s.

MAs adhere to the *maximal progress assumption*, prescribing τ-transitions to never be delayed. Hence, a state that has at least one outgoing τ-transition can never take a Markovian transition. This fact is captured below in the definition of extended transitions, which is used to provide a uniform manner for dealing with both probabilistic and Markovian transitions.

Definition 3 (Extended action set). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be an MA, then the* extended action set of $\mathcal{M}$ *is given by $A^\chi = A \cup \{\chi(r) \mid r \in \mathbb{R}_{>0}\}$. The actions $\chi(r)$ represent exit rates and are used to distinguish probabilistic and Markovian transitions. For $\alpha = \chi(\lambda)$, we define $E(\alpha) = \lambda$. If $\alpha \in A$, we set $E(\alpha) = 0$. Given a state $s \in S$ and an action $\alpha \in A^\chi$, we write $s \xrightarrow{\alpha} \mu$ if either*

- *$\alpha \in A$ and $s \xhookrightarrow{\alpha} \mu$, or*
- *$\alpha = \chi(E(s))$, $E(s) > 0$, $\mu = \mathbb{P}_s$ and there is no μ' such that $s \xhookrightarrow{\tau} \mu'$.*

A transition $s \xrightarrow{\alpha} \mu$ *is called an* extended transition. *We use* $s \xrightarrow{\alpha} t$ *to denote* $s \xrightarrow{\alpha} \mathbb{1}_t$, *and write* $s \to t$ *if there is at least one action* α *such that* $s \xrightarrow{\alpha} t$. *We write* $s \xrightarrow{\alpha,\mu} s'$ *if there is an extended transition* $s \xrightarrow{\alpha} \mu$ *such that* $\mu(s') > 0$.

Note that each state has an extended transition per probabilistic transition, while it has only one for all its Markovian transitions together (if there are any).

We now formally introduce the MRA. For simplicity of the reward functions, we chose to define MRAs in terms of extended actions. Hence, instead of two separate probabilistic and Markovian transition relations, there is only one transition relation. This also simplifies the notion of bisimulation introduced later.

Definition 4 (Markov Reward Automata). *A* Markov Reward Automaton (MRA) *is a tuple* $\mathcal{M} = \langle S, s^0, A, T, \rho \rangle$, *where*

- S *is a countable set of* states, *of which* $s^0 \in S$ *is the* initial state*;*
- $A \subseteq Act$ *is a countable set of* actions*;*
- $T \subseteq S \times A^\chi \times \mathbb{R}_{\geq 0} \times \mathsf{Distr}(S)$ *is the* transition relation *including* action rewards*;*
- $\rho \colon S \to \mathbb{R}_{\geq 0}$ *is the* state-reward function.

We require for each $s \in S$ *that there is at most one transition labeled with* $\chi(\cdot)$. *Further, we require that* T *is countable and write* $s \xrightarrow{\alpha}_r \mu$ *if* $(s, \alpha, r, \mu) \in T$.

The function ρ associates a real number to each state. This number may be zero, indicating the absence of a reward. The state-based rewards are gained *while being in a state*, and are proportional to the duration of this stay. The action-based rewards are gained instantaneously *when taking a transition* and are included directly in the transition relation.

2.2 Paths, policies and rewards

As for traditional labelled transition systems (LTSs), the behaviour of MAs and MRAs can also be expressed by means of paths. A *path* in $\mathcal{M}$ is a finite sequence $\pi^{\text{fin}} = s_0 \xrightarrow{a_1,\mu_1,t_1}_{r_1} s_1 \xrightarrow{a_2,\mu_2,t_2}_{r_2} \dots \xrightarrow{a_n,\mu_n,t_n}_{r_n} s_n$ from some state s_0 to a state s_n ($n \geq 0$), or an infinite sequence $\pi^{\text{inf}} = s_0 \xrightarrow{a_1,\mu_1,t_1}_{r_1} s_1 \xrightarrow{a_2,\mu_2,t_2}_{r_2} s_2 \xrightarrow{a_3,\mu_3,t_3} \dots$, with $s_i \in S$ for all $0 \leq i \leq n$ and all $0 \leq i$, respectively. The step $s_i \xrightarrow{a_i,\mu_i,t_i}_{r_i} s_{i+1}$ denotes that after residing t_i time units in s_i, the MRA has moved via action a_i and probability distribution μ_i to s_{i+1} obtaining r_i action reward. We use $\mathit{prefix}(\pi, t)$ to denote the prefix of path π up to and including time t, formally $\mathit{prefix}(\pi, t) = s_0 \xrightarrow{a_1,\mu_1,t_1}_{r_1} \dots \xrightarrow{a_i,\mu_i,t_i}_{r_i} s_i$ such that $t_1 + \dots + t_i \leq t$ and $t_1 + \dots + t_i + t_{i+1} > t$. We use $\mathit{step}(\pi, i)$ to denote the transition $s_{i-1} \xrightarrow{a_i}_{r_i} \mu_i$. When π is finite we define $|\pi| = n$, $\mathit{last}(\pi) = s_n$, and for every path $\pi[i] = s_i$. Further, we denote by π^j the path π up to and including state s_j. Let paths^* and paths denote the set of finite and infinite paths, respectively. We define the *total reward* of a finite path π by

$$\mathit{reward}(\pi) = \sum_{i=1}^{|\pi|} \rho(\pi[i-1]) \cdot t_i + r_i \tag{1}$$

Rewards can be used to model many quantitative aspects of systems, like energy consumption, memory usage, deployment or maintenance costs, etc. The total reward of a path (e.g., total amount of energy consumed) is obtained by adding all rewards along that path, that is, all state rewards multiplied by the sojourn times of the corresponding states plus all action rewards on the path.

Policies. Policies resolve the nondeterministic choices in an MRA, i.e., make a choice over the outgoing probabilistic transitions in a state. Given a policy, the behaviour of an MRA is fully probabilistic. Formally, a *policy*, ranged over by D, is a measurable function such that $D\colon \mathit{paths}^* \to \mathsf{Distr}(T)$ and $D(\pi)$ chooses only from transitions that emanate from $\mathit{last}(\pi)$. We denote by GM (generic measurable) the most general class of such policies which are measurable; for more details on measurability see [25]. Policies are classified based on the level of information they used to resolve nondeterminism. A stationary deterministic policy is a mapping $D\colon S \to T$ such that $D(s)$ chooses only from transitions that emanate from s; such policies always take the same transition in a state s. A time-dependent policy may decide on the basis of the states visited so far and their timings. For more details about different classes of policies and their relations we refer to [26]. Given a policy D and an initial state s, a measurable set of paths is equipped with the probability measure $\Pr_{s,D}$.

2.3 Strong bisimulation

We define a notion of strong bisimulation for MRAs. As for LTSs, PAs, IMCs and MAs, it equates systems the are equivalent in the sense that every step of one system can be mimicked by the other, and vice versa.

Definition 5 (Strong bisimulation). *Given an MRA $\mathcal{M} = \langle S, s^0, A, T, \rho\rangle$, an equivalence relation $R \subseteq S \times S$ is a* strong bisimulation *for $\mathcal{M}$ if for every $(s, s') \in R$ and all $\alpha \in A^\chi, \mu \in \mathsf{Distr}(S), r \in \mathbb{R}_{\geq 0}$, it holds that $\rho(s) = \rho(s')$ and*

$$s \xrightarrow{\alpha}_r \mu \implies \exists \mu' \in \mathsf{Distr}(S)\ .\ s' \xrightarrow{\alpha}_r \mu' \wedge \mu \equiv_R \mu'$$

Two states $s, s' \in S$ are strongly bisimilar *(denoted by $s \approx s'$) if there exists a strong bisimulation R for $\mathcal{M}$ such that $(s, s') \in R$. Two MAs $\mathcal{M}, \mathcal{M}'$ are strongly bisimilar (denoted by $\mathcal{M} \approx \mathcal{M}'$) if their initial states are strongly bisimilar in their disjoint union.*

Clearly, when setting all state-based and action-based rewards to 0, MRAs coincide with MAs. Additionally, our definition of strong bisimulation then reduces to the definition of strong bisimulation for MAs. Since it was already shown in [14] that strong bisimulation for MAs coincides with the corresponding notions for all subclasses of MAs, this also holds for our definition. Hence, it safely generalises the existing notions of strong bisimulation.

3 Quantitative analysis

This section shows how to perform quantitative analyses on MRAs. We will focus on three common reward measures: (1) The expected cumulative reward until reaching a set of goal states, (2) the expected cumulative reward until a given time-bound, and (3) the long-run average reward. Typical examples where these algorithms can be used are respectively: to minimise the average energy consumption needed to download and install a medium-size software update; to minimise the average maintenance cost of a railroad line over the first year of deployment; and to maximise the yearly revenues of a data center over a long time horizon. In the following we lift the algorithms from [18] to the realm of rewards. We focus on maximising the properties. The minimisation problem can be solved similarly — namely, by replacing max by min and sup by inf below.

3.1 Notation and preprocessing

Throughout this section, we consider a fixed MRA $\mathcal{M}$ with state space S and a set of goal states $G \subseteq S$. To facilitate the algorithms, we first perform three preprocessing steps. (1) We consider only closed MRAs, which are not subject to further interaction. Therefore, we hide all actions (renaming them to τ), focussing on their induced rewards. (2) Due to the maximal progress assumption, a Markovian transition will never be executed from a state with outgoing τ-transitions. Hence, we remove such Markovian transitions. Thus, each state either has one outgoing Markovian transition or only probabilistic outgoing transitions. We call these states Markovian and probabilistic respectively, and use MS and PS to denote the sets of Markovian and probabilistic states. (3) To distinguish the different τ-transitions emerging from a state $s \in PS$, we assume w.l.o.g. that these are numbered from 1 to n_s, where n_s is the number of outgoing transitions. We write $\mu_s^{\tau_i}$ for the distribution induced by taking τ_i in state s and we write $r_s^{\tau_i}$ for the reward. For Markovian transitions we write $\mathbb{P}_s$ and r_s, respectively.

3.2 Goal-bounded expected cumulative reward

We are interested in the minimal and maximal expected cumulative reward until reaching a set of goal states $G \subseteq S$. That is, we accumulate the state and transition rewards until a state in G is reached; if no state in G is reached, we keep on accumulating rewards.

The random variable $V_G \colon \mathit{paths} \to \mathbb{R}_{\geq 0}^{\infty}$ yields the accumulated reward before first visiting some state in G. For an infinite path π, we define

$$V_G(\pi) = \begin{cases} \mathit{reward}(\pi^j) & \text{if } \pi[j] \in G \ \wedge \ \forall i < j.\ \pi[i] \notin G \\ \mathit{reward}(\pi) & \text{if } \forall i.\ \pi[i] \notin G \end{cases}$$

The maximal expected reward to reach G from $s \in S$ is then defined as

$$\mathsf{eR}^{\max}(s, G) = \sup_{D \in GM} \mathbb{E}_{s,D}(V_G) = \sup_{D \in GM} \int_{\mathit{paths}} V_G(\pi) \Pr_{s,D}(d\pi) \tag{2}$$

where D is an arbitrary policy on $\mathcal{M}$.

To compute $\mathsf{eR}^{\max}$ we turn it into a classical Bellman equation: For all goal states, no more reward is accumulated, so their expected reward is zero. For Markovian states $s \notin G$, the state reward of s is weighted with the expected sojourn time in s plus the expected reward accumulated via its successor states plus the transition reward to them. For a probabilistic state $s \notin G$, we select the action that maximises the expected cumulative reward. Note that, since the accumulated reward is only relevant until reaching a state in G, we may turn all states in G into absorbing Markovian states.

Theorem 1 (Bellman equation) *The function* $\mathsf{eR}^{\max} \colon S \to \mathbb{R}^{\infty}_{\geq 0}$ *is the unique fixed point of the Bellman equation*

$$v(s) = \begin{cases} \frac{\rho(s)}{E(s)} + \sum\limits_{s' \in S} \mathbb{P}_s(s') \cdot (v(s') + r_s) & \text{if } s \in MS \setminus G \\ \max\limits_{\alpha \in A(s)} \sum\limits_{s' \in S} \mu_s^{\alpha}(s') \cdot (v(s') + r_s^{\alpha}) & \text{if } s \in PS \setminus G \\ 0 & \text{if } s \in G. \end{cases}$$

A direct consequence of Theorem 1 is that the supremum in (2) is attained by a stationary deterministic policy. Moreover, this result enables us to use standard solution techniques such as value iteration and linear programming to compute $\mathsf{eR}^{\max}(s, G)$. Note that by assigning $\rho(s) = 1$ to all $s \in MS$ and setting all other rewards to 0, we compute the expected time to reach a set of goal states.

3.3 Time-bounded expected cumulative reward

A time-bounded reward is the reward gained until a time bound t is reached and is denoted by the random variable $\mathit{reward}(\cdot, t)$. For an infinite path π, we first find the prefix of π up to t and then compute the reward using (1), i. e.

$$\mathit{reward}(\pi, t) = \mathit{reward}(\mathit{prefix}(\pi, t)) \tag{3}$$

The maximum time-bounded reward then is the maximum expected reward gained within some interval $I = [0, b]$, starting from some initial state s:

$$\mathcal{R}^{\max}(s, b) = \sup_{D \in GM} \int_{paths} \mathit{reward}(\pi, b) \Pr_{s,D}(d\pi) \tag{4}$$

Similar to time-bounded reachability there is a fixed point characterisation (FPC) for computing the optimal reward within some interval of time. Here we focus on the maximum case; the minimum can be extracted similarly.

Lemma 2 (Fixed Point Characterisation) *Given a Markov reward automaton* $\mathcal{M}$ *and a time bound* $b \geq 0$*. The maximum expected cumulative reward from state* $s \in S$ *until time bound* b *is the least fixed point of higher order operator*

$\Omega\colon (S \times \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0}) \mapsto (S \times \mathbb{R}_{\geq 0} \mapsto \mathbb{R}_{\geq 0})$, *such that*

$$\Omega(F)(s,b) = \begin{cases} \left(r_s + \frac{\rho(s)}{E(s)}\right)(1 - e^{-E(s)b}) \\ \quad + \int_0^b E(s) e^{-E(s)t} \sum_{s' \in S} \mathbb{P}_s(s') F(s', b-t)\, \mathrm{d}t & s \in MS \wedge b \neq 0 \\ \max_{\alpha \in A(s)} \left(r_s^\alpha + \sum_{s' \in S} \mu_s^\alpha(s') F(s', b)\right) & s \in PS \\ 0 & \textit{otherwise.} \end{cases}$$

This FPC is a generalisation of that for time-bounded reachability [18, Lemma 1], taking both action and state rewards into account. The proof goes along the same lines as that of [25, Theorem 6.1].

Discretisation. Similar to time-bounded reachability, the FPC is not algorithmically tractable and needs to be discretised: we have to divide the time horizon $[0, b]$ into a (generally large) number of equidistant time steps, each of length $0 < \delta \leq b$, such that $b = k\delta$ for some $k \in \mathbb{N}$. First, we express $\mathcal{R}^{\max}(s, b)$ in terms of its behaviour in the first discretisation step $[0, \delta)$. To do so, we partition the paths from s into the set $\mathcal{P}_1$ of paths that make their first Markovian jump in $[0, \delta)$ and the set $\mathcal{P}_2$ of paths that do not. We write $\mathcal{R}^{\max}(s, b)$ as the sum of

1. The expected reward obtained in $[0, \delta)$ by paths from $\mathcal{P}_1$
2. The expected reward obtained in $[\delta, b]$ by paths from $\mathcal{P}_1$
3. The expected reward obtained in $[0, \delta)$ by paths from $\mathcal{P}_2$
4. The expected reward obtained in $[\delta, b]$ by paths from $\mathcal{P}_2$

It turns out to be convenient to combine the first three items, denoted by $A(s, b)$, since the resulting term resembles the expression in Lemma 2:

$$\begin{aligned} A(s,b) &= \rho(s)\delta e^{-E(s)\delta} + \int_0^\delta E(s) e^{-E(s)t} \Big(\rho(s)t + r_s + \sum_{s' \in S} \mathbb{P}_s(s') \mathcal{R}^{\max}(s', b-t)\Big)\, \mathrm{d}t \\ &= \Big(r_s + \frac{\rho(s)}{E(s)}\Big)\Big(1 - e^{-E(s)\delta}\Big) + \int_0^\delta E(s) e^{-E(s)t} \sum_{s' \in S} \mathbb{P}_s(s') \mathcal{R}^{\max}(s', b-t)\, \mathrm{d}t \end{aligned} \tag{5}$$

where the first equality follows directly from the definition of $A(s, b)$ and the second equality is along the same lines as the proof of Lemma 2. It can easily be seen that $\mathcal{R}^{\max}(s, b) = A(s, b) + e^{-E(s)\delta} \mathcal{R}^{\max}(s, b - \delta)$.

Exact computation of $A(s, b)$ is in general still intractable due to the term $\mathcal{R}^{\max}(s', b - t)$. However, if the discretisation constant δ is very small, then, with high probability, at most one Markovian jump happens in each discretisation step. Hence, the reward gained by paths having multiple Markovian jumps within at least one such interval is negligible and can be omitted from the computation, while introducing only a small error. Technically, that means that we don't have to remember the remaining time within a discretisation step after a Markovian jump has happened. We can therefore discretise $A(s, b)$ into $\tilde{A}_\delta(s, k)$ and $\mathcal{R}^{\max}(s, b)$ into $\tilde{\mathcal{R}}^{\max}_\delta(s, k)$, just counting the number of discretisation steps k that are left instead of the actual time bound b:

$$\tilde{\mathcal{R}}^{\max}_\delta(s,k) = \tilde{A}_\delta(s,k) + e^{-E(s)\delta} \tilde{\mathcal{R}}^{\max}_\delta(s, k-1), \quad s \in MS \tag{6}$$

where $\tilde{A}_\delta(s,k)$ is defined by

$$\begin{aligned}\tilde{A}_\delta(s,k) &= \Big(r_s + \frac{\rho(s)}{E(s)}\Big)\Big(1 - e^{-E(s)\delta}\Big) + \int_0^\delta E(s)e^{-E(s)t} \sum_{s'\in S} \mathbb{P}_s(s')\tilde{\mathcal{R}}^{\max}_\delta(s', k-1)\,\mathrm{d}t \\ &= \Big(r_s + \frac{\rho(s)}{E(s)} + \sum_{s'\in S} \mathbb{P}_s(s')\tilde{\mathcal{R}}^{\max}_\delta(s', k-1)\Big)\Big(1 - e^{-E(s)\delta}\Big) \qquad (7)\end{aligned}$$

Note that we used $\tilde{\mathcal{R}}^{\max}_\delta(s,k-1)$ instead of both $\mathcal{R}^{\max}(s,b-\delta)$ and $\mathcal{R}^{\max}(s,b-t)$.

Eq. (6) and (7) help us to establish a tractable discretised version of the FPC described in Lemma 2 and to formally define the discretised maximum time-bounded reward afterwards:

Definition 6 (Discretised Maximum Time-Bounded Reward). *Let $\mathcal{M}$ be an MRA, $b \geq 0$ a time bound and $\delta > 0$ a discretisation step such that $b = k\delta$ for some $k \in \mathbb{N}$. The discretised maximum time-bounded cumulative reward, $\tilde{\mathcal{R}}^{\max}_\delta$, is defined as the least fixed point of higher order operator $\Omega_\delta \colon (S \times \mathbb{N} \mapsto \mathbb{R}_{\geq 0}) \mapsto (S \times \mathbb{N} \mapsto \mathbb{R}_{\geq 0})$, such that*

$$\Omega_\delta(F)(s,k) = \begin{cases} \big(r_s + \frac{\rho(s)}{E(s)} + \sum_{s'\in S} \mathbb{P}_s(s')F(s',k-1)\big)\big(1 - \mathrm{e}^{-E(s)\delta}\big) & \\ \qquad +\mathrm{e}^{-E(s)\delta}F(s,k-1) & s \in MS \wedge k \neq 0 \\ \max_{\alpha\in A(s)} \big(r^\alpha_s + \sum_{s'\in S} \mu^\alpha_s(s')F(s',k)\big) & s \in PS \\ 0 & \textit{otherwise.} \end{cases}$$

The reason behind the tractability of $\tilde{\mathcal{R}}^{\max}_\delta$ is hidden in Eq. (7). It brings two simplifications to the computation. First, it implies that $\tilde{\mathcal{R}}^{\max}_\delta$ is the conditional expected reward given that each step carries at most one Markovian transition. Second, it neglects to compute the reward after the first Markovian jump and simply assume that it is zero. We have shown the formal specification of the simplifications in [19, Lemma C1]. With the help of these simplifications, reward computation becomes tractable but indeed inexact.

The accuracy of $\tilde{\mathcal{R}}^{\max}_\delta$ depends on some parameters including the step size δ. The smaller δ is, the better the quality of discretisation is. It is possible to quantify the quality of the discretisation. To this end we need first to define some parameters of MRA. For a given MRA $\mathcal{M}$, assume that λ is the maximum exit rate of any Markovian state, i. e. $\lambda = \max_{s\in MS} E(s)$, and $\rho_{\max}$ is maximum state reward of any Markovian state, i. e. $\rho_{\max} = \max_{s\in MS} \rho(s)$. Moreover we define $r_{\max}$ as the maximum action reward that can be gained between two consecutive Markovian jumps. The value can be computed via Theorem 1, where we set Markovian states as the goal states. Given that $\mathsf{eR}^{\max}(s, MS)$ has already been computed, we define $r(s) = r_s + \sum_{s'\in S} \mathsf{eR}^{\max}(s', MS)$ for $s \in MS$, and $r(s) = \mathsf{eR}^{\max}(s, MS)$ otherwise. Finally we have $r_{\max} = \max_{s\in S} r(s)$. Note that in practice we use a value iteration algorithm to compute $r_{\max}$. With all of the parameters known, the following theorem quantifies the quality of the abstraction.

Theorem 3 *Let $\mathcal{M}$ be an MRA, $b \geq 0$ be a time bound, $\delta > 0$ be a discretisation step such that $b = k\delta$ for some $k \in \mathbb{N}$. Then for all $s \in S$:*

$$\tilde{\mathcal{R}}^{\max}_{\delta}(s,k) \leq \mathcal{R}^{\max}(s,b) \leq \tilde{\mathcal{R}}^{\max}_{\delta}(s,k) + \frac{b\lambda}{2}(\rho_{\max} + r_{\max}\lambda)(1 + \frac{b\lambda}{2})\delta$$

3.4 Long-run average reward

Next, we are interested in the average cumulative reward induced by a set of goal states $G \subseteq S$ in the long-run. Hence, all state and action rewards for states $s \in S \setminus G$ are set to 0. We define the random variable $\mathcal{L}_{\mathcal{M}} \colon \mathit{paths} \to \mathbb{R}_{\geq 0}$ as the long-run reward over paths in MRA $\mathcal{M}$. For an infinite path π let

$$\mathcal{L}_{\mathcal{M}}(\pi) = \lim_{t \to \infty} \frac{1}{t} \cdot \mathit{reward}(\pi, t).$$

Then, the maximal long-run average reward on $\mathcal{M}$ starting in state $s \in S$ is:

$$\mathsf{LRR}^{\max}_{\mathcal{M}}(s) = \sup_{D \in GM} \mathbb{E}_{s,D}(\mathcal{L}_{\mathcal{M}}) = \sup_{D \in GM} \int_{paths} \mathcal{L}_{\mathcal{M}}(\pi) \Pr_{s,D}(d\pi). \tag{8}$$

The computation of the expected long-run reward can be split into three steps:

1. Determine all maximal end components of MRA $\mathcal{M}$;
2. Determine $\mathsf{LRR}^{\max}_{\mathcal{M}_i}$ for each maximal end component $\mathcal{M}_i$;
3. Reduce the computation of $\mathsf{LRR}^{\max}_{\mathcal{M}}(s)$ to an SSP problem.

A sub-MRA $\mathcal{M}$ is a pair (S', K) where $S' \in S$ and K is a function that assigns to each state $s \in S'$ a non-empty set of actions, such that for all $\alpha \in K(s)$, $s \xrightarrow{\alpha} \mu$ with $\mu(s') > 0$ implies $s' \in S'$. An *end component* is a sub-MRA whose underlying graph is strongly connected; it is maximal (a *MEC*) w.r.t. K if it is not contained in any other end component (S'', K). In this section we focus on the second step. The first step can be performed by a graph-based algorithm [8,10] and the third step is as in [18].

A MEC can be seen as a unichain MRA: an MRA that yields a strongly connected graph structure under any stationary deterministic policy.

Theorem 4 *For a unichain MRA $\mathcal{M}$, for each $s \in S$ the value of $\mathsf{LRR}^{\max}_{\mathcal{M}}(s)$ equals*

$$\mathsf{LRR}^{\max}_{\mathcal{M}} = \sup_{D} \sum_{s \in S} \Big(\rho(s) \cdot \mathsf{LRA}^{\mathsf{D}}(s) + r_s^{D(s)} \cdot \nu^D(s) \Big)$$

where ν is the frequency of passing through a state, defined by

$$\nu^D(s) = \begin{cases} \mathsf{LRA}^D(s) \cdot E(s) & \text{if } s \in MS \\ \sum_{s' \in S} \nu^D(s') \cdot \mu_{s'}^{D(s')}(s) & \text{if } s \in PS \end{cases}$$

and $\mathsf{LRA}^D(s)$ is the long-run average time spent in state s under stationary deterministic policy D.

Thus, the frequency of passing through a Markovian state equals the long-run average time spent in s times the exit rate, and for a probabilistic state it is the accumulation of the frequencies of the incoming transitions. Hence, the long-run reward gathered by a state s is defined by the state reward weighted with the average time spent in s and the action reward weighted by the frequency of passing through the state. Since in a unichain MRA $\mathcal{M}$, for any two states s, s', $\mathsf{LRR}^{\max}_{\mathcal{M}}(s)$ and $\mathsf{LRR}^{\max}_{\mathcal{M}}(s')$ coincides, we omit the starting state and just write $\mathsf{LRR}^{\max}_{\mathcal{M}}$. Note that probabilistic states are left immediately, so $\mathsf{LRA}^{D}(s) = 0$ if $s \in PS$. Further, by assigning $\rho(s) = 1$ to all $s \in MS \cap G$ and setting all other rewards to 0, we compute the long-run average time spent in a set of goal states.

Theorem 5 *The long-run average reward of a unichain MRA coincides with the limit of the time-bounded expected cumulative reward, such that* $\mathsf{LRR}^{D}(s) = \lim_{t\to\infty} \frac{1}{t}\mathcal{R}^{D}(s,t)$.

For the equation from Theorem 4 it would be too expensive to compute for all possible policies and for each state the long-run average time as well as the frequency of passing through a state and weigh those with the associated rewards. Instead, we compute $\mathsf{LRR}^{\max}_{\mathcal{M}}$ by solving a system of linear inequations following the concepts of [10]. Given a unichain MRA $\mathcal{M}$, let k denote the optimal average reward accumulated in the long-run and executing the optimal policy. Then, for all $s \in S$ there is a function $h(s)$ that describes a differential cost per visit to state s, such that a system of inequations can be constructed as follows:

Minimise k subject to:

$$\begin{cases} h(s_i) = \frac{\overline{\rho}(s_i)}{E(s_i)} - \frac{k}{E(s_i)} + \sum_{s_j \in S} \mathbb{P}_{s_i}(s_j) \cdot h(s_j) & \text{if } s_i \in MS \\ h(s_i) \geq r^{\alpha}_{s_i} + \sum_{s_j \in S} \mu^{\alpha}_{s_i}(s_j) \cdot h(s_j) & \text{if } s_i \in PS \wedge \forall \alpha \in A(s_i) \end{cases} \tag{9}$$

where the state and action reward of Markovian states are combined as $\overline{\rho}(s_i) = \rho(s_i) + (r_{s_i} \cdot E(s_i))$. Standard linear programming algorithms, e.g., the simplex method [35], can be applied to solve the above system of linear equations.

To obtain the long-run average reward in an arbitrary MRA, we have to weigh the obtained long-run rewards in each maximal end component with the probability to reach those from s. This is equivalent to the third step in the long-run average computation of [18]. Further, for the discrete time setting [7] considers multiple long-run average objectives.

4 MAPA with rewards

The Markov Automata Process Algebra (MAPA) language allows MAs to be generated in an efficient and effective manner [31]. It is based on μCRL [16], allowing the standard process-algebraic constructs such as nondeterministic choice and action prefix to be used in a data-rich context: processes are equipped with a set of variables over user-definable data types, and actions can be parameterised

based on the values of these variables. Additionally, conditions can be used to restrict behaviour, and nondeterministic choices over data types are possible. MAPA adds two operators to μCRL: a probabilistic choice over data types and a Markovian delay (both possibly depending on data parameters).

We extended MAPA by accompanying it with rewards. Due to the action-based approach of process algebra, there is a clear separation between the action-based and state-based rewards. *Action-based rewards* are just added as decorations to the actions in the process-algebraic specification: we use $a[r]\sum_{x:D} f : p$ to denote an action a having reward r, continuing as process p (where the variable x gets a value from its domain D based on a probabilistic expression f). We refer to [31] for a detailed exposition of the syntax and semantics of MAPA; this is trivially generalised to incorporate the action-based rewards.

State-based rewards are dealt with separately. They can be assigned to *conditions*; each state that fulfills a reward's condition is then assigned that reward. If a state satisfies multiple conditions, the rewards are accumulated.

4.1 MAMA extensions

Since realistic systems often consist of a very large number of states, we do not want to construct their MRA models manually. Rather, we prefer to specify them as the parallel composition of multiple components. This approach was applied earlier to generate MAs, using a tool called SCOOP [30,18,31]. It generates MAs from MAPA specifications, applying several reduction techniques in the process. The underlying philosophy is to already reduce on the specification, not having to first generate a large model before being able to minimise. The parallel composition of MRAs is described in the technical report [19] and is equivalent to [11] for the probabilistic transitions.

We extended SCOOP to parse action-based and state-based rewards. Action-based rewards are stored as part of the transitions, while state-based rewards are represented internally by self-loops. Additionally, we generalised most of its reduction techniques to take into account the new rewards. The following reduction techniques are now also applicable to MRAs:

Dead variable reduction. This technique resets variables if their value is not needed anymore until they are overwritten. Instead of only checking whether a variable is used in conditions or actions, we generalised this technique to also check if it is used in reward expressions.

Maximal progress reduction. This technique removes Markovian transitions from states also having τ-transitions. It can be applied unchanged to MRAs.

Basic reduction techniques. The basic reduction techniques omit variables that are never changed, omit nondeterministic choices that only have one option and simplify expressions where possible. These three techniques were easily generalised by taking the reward expressions into account as well.

Confluence reduction was not yet generalised, as it is based on a much more complicated notion of bisimulation (that is not yet available for MRAs).

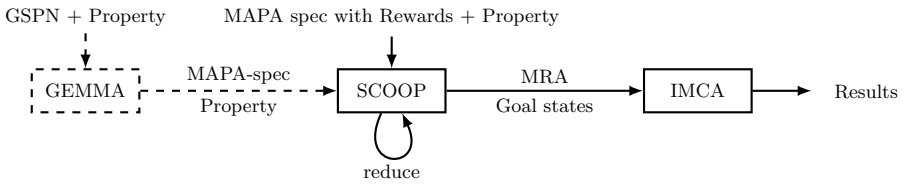

Fig. 1. Analysing Markov Reward Automata using the MAMA tool chain.

SCOOP takes both the action-based and state-based rewards into account when generating an input file for the IMCA toolset. This toolset implements several algorithms for computing reward-based properties, as detailed before. The connection of the tool-chain is depicted in Figure 1.

5 Case studies

To assess the performance of the algorithms and implementation, we provide two case studies: A server polling system based on [29], and a fault-tolerant workstation cluster based on [22]. Rewards were added to both examples. The experiments were conducted on a 2.2 GHz Intel® Core™ i7-2670QM processor with 8 GB RAM, running Linux.

Polling system. Figure 2 shows the MAPA specification of the polling system. It consists of two stations, each providing a job queue, and one server. When the server polls a job from a station, there is a 10% chance that it will erroneously remain in the queue. An impulse reward of 0.1 is given each time a server takes a job, and a reward of 0.01 per time unit is given for each job in the queue. The rewards are meant to be interpreted as costs in this example, for having a job processed and for taking up server memory, respectively.

Tables 1 and 3 show the results obtained by the MAMA tool-chain when analysing for different queue sizes Q and different numbers of job types N. The

$$\textbf{constant } \mathit{queueSize} = Q, \mathit{nrOfJobTypes} = N$$
$$\textbf{type } \mathit{Stations} = \{1,2\},\ \ \mathit{Jobs} = \{1,\ldots,\mathit{nrOfJobTypes}\}$$

$$\begin{aligned}\mathit{Station}(i:\mathit{Stations}, q:\text{Queue}, \mathit{size}:\{0..\mathit{queueSize}\}) \\ = \mathit{size} < \mathit{queueSize} \Rightarrow (2i+1)\cdot\textstyle\sum_{j:\mathit{Jobs}} \mathit{arrive}(j)\cdot \mathit{Station}(i,\text{enqueue}(q,j),\mathit{size}+1) \\ + \mathit{size} > 0 \quad \Rightarrow \mathit{deliver}(i,\text{head}(q))\cdot \sum_{k\in\{1,9\}} \tfrac{k}{10} : k = 1 \Rightarrow \mathit{Station}(i,q,\mathit{size}) \\ + k = 9 \Rightarrow \mathit{Station}(i,\text{tail}(q),\mathit{size}-1)\end{aligned}$$

$$\mathit{Server} = \textstyle\sum_{n:\mathit{Stations}} \sum_{j:\mathit{Jobs}} \mathit{poll}(n,j)\underline{[0.1]}\cdot(2*j)\cdot\mathit{finish}(j)\cdot\mathit{Server}$$

$$\gamma(\mathit{poll},\mathit{deliver}) = \mathit{copy} \qquad \text{// actions } \mathit{poll} \text{ and } \mathit{deliver} \text{ synchronise and yield action } \mathit{copy}$$

$$\mathit{System} = \tau_{\{\mathit{copy},\mathit{arrive},\mathit{finish}\}}(\partial_{\{\mathit{poll},\mathit{deliver}\}}(\mathit{Station}(1,\text{empty},0) \,||\, \mathit{Station}(2,\text{empty},0) \,||\, \mathit{Server}))$$

$$\underline{\textbf{state reward } \texttt{true} \rightarrow \ \mathit{size}_1 * 0.01 + \mathit{size}_2 * 0.01}$$

Fig. 2. MAPA specification of a nondeterministic polling system.

Q	N	T_{lim}	Time-bounded reward min	T(min)	max	T(max)
2	3	1	0.626	0.46	0.814	0.46
2	3	2	0.914	1.64	1.389	1.66
2	3	10	1.005	161.73	2.189	166.59
3	3	1	0.681	4.90	0.893	4.75
3	3	2	1.121	16.69	1.754	17.11
3	3	10	1.314	1653	4.425	1687

Table 1. Time-bounded rewards for the polling system (T in seconds).

N	Q	T_{lim}	Time-bounded reward min	T(min)	max	T(max)
4	3	10	0.0126	5.47	0.0126	5.53
4	3	20	0.0267	38.58	0.0267	38.98
4	3	50	0.0701	579.66	0.0701	576.13
4	3	100	0.143	4607	0.143	4540
4	5	10	0.0114	4.17	0.0114	4.23
4	5	20	0.0232	28.54	0.0232	28.75
4	5	50	0.0584	444.39	0.0584	442.63
4	5	100	0.1154	3520.18	0.1154	3521.70

Table 2. Time-bounded rewards for the workstation cluster (T in seconds).

goal states for the expected reward are those when both queues are full. The error-bound for the time-bounded reward analysis was set to 0.1.

The tables show that the minimal reward does not depend on the number of job types, while the maximal reward does. The long-run reward computation is, for this example, considerably slower than the expected reward, and both increase more than linear with the number of states. The time-bounded reward is more affected by the time bound than the number of states, and the computation time does not significantly differ between the maximal and minimal queries.

Workstation cluster. The second case study is based on a fault-tolerant workstation cluster, described as a GSPN in [25]. Using the GEMMA [2] tool, the GSPN was converted into a MAPA specification.

The workstation cluster consists of two groups of N workstations, each group connected by one switch. The two groups are connected to each other by a backbone. Workstations, switches and the backbone experience exponentially distributed failures, and can be repaired one at a time. If multiple components are eligible for repair at the same time, the choice is nondeterministic. The overall cluster is considered operational if at least Q workstations are operational and connected to each other. Rewards have been added to the system to simulate the costs of repairs and downtime. Repairing a workstation has cost 0.3, a switch costs 0.1, and the backbone costs 1 to repair. If fewer than Q workstations are operational and connected, a cost of 1 per unit time is incurred.

Tables 2 and 4 show the analysis results for this example. The goal states for the expected reward are the states where not enough operational workstations are connected. The error bound for the time-bounded reward analysis was 0.1. For this example, the long-run rewards are quicker to compute than the expected rewards. The long-run rewards do not vary much with the scheduler, since multiple simultaneous failures are rare in this system. This also explains the large expected rewards when Q is low: many repairs will occur before the cluster fails. The time-bounded rewards also show almost no dependence on the scheduler.

				Long-run reward				Expected reward			
Q	N	$\|S\|$	$\|G\|$	min	T(min)	max	T(max)	min	T(min)	max	T(max)
2	3	1159	405	0.731	0.61	1.048	0.43	0.735	0.28	2.110	0.43
2	4	3488	1536	0.731	3.76	1.119	2.21	0.735	0.93	3.227	2.01
3	3	11122	3645	0.750	95.60	1.107	19.14	1.034	3.14	4.752	8.14
3	4	57632	24576	0.750	5154.6	1.198	705.8	1.034	31.80	8.878	95.87
4	2	5706	1024	0.769	38.03	0.968	5.73	1.330	3.12	4.199	3.12
4	3	102247	32805	Timeout(2h)				1.330	63.24	9.654	192.18

Table 3. Long-run and expected rewards for the polling system (T in seconds).

				Long-run reward				Expected reward			
N	Q	$\|S\|$	$\|G\|$	min	T(min)	max	T(max)	min	T(min)	max	T(max)
4	3	1439	1008	0.00145	0.49	0.00145	0.60	2717	158.5	2718	138.2
4	5	1439	621	0.00501	0.45	0.00505	0.61	1.714	0.56	1.714	0.59
4	8	1439	1438	0.00701	0.48	0.00705	0.64	0	0.50	0	0.50
8	6	4876	3584	0.00145	2.18	0.00145	3.71	2896	783.7	2896	786.6
8	8	4876	4415	0.00146	1.93	0.00147	3.34	285.5	57.13	285.5	54.33
8	10	4883	4783	0.00501	1.92	0.00505	3.36	1.714	2.31	1.714	2.33
8	16	4895	4894	0.00701	2.09	0.00705	3.89	0	2.43	0	2.19

Table 4. Long-run and expected rewards for the workstation cluster (T in seconds).

6 Conclusions and future work

We introduced the Markov Reward Automaton (MRA), an extension of the Markov Automaton (MA) featuring both state-based and action-based rewards (or, equivalently, costs). We defined strong bisimulation for MRAs, and validated it by stating that our notion coincides with the traditional notions of strong bisimulation for MAs. We generalised the MAPA language to efficiently model MRAs by process-algabraic specifications, and extended the SCOOP tool to automatically generate MRAs from these specifications. Furthermore, we presented three algorithms, for computing the expected reward until reaching a set of goal states, for computing the expected reward until reaching a time-bound, and for computing the long-run average reward while visiting a set of states. Our modelling framework and algorithms allow for a wide variety of systems—featuring nondeterminism, discrete probabilistic choice, continuous stochastic timing and action-based and state-based rewards—to be efficiently modelled, generated and analysed.

Future work will focus on developing weak notions of bisimulation for MRAs, possibly allowing the generalisation of confluence reduction. For quantitative analysis, future work will focus on considering negative rewards, optimisations with respect to time and reward-bounded reachability properties, as well as the handling of several rewards as multi-optimisation problems.

Acknowledgement. This work has been supported by the NWO project SYRUP (612.063.817), by the STW-ProRail partnership program ExploRail under the project ArRangeer (12238), by the DFG/NWO bilateral project ROCKS (DN

63-257), by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS), and by the European Union Seventh Framework Programmes under grant agreement no. 295261 (MEALS), 318490 (SENSATION) and 318003 (TREsPASS). We would like to thank Joost-Pieter Katoen for the fruitful discussions.

References

1. S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In *FORMATS*, volume 2791 of *LNCS*, pages 88–104. Springer, 2003.
2. R. Bamberg. Non-deterministic generalised stochastic Petri nets modelling and analysis. Master's thesis, University of Twente, 2012.
3. M. Bernardo. An algebra-based method to associate rewards with EMPA terms. In *ICALP*, volume 1256 of *LNCS*, pages 358–368. Springer, 1997.
4. H. Boudali, P. Crouzen, and M. I. A. Stoelinga. A rigorous, compositional, and extensible framework for dynamic fault tree analysis. *IEEE Transactions on Dependable and Secure Computing*, 7(2):128–143, 2010.
5. M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri. Safety, dependability and performance analysis of extended AADL models. *The Computer Journal*, 54(5):754–775, 2011.
6. B. Braitling, L. M. F. Fioriti, H. Hatefi, R. Wimmer, B. Becker, and H. Hermanns. MeGARA: Menu-based game abstraction and abstraction refinement of Markov automata. In *QAPL*, volume 154 of *EPTCS*, pages 48–63, 2014.
7. T. Brazdil, V. Brozek, K. Chatterjee, V. Forejt, and A. Kucera. Two views on multiple mean-payoff objectives in Markov decision processes. In *LICS*, pages 33–42. IEEE, 2011.
8. K. Chatterjee and M. Henzinger. Faster and dynamic algorithms for maximal end-component decomposition and related graph problems in probabilistic verification. In *SODA*, pages 1318–1336. SIAM, 2011.
9. G. Clark. Formalising the specification of rewards with PEPA. In *PAPM*, pages 139–160, 1996.
10. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, 1997.
11. Y. Deng and M. Hennessy. Compositional reasoning for weighted Markov decision processes. *Science of Computer Programming*, 78(12):2537 – 2579, 2013. Special Section on International Software Product Line Conference 2010 and Fundamentals of Software Engineering (selected papers of FSEN 2011).
12. Y. Deng and M. Hennessy. On the semantics of Markov automata. *Information and Computation*, 222:139–168, 2013.
13. C. Eisentraut, H. Hermanns, J.-P. Katoen, and L. Zhang. A semantics for every GSPN. In *ICATPN*, volume 7927 of *LNCS*, pages 90–109. Springer, 2013.
14. C. Eisentraut, H. Hermanns, and L. Zhang. Concurrency and composition in a stochastic world. In *CONCUR*, volume 6269 of *LNCS*, pages 21–39. Springer, 2010.
15. C. Eisentraut, H. Hermanns, and L. Zhang. On probabilistic automata in continuous time. In *LICS*, pages 342–351. IEEE, 2010.
16. J. F. Groote and A. Ponse. The syntax and semantics of μCRL. In *ACP*, Workshops in Computing, pages 26–62. Springer, 1995.
17. D. Guck, T. Han, J.-P. Katoen, and M. R. Neuhäußer. Quantitative timed analysis of interactive Markov chains. In *NFM*, volume 7226 of *LNCS*, pages 8–23. Springer, 2012.

18. D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. Modelling, reduction and analysis of Markov automata. In *QEST*, volume 8054 of *LNCS*, pages 55–71. Springer, 2013.
19. D. Guck, M. Timmer, H. Hatefi, E. J. J. Ruijters, and M. I. A. Stoelinga. Modelling and analysis of Markov reward automata (extended version). Technical Report TR-CTIT-14-06, CTIT, University of Twente, Enschede, 2014.
20. H. Hatefi and H. Hermanns. Model checking algorithms for Markov automata. *Electronic Communications of the EASST*, 53, 2012.
21. B. R. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking performability properties. In *DSN*, pages 103–112. IEEE, 2002.
22. B. R. Haverkort, H. Hermanns, and J.-P. Katoen. On the use of model checking techniques for dependability evaluation. In *SRDS*, pages 228–237. IEEE, 2000.
23. H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *LNCS*. Springer, 2002.
24. J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Performance Evaluation*, 68(2):90–104, 2011.
25. M. R. Neuhäußer. *Model Checking Nondeterministic and Randomly Timed Systems*. PhD thesis, University of Twente, 2010.
26. M. R. Neuhäußer, M. I. A. Stoelinga, and J.-P. Katoen. Delayed nondeterminism in continuous-time Markov decision processes. In *FOSSACS*, volume 5504 of *LNCS*, pages 364–379. Springer, 2009.
27. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.
28. L. Song, L. Zhang, and J. C. Godskesen. Late weak bisimulation for Markov automata. Technical report, ArXiv e-prints, 2012.
29. M. M. Srinivasan. Nondeterministic polling systems. *Management Science*, 37(6):667–681, 1991.
30. M. Timmer. SCOOP: A tool for symbolic optimisations of probabilistic processes. In *QEST*, pages 149–150. IEEE, 2011.
31. M. Timmer. *Efficient Modelling, Generation and Analysis of Markov Automata*. PhD thesis, University of Twente, 2013.
32. M. Timmer, J.-P. Katoen, J. C. van de Pol, and M. I. A. Stoelinga. Efficient modelling and generation of Markov automata. In *CONCUR*, volume 7454 of *LNCS*, pages 364–379. Springer, 2012.
33. M. Timmer, M. I. A. Stoelinga, and J. C. van de Pol. Confluence reduction for Markov automata. In *FORMATS*, volume 8053 of *LNCS*, pages 243–257. Springer, 2013.
34. J. C. van de Pol and M. Timmer. State space reduction of linear processes using control flow reconstruction. In *ATVA*, volume 5799 of *LNCS*, pages 54–68. Springer, 2009.
35. R. Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996.