Efficient Modelling and Generation of Markov Automata*

Mark Timmer[1], Joost-Pieter Katoen[1,2], Jaco van de Pol[1], and Mariëlle Stoelinga[1]

[1] Formal Methods and Tools, Faculty of EEMCS
University of Twente, The Netherlands
{timmer,vdpol,m.i.a.stoelinga}@cs.utwente.nl
[2] Software Modeling and Verification Group
RWTH Aachen University, Germany
katoen@cs.rwth-aachen.de

Abstract. This paper introduces a framework for the efficient modelling and generation of Markov automata. It consists of (1) the data-rich process-algebraic language MAPA, allowing concise modelling of systems with nondeterminism, probability and Markovian timing; (2) a restricted form of the language, the MLPPE, enabling easy state space generation and parallel composition; and (3) several syntactic reduction techniques on the MLPPE format, for generating equivalent but smaller models.

Technically, the framework relies on an encoding of MAPA into the existing prCRL language for probabilistic automata. First, we identify a class of transformations on prCRL that can be lifted to the Markovian realm using our encoding. Then, we employ this result to reuse prCRL's linearisation procedure to transform any MAPA specification to an equivalent MLPPE, and to lift three prCRL reduction techniques to MAPA. Additionally, we define two novel reduction techniques for MLPPEs. All our techniques treat data as well as Markovian and interactive behaviour in a fully symbolic manner, working on specifications instead of models and thus reducing state spaces prior to their construction. The framework has been implemented in our tool SCOOP, and a case study on polling systems and mutual exclusion protocols shows its practical applicability.

1 Introduction

In the past decade, much research has been devoted to improving the efficiency of probabilistic model checking: verifying properties on systems that are governed by, in general, both probabilistic and nondeterministic choices. This way, many models in areas like distributed systems, networking, security and systems biology have been successfully used for dependability and performance analysis.

Recently, a new type of model that captures much richer behaviour was introduced: Markov automata (MAs) [5, 4, 3]. In addition to nondeterministic and

* This research has been partially funded by NWO under grants 612.063.817 (SYRUP) and Dn 63-257 (ROCKS).

probabilistic choices, MAs also contain Markovian transitions, i.e., transitions subject to an exponentially distributed delay. Hence, MAs can be seen as a unification of probabilistic automata (PAs) [16, 18] (containing nondeterministic and probabilistic transitions) and interactive Markov chains (IMCs) [8] (containing nondeterministic and Markovian transitions). They provide a natural semantics for a wide variety of specification languages for concurrent systems, including Generalized Stochastic Petri Nets [12], the domain-specific language AADL [2] and (dynamic) fault trees [1]; i.e., MAs are very general and, except for hard real-time deadlines, can describe most behaviour that is modelled today.

Example 1. Figure 1 shows the state space of a polling system with two arrival stations and probabilistically erroneous behaviour (inspired by [17]). Although probability can sometimes be encoded in rates (e.g., having $(0,0,0) \xrightarrow{0.1\lambda_1} (1,0,1)$ and $(0,0,0) \xrightarrow{0.9\lambda_1} (0,0,1)$ instead of the current λ_1-transition from $(0,0,0)$ and the τ-transition from $(1,0,0)$), the transitions leaving $(1,1,0)$ cannot be encoded like that, due to the nondeterminism between them. Thus, this system could not be represented by an IMC (and neither a PA, due to the Markovian rates). □

Although several formalisms to specify PAs and IMCs exist [10, 6], no data-rich specification language for MAs has been introduced so far. Since realistic systems often consist of a very large number of states, such a method to model systems on a higher level, instead of explicitly providing the state space, is vital. Additionally, the omnipresent state space explosion also applies to MAs. Therefore, high-level specifications are an essential starting point for syntactic optimisations that aim to reduce the size of the state spaces to be constructed.

Our approach. We introduce a new process-algebraic specification language for MAs, called MAPA (Markov Automata Process Algebra). It is based on the prCRL language for PAs [10], which was in turn based on μCRL [7]. MAPA supports the use of data for efficient modelling in the presence of nondeterministic

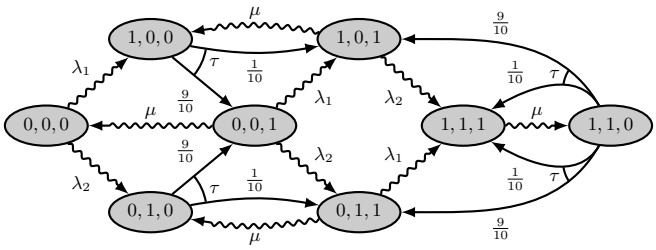

Fig. 1. A queueing system, consisting of a server and two stations. The two stations have incoming requests with rates λ_1, λ_2, which are stored until fetched by the server. If both stations contain a job, the server chooses nondeterministically. Jobs are processed with rate μ, and when polling a station, there is a $\frac{1}{10}$ probability that the job is erroneously kept in the station after being fetched. Each state is represented as a tuple (s_1, s_2, j), with s_i the number of jobs in station i, and j the number of jobs in the server. For simplicity we assume that each component can hold at most one job.

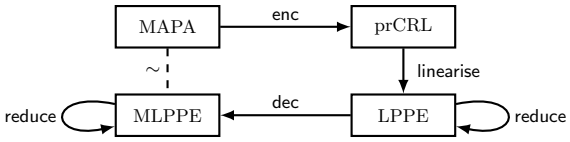

Fig. 2. Linearising MAPA specifications using prCRL linerarisation.

and probabilistic choices, as well as Markovian delays. We define a normal form for MAPA: the Markovian Linear Probabilistic Process Equation (MLPPE). Like the LPPE for prCRL, it allows for easy state space generation and parallel composition, and simplifies the definition of syntactic reduction techniques. These reduce the MA underlying a MAPA specification prior to its generation.

We present an encoding of MAPA into prCRL, to exploit many useful results from the prCRL context. This is non-trivial, since strong bisimulation (or even isomorphism) of PAs does not guarantee bisimulation of the MAs obtained after decoding. Therefore, we introduce a notion of bisimulation on prCRL terms, based on the preservation of derivations. We show that, for any prCRL transformation f that respects our *derivation-preserving bisimulation*, $\mathsf{dec} \circ f \circ \mathsf{enc}$ preserves strong bisimulation, i.e., $\mathsf{dec}\,(f(\mathsf{enc}\,(M)))$ is strongly bisimilar to M for every MAPA specification M. This implies that many useful prCRL transformations are directly applicable to MAPA specifications. We show that this is the case for the linearisation procedure of [10]; as a result, we can reuse it to transform any MAPA specifications to an equivalent MLPPE. We show that three previously defined reduction techniques also respect derivation-preserving bisimulation. Hence, they can now be applied to Markovian models as well. Moreover, we describe two novel reduction techniques for MLPPEs. We implemented the complete framework in our tool SCOOP [21], and show its applicability using the aforementioned polling system and a probabilistic mutual exclusion protocol.

Figure 2 summarises the procedure of encoding a specification into prCRL, linearising, reducing, decoding, and possibly reducing some more, obtaining an efficient MLPPE that is strongly bisimilar to the original specification. Since MAs generalise many existing formalisms (LTSs, DTMCs, CTMCs, IMCs, PAs), we can just as well use MAPA and all our reduction techniques on such models. Thus, this paper provides an overarching framework for efficiently modelling and optimising specifications for all of these models.

Overview of the paper. We introduce the preliminaries of MAs in Section 2, and the language MAPA in Section 3. The encoding in prCRL, as well as linearisation, is dealt with in Section 4. Then, Section 5 presents various reductions techniques, which are applied to a case study in Section 6. The paper is concluded in Section 7. Due to space limitations, we refer to [19] for the (straightforward) definition of parallel composition and all complete proofs.

Acknowledgements. We thank Erik de Vink for his many helpful comments on an earlier draft of this paper, as well as Pedro d'Argenio for his useful insights.

2 Preliminaries

Definition 1 (Basics). *Given a set S, an element $s \in S$ and a sequence $\sigma = \langle s_1, s_2, \ldots, s_n \rangle \in S^*$, we use $s + \sigma$ to denote $\langle s, s_1, s_2, \ldots, s_n \rangle$.*

A probability distribution *over a countable set S is a function $\mu \colon S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. We denote by $\mathsf{Distr}(S)$ the sets of all such functions. For $S' \subseteq S$, let $\mu(S') = \sum_{s \in S'} \mu(s)$. We define the lifting $\mu_f \in \mathsf{Distr}(T)$ of μ over a function $f \colon S \to T$ by $\mu_f(t) = \mu(f^{-1}(t))$. Note that, for injective f, $\mu_f(f(s)) = \mu(s)$ for every $s \in S$. We let $\mathsf{supp}(\mu) = \{s \in S \mid \mu(s) > 0\}$ be the* support *of μ, and write $\mathbb{1}_s$ for the* Dirac distribution *for s, determined by $\mathbb{1}_s(s) = 1$.*

Given an equivalence relation $R \subseteq S \times S$, we write $[s]_R$ for the equivalence class induced by s, i.e., $[s]_R = \{s' \in S \mid (s, s') \in R\}$. We denote the set of all such equivalence classes by S/R. Given two probability distributions μ, μ' over S, we write $\mu \equiv_R \mu'$ to denote that $\mu([s]_R) = \mu'([s]_R)$ for every $s \in S$.

An MA is a transition system in which the set of transitions is partitioned into interactive transitions (which are equivalent to the transitions of a PA) and Markovian transitions (which are equivalent to the transitions of an IMC). The following definition formalises this, and provides notations for MAs. We assume a countable universe *Act* of actions, with $\tau \in \mathit{Act}$ 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*;*
- *$A \subseteq \mathit{Act}$ is a countable set of* actions*;*
- *$\hookrightarrow \subseteq S \times A \times \mathsf{Distr}(S)$ is the* interactive transition relation*;*
- *$\rightsquigarrow \subseteq S \times \mathbb{R}_{>0} \times S$ is the* Markovian transition relation*.*

If $(s, a, \mu) \in \hookrightarrow$, we write $s \xhookrightarrow{a} \mu$ and say that the action a can be executed *from state s, after which the probability to go to $s' \in S$ 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.*

The rate between two states $s, s' \in S$ *is $\mathit{rate}(s, s') = \sum_{(s,\lambda,s') \in \rightsquigarrow} \lambda$, and the* outgoing rate of s *is $\mathit{rate}(s) = \sum_{s' \in S} \mathit{rate}(s, s')$. We require $\mathit{rate}(s) < \infty$ for every state $s \in S$. If $\mathit{rate}(s) > 0$, the* branching probability distribution *after this delay is denoted by $\mathbb{P}_s$ and defined by $\mathbb{P}_s(s') = \frac{\mathit{rate}(s,s')}{\mathit{rate}(s)}$ for every $s' \in S$.*

Remark 1. As we focus on data with possibly infinite domains, we need countable state spaces. Although this is problematic for weak bisimulation [5], it does not hinder us since we only depend on strong bisimulation.

We do need a finite exit rate for every state. After all, given a state s with $\mathit{rate}(s) = \infty$, there is no obvious measure for the next state distribution of s. Also, if all states reachable from s would be considered equivalent by a bisimulation relation, the bisimulation quotient would be ill-defined as it would yield a Markovian transition with rate ∞ (which is not allowed). Fortunately, restricting to finite exit rates is no severe limitation; it still allows infinite chains of states connected by finite rates, as often seen in the context of queueing systems. Also, it still allows infinite branching with for instance rates $\frac{1}{2}\lambda$, $\frac{1}{4}\lambda$, $\frac{1}{8}\lambda$, ... □

Following [5], we define a special action $\chi(r)$ to denote a delay with rate r, enabling a uniform treatment of interactive and Markovian transitions via *extended actions*. As usual [8, 5], we employ the *maximal progress assumption*: time is only allowed to progress in states without outgoing τ-transitions (since they are assumed to be infinitely fast). This is taken into account by only having extended actions representing Markovian delay from states that do not enable an interactive transition $s \xhookrightarrow{\tau} \mu'$.

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 $A^\chi = A \cup \{\chi(r) \mid r \in \mathbb{R}_{>0}\}$. 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*
- *$\alpha = \chi(\mathit{rate}(s))$, $\mathit{rate}(s) > 0$, $\mu = \mathbb{P}_s$ and there is no μ' such that $s \xhookrightarrow{\tau} \mu'$.*

Based on extended actions, we introduce strong bisimulation and isomorphism.

Definition 4 (Strong bisimulation). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be an MA, then an equivalence relation $R \subseteq S \times S$ is a* strong bisimulation *if for every pair $(s, s') \in R$, action $a \in A^\chi$ and transition $s \xrightarrow{a} \mu$, there is a μ' such that $s' \xrightarrow{a} \mu'$ and $\mu \equiv_R \mu'$.*

Two states $s, t \in S$ are strongly bisimilar (denoted by $s \sim t$) if there exists a bisimulation relation R such that $(s, t) \in R$. Two MAs $\mathcal{M}_1, \mathcal{M}_2$ are strongly bisimilar (denoted $\mathcal{M}_1 \sim \mathcal{M}_2$) if their initial states are strongly bisimilar in their disjoint union.

Definition 5 (Isomorphism). *Let $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be an MA, then two states $s, s' \in S$ are* isomorphic *(denoted by $s \cong s'$) if there exists a bijection $f \colon S \to S$ such that $f(s) = s'$ and $\forall t \in S, \mu \in \mathsf{Distr}(S), a \in A^\chi \;.\; t \xrightarrow{a} \mu \Leftrightarrow f(t) \xrightarrow{a} \mu_f$. Two MAs $\mathcal{M}_1, \mathcal{M}_2$ are isomorphic (denoted $\mathcal{M}_1 \cong \mathcal{M}_2$) if their initial states are isomorphic in their disjoint union.*

Obviously, isomorphism implies strong probabilistic bisimulation, as the reflexive and symmetric closure of $\{(s, f(s)) \mid s \in S\}$ is a bisimulation relation.

MAs generalise many classes of systems. Most importantly for this paper, they generalise Segala's PAs [16].

Definition 6 (Probabilistic automata). *A* probabilistic automaton *(PA) is an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow \rangle$ without any Markovian transitions, i.e., $\rightsquigarrow = \emptyset$.*

The definitions of strong bisimulation and isomorphism for MAs correspond to those for PAs, if the MA only contains interactive transitions. So, if two PAs are strongly bisimilar or isomorphic, so are their corresponding MA representations. Therefore, we use the same notations for strong bisimulation and isomorphism of PAs as we do for MAs.

Additionally, we can obtain IMCs by restricting to Dirac distributions for the interactive transitions, CTMCs by taking $\hookrightarrow = \emptyset$, DTMCs by taking $\rightsquigarrow = \emptyset$ and having only one transition $(s, a, \mu) \in \hookrightarrow$ for every $s \in S$, and LTSs by taking $\rightsquigarrow = \emptyset$ and using only Dirac distributions for the interactive transitions [4]. Hence, the results in this paper can be applied to all these models.

3 Markov Automata Process Algebra

We introduce Markov Automata Process Algebra (MAPA), a language in which all conditions, nondeterministic and probabilistic choices, and Markovian delays may depend on data parameters. We assume an external mechanism for the evaluation of expressions (e.g., equational logic, or a fixed data language), able to handle at least boolean and real-valued expressions. Also, we assume that any expression that does not contain variables can be evaluated. Note that this restricts the expressiveness of the data language. In the examples we use an intuitive data language, containing basic arithmetic and boolean operators.

We generally refer to data types with upper-case letters $D, E, \ldots$, and to variables with lower-case letters $u, v, \ldots$.

Definition 7 (Process terms). *A* process term *in MAPA is any term that can be generated by the following grammar:*

$$p ::= Y(\boldsymbol{t}) \;\mid\; c \Rightarrow p \;\mid\; p + p \;\mid\; \textstyle\sum_{\boldsymbol{x}:\boldsymbol{D}} p \;\mid\; a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p \;\mid\; (\lambda) \cdot p$$

Here, Y is a process name, $\boldsymbol{t}$ a vector of expressions, c a boolean expression, $\boldsymbol{x}$ a vector of variables ranging over a (possibly infinite) type $\boldsymbol{D}$, $a \in Act$ a (parameterised) atomic action, f a real-valued expression yielding values in $[0, 1]$, and λ an expression yielding positive real numbers (rates). We write $p = p'$ for syntactically identical process terms. Note that, if $|\boldsymbol{x}| > 1$, $\boldsymbol{D}$ is a Cartesian product, as for instance in $\sum_{(m,i):\{m_1,m_2\}\times\{1,2,3\}} \mathrm{send}(m, i) \ldots$.

Given an expression t, a process term p and two vectors $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{d} = (d_1, \ldots, d_n)$, we use $t[\boldsymbol{x} := \boldsymbol{d}]$ to denote the result of substituting every x_i in t by d_i, and $p[\boldsymbol{x} := \boldsymbol{d}]$ for the result of applying this to every expression in p.

In a process term, $Y(\boldsymbol{t})$ denotes *process instantiation*, where $\boldsymbol{t}$ instantiates Y's process variables as defined below (allowing recursion). The term $c \Rightarrow p$ behaves as p if the *condition* c holds, and cannot do anything otherwise. The $+$ operator denotes *nondeterministic choice*, and $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$ a (possibly infinite) *nondeterministic choice over data type* $\boldsymbol{D}$. The term $a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p$ performs the action $a(\boldsymbol{t})$ and then does a *probabilistic choice* over $\boldsymbol{D}$. It uses the value $f[\boldsymbol{x} := \boldsymbol{d}]$ as the probability of choosing each $\boldsymbol{d} \in \boldsymbol{D}$. Finally, $(\lambda) \cdot p$ can behave as p after a delay, determined by a negative exponential distribution with rate λ.

Definition 8 (Specifications). *A* MAPA specification *is given by a tuple* $M = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i\}, X_j(\boldsymbol{t}))$ *consisting of a set of uniquely-named* processes X_i*, each defined by a* process equation $X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i$*, and an* initial process $X_j(\boldsymbol{t})$*. In a process equation,* $\boldsymbol{x_i}$ *is a vector of* process variables *with type* $\boldsymbol{D_i}$*, and* p_i *(the* right-hand side*) is a* process term *specifying the behaviour of* X_i*.*

A variable v *in an expression in a right-hand side* p_i *is* bound *if it is an element of* $\boldsymbol{x_i}$ *or it occurs within a construct* $\sum_{\boldsymbol{x}:\boldsymbol{D}}$ *or* $\sum_{\boldsymbol{x}:\boldsymbol{D}}$ *such that* v *is an element of* $\boldsymbol{x}$*. Variables that are not bound are said to be* free*.*

A prCRL specification [10] is a MAPA specification without rates.

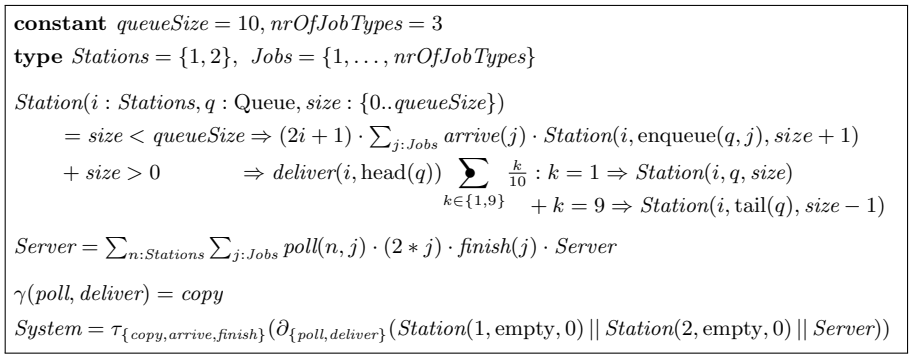

Fig. 3. Specification of a polling system.

We generally refer to process terms with lower-case letters p, q, r, and to processes with capitals X, Y, Z. Also, we will often write $X(x_1 : D_1, \ldots, x_n : D_n)$ for $X((x_1, \ldots, x_n) : (D_1 \times \cdots \times D_n))$. The syntactic sugar introduced for prCRL [10] can be lifted directly to MAPA. Most importantly, we write $a(\boldsymbol{t}) \cdot p$ for the action $a(\boldsymbol{t})$ that goes to p with probability 1.

Parallel composition. Using MAPA processes as basic building blocks, we support the modular construction of large systems via top-level parallelism, encapsulation, hiding, and renaming. This can be defined straightforwardly [19].

Example 2. Figure 3 shows the specification for a slightly more involved variant of the system explained in Example 1. Instead of having just one type of job, as was the case there, we now allow a number of different kinds of jobs (with different service rates). Also, we allow the stations to have larger buffers.

The specification uses three data types: a set *Stations* with identifiers for the two stations, a set *Jobs* with the possible incoming jobs, and a built-in type *Queue*. The arrival rate for station i is set to $2i + 1$, so in terms of the rates in Figure 1 we have $\lambda_1 = 3$ and $\lambda_2 = 5$. Each job j is served with rate $2j$.

The stations receive jobs if their queue is not full, and are able to deliver jobs if their queue is not empty. As explained before, removal of jobs from the queue fails with probability $\frac{1}{10}$. The server continuously polls the stations and works on their jobs. The system is composed of the server and two stations, communicating via the *poll* and *deliver* actions. □

3.1 Static and operational semantics

Not all syntactically correct MAPA specifications are meaningful. The following definition formulates additional well-formedness conditions. The first two constraints ensure that a specification does not refer to undefined variables or processes, the third is needed to obtain valid probability distributions, and the fourth ensures that the specification has a unique solution (modulo strong probabilistic bisimulation). Additionally, all exit rates should be finite. This is discussed in Remark 2, after providing the operational semantics and MLPPE format.

To define well-formedness, we require the concept of *unguardedness*. We say that a process term $Y(\boldsymbol{t})$ can go *unguarded* to Y. Moreover, $c \Rightarrow p$ can go unguarded to Y if p can, $p+q$ if either p or q can, and $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$ if p can, whereas $a(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : p$ and $(\lambda) \cdot p$ cannot go unguarded anywhere.

Definition 9 (Well-formed). *A MAPA specification* $M = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i\}, X_j(\boldsymbol{t}))$ *is* well-formed *if the following four constraints are all satisfied:*

- *There are no free variables.*
- *For every instantiation* $Y(\boldsymbol{t}')$ *occurring in some* p_i*, there exists a process equation* $(X_k(\boldsymbol{x_k} : \boldsymbol{D_k}) = p_k) \in M$ *such that* $X_k = Y$ *and* $\boldsymbol{t}'$ *is of type* $\boldsymbol{D_k}$*. Also, the vector* $\boldsymbol{t}$ *used in the initial process is of type* $\boldsymbol{D_j}$*.*
- *For every construct* $a(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : p$ *occurring in a right-hand side* p_i *it holds that* $\sum_{\boldsymbol{d}\in\boldsymbol{D}} f[\boldsymbol{x} := \boldsymbol{d}] = 1$ *for every possible valuation of the free variables in* $f[\boldsymbol{x} := \boldsymbol{d}]$ *(the summation now used in the mathematical sense).*
- *For every process* Y*, there is no sequence of processes* $X_1, X_2, \ldots, X_n$ *(with* $n \geq 2$*) such that* $Y = X_1 = X_n$ *and every* p_j *can go unguarded to* p_{j+1}*.*

We assume from now on that every MAPA specification is well-formed.

The operational semantics of well-formed MAPA is given by an MA, based on the SOS rules in Figure 4. These rules provide derivations for process terms, like for classical process algebras, but additionally keep track of the rules used in a derivation. A mapping to MAs is only provided for process terms without free variables; this is consistent with our notion of well-formedness. Note that, without the new MSTEP rule, the semantics corresponds precisely to prCRL [10].

Definition 10 (Derivations). *An* α-derivation *from* p *to* β *is a sequence of SOS rules* $\mathcal{D}$ *such that* $p \xrightarrow{\alpha}_{\mathcal{D}} \beta$*. We denote the set of all derivations by* Δ*, and the set of* Markovian derivations *from* p *to* p' *by*

$$\mathsf{MD}(p, p') = \{(\lambda, \mathcal{D}) \in \mathbb{R} \times \Delta \mid p \xrightarrow{\lambda}_{\mathcal{D}} p', \text{MSTEP} \in \mathcal{D}\}.$$

Note that NSUM is instantiated with a data element to distinguish between, for instance, $\sum_{d:\{1,2\}} a(d) \cdot p \xrightarrow{a(d_1)}_{\text{NSUM}(d_1)} p$ and $\sum_{d:\{1,2\}} a(d) \cdot p \xrightarrow{a(d_2)}_{\text{NSUM}(d_2)} p$.

$$\text{INST}\ \frac{p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\alpha}_{\mathcal{D}} \beta}{Y(\boldsymbol{d}) \xrightarrow{\alpha}_{\text{INST}+\mathcal{D}} \beta}\ \text{if } Y(\boldsymbol{x} : \boldsymbol{D}) = p \qquad \text{IMPLIES}\ \frac{p \xrightarrow{\alpha}_{\mathcal{D}} \beta}{c \Rightarrow p \xrightarrow{\alpha}_{\text{IMPLIES}+\mathcal{D}} \beta}\ \text{if } c \text{ holds}$$

$$\text{NCHOICEL}\ \frac{p \xrightarrow{\alpha}_{\mathcal{D}} \beta}{p+q \xrightarrow{\alpha}_{\text{NCHOICEL}+\mathcal{D}} \beta} \qquad \text{NCHOICER}\ \frac{q \xrightarrow{\alpha}_{\mathcal{D}} \beta}{p+q \xrightarrow{\alpha}_{\text{NCHOICER}+\mathcal{D}} \beta}$$

$$\text{NSUM}(\boldsymbol{d})\ \frac{p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\alpha}_{\mathcal{D}} \beta}{\sum_{\boldsymbol{x}:\boldsymbol{D}} p \xrightarrow{\alpha}_{\text{NSUM}(d)+\mathcal{D}} \beta}\ \text{if } \boldsymbol{d} \in \boldsymbol{D} \qquad \text{MSTEP}\ \frac{-}{(\lambda) \cdot p \xrightarrow{\lambda}_{\text{MSUM}} p}$$

$$\text{PSUM}\ \frac{-}{a(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : p \xrightarrow{a(\boldsymbol{t})}_{\text{PSUM}} \mu}\ \text{where } \mu(p[\boldsymbol{x} := \boldsymbol{d}]) = \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ p[\boldsymbol{x}:=\boldsymbol{d}] = p[\boldsymbol{x}:=\boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}'], \text{ for every } \boldsymbol{d} \in \boldsymbol{D}$$

Fig. 4. SOS rules for MAPA.

Example 3. Consider $p = (\lambda_1) \cdot q + (\sum_{n:\{1,2,3\}} n < 3 \Rightarrow (\lambda_2) \cdot q)$. We derive

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{-}{(\lambda_2)\cdot q \xrightarrow{\lambda_2}_{\langle \text{MSTEP}\rangle} q}\,\text{MSTEP}}{1<3 \Rightarrow (\lambda_2)\cdot q \xrightarrow{\lambda_2}_{\langle \text{IMPLIES},\text{MSTEP}\rangle} q}\,\text{IMPLIES}}{\sum_{n:\{1,2,3\}} n<3 \Rightarrow (\lambda_2)\cdot q \xrightarrow{\lambda_2}_{\langle \text{NSUM}(1),\text{IMPLIES},\text{MSTEP}\rangle} q}\,\text{NSUM}(1)}{(\lambda_1)\cdot q + \sum_{n:\{1,2,3\}} n<3 \Rightarrow (\lambda_2)\cdot q \xrightarrow{\lambda_2}_{\langle \text{NCHOICER},\text{NSUM}(1),\text{IMPLIES},\text{MSTEP}\rangle} q}\,\text{NCHOICER}$$

So, $p \xrightarrow{\lambda_2}_{\mathcal{D}} q$ with $\mathcal{D} = \langle \text{NCHOICER}, \text{NSUM}(1), \text{IMPLIES}, \text{MSTEP}\rangle$. Similarly, we can find one other derivation $\mathcal{D}'$ with rate λ_2 using NSUM(2), and finally $p \xrightarrow{\lambda_1}_{\mathcal{D}''} q$ with $\mathcal{D}'' = \langle \text{NCHOICEL}, \text{MSTEP}\rangle$. Since these are the only derivations from p to q, we find $\mathsf{MD}(p,q) = \{(\lambda_2, \mathcal{D}), (\lambda_2, \mathcal{D}'), (\lambda_1, \mathcal{D}'')\}$. □

Definition 11 (Operational semantics). *The semantics of a MAPA specification $M = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i\}, X_j(\boldsymbol{t}))$ is an MA $\mathcal{M} = \langle S, s^0, A, \hookrightarrow, \rightsquigarrow\rangle$, where*

- *S is the set of all MAPA process terms without free variables, and $s^0 = X_j(\boldsymbol{t})$;*
- *$A = \{a(\boldsymbol{t}) \mid a \in Act, \boldsymbol{t}$ is a vector of expressions without free variables$\}$*
- *$\hookrightarrow$ is the smallest relation such that $(p, a, \mu) \in \hookrightarrow$ if $p \xrightarrow{a}_{\mathcal{D}} \mu$ is derivable using the SOS rules in Figure 4 for some $\mathcal{D}$ such that* MSTEP $\notin \mathcal{D}$*;*
- *$\rightsquigarrow$ is the smallest relation such that $(p, \lambda, p') \in \rightsquigarrow$ if $\mathsf{MD}(p, p') \neq \emptyset$ and $\lambda = \sum_{(\lambda', \mathcal{D}) \in \mathsf{MD}(p,p')} \lambda'$.*

Note that, for $\rightsquigarrow$, we sum the rates of all Markovian derivations from p to p'. For Example 3, this yields $p \overset{\lambda}{\rightsquigarrow} q$ with $\lambda = \lambda_1 + 2\lambda_2$. Just applying the SOS rules as for $\hookrightarrow$ would yield $(\lambda) \cdot p' + (\lambda) \cdot p' \overset{\lambda}{\rightsquigarrow} p'$. However, as the race between the two exponentially distributed transitions doubles the speed of going to p, we want to obtain $(\lambda) \cdot p' + (\lambda) \cdot p' \overset{2\lambda}{\rightsquigarrow} p'$. This issue has been recognised before, leading to state-to-function transition systems [11], multi-transition systems [9], and derivation-labelled transitions [15]. Our approach is based on the latter.

An appealing implication of the derivation-based semantics is that parallel composition can easily be defined for MAPA: we can do without the extra clause for parallel self-loops that was needed in [5]. See [19] for more details.

Given a MAPA specification M and its underlying MA $\mathcal{M}$, two process terms in M are isomorphic if their corresponding states in $\mathcal{M}$ are isomorphic. Two specifications with underlying MAs $\mathcal{M}_1, \mathcal{M}_2$ are isomorphic if $\mathcal{M}_1$ is isomorphic to $\mathcal{M}_2$. Bisimilar process terms and specifications are defined in the same way.

3.2 Markovian Linear Probabilistic Process Equations

To simplify state space generation and enable reduction techniques, we introduce a normal form for MAPA: the MLPPE. It generalises the LPPE format for prCRL [10], which in turn was based on the LPE format for μCRL [7]. In the LPPE format, there is precisely one process, which consists of a nondeterministic choice between a set of *summands*. Each of these summands potentially contains a nondeterministic choice, followed by a condition, an interactive action and a probabilistic choice that determines the next state. The MLPPE additionally allows summands with a rate instead of an action.

Definition 12 (MLPPEs). *An* MLPPE (Markovian linear probabilistic process equation) *is a MAPA specification of the following format:*

$$X(\boldsymbol{g}:\boldsymbol{G}) = \sum_{i\in I}\sum_{\boldsymbol{d_i}:\boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i}:\boldsymbol{E_i}} f_i : X(\boldsymbol{n_i}) + \sum_{j\in J}\sum_{\boldsymbol{d_j}:\boldsymbol{D_j}} c_j \Rightarrow (\lambda_j)\cdot X(\boldsymbol{n_j})$$

The first $|I|$ nondeterministic choices are referred to as interactive summands*, the last $|J|$ as* Markovian summands*.*

The two outer summations are abbreviations of nondeterministic choices between the summands. The expressions c_i, $\boldsymbol{b_i}$, f_i and $\boldsymbol{n_i}$ may depend on $\boldsymbol{g}$ and $\boldsymbol{d_i}$, and f_i and $\boldsymbol{n_i}$ also on $\boldsymbol{e_i}$. Similarly, c_j, λ_j and $\boldsymbol{n_j}$ may depend on $\boldsymbol{g}$ and $\boldsymbol{d_j}$.

Each state of an MLPPE corresponds to a valuation of its global variables, due to the recursive call immediately after each action or delay. Therefore, every reachable state in the underlying MA can be uniquely identified with one of the vectors $\boldsymbol{g'} \in \boldsymbol{G}$ (with the initial vector identifying the initial state). From the SOS rules, it follows that for all $\boldsymbol{g'} \in \boldsymbol{G}$, there is a transition $\boldsymbol{g'} \xhookrightarrow{a(\boldsymbol{q})} \mu$ if and only if for at least one summand $i \in I$ there is a local choice $\boldsymbol{d'_i} \in \boldsymbol{D_i}$ such that

$$c_i \wedge a_i(\boldsymbol{b_i}) = a(\boldsymbol{q}) \wedge \forall \boldsymbol{e'_i} \in \boldsymbol{E_i} \,.\, \mu(\boldsymbol{n_i}[\boldsymbol{e_i} := \boldsymbol{e'_i}]) = \sum_{\substack{\boldsymbol{e''_i}\in\boldsymbol{E_i} \\ \boldsymbol{n_i}[\boldsymbol{e_i}:=\boldsymbol{e'_i}]=\boldsymbol{n_i}[\boldsymbol{e_i}:=\boldsymbol{e''_i}]}} f_i[\boldsymbol{e_i} := \boldsymbol{e''_i}],$$

where, for readability, the substitution $[(\boldsymbol{g}, \boldsymbol{d_i}) := (\boldsymbol{g'}, \boldsymbol{d'_i})]$ is omitted from c_i, $\boldsymbol{b_i}$, $\boldsymbol{n_i}$ and f_i. Additionally, there is a transition $\boldsymbol{g'} \overset{\lambda}{\rightsquigarrow} \boldsymbol{g''}$ if and only if $\lambda > 0$ and

$$\lambda = \sum_{\substack{(j,\boldsymbol{d'_j})\in J\times\boldsymbol{D_j} \\ c_j[(\boldsymbol{g},\boldsymbol{d_j}):=(\boldsymbol{g'},\boldsymbol{d'_j})]\wedge\boldsymbol{n_j}[(\boldsymbol{g},\boldsymbol{d_j}):=(\boldsymbol{g'},\boldsymbol{d'_j})]=\boldsymbol{g''}}} \lambda_j[(\boldsymbol{g}, \boldsymbol{d_j}) := (\boldsymbol{g'}, \boldsymbol{d'_j})]$$

Remark 2. For the semantics to be an MA with finite outgoing rates, we need $\sum_{p'}\sum_{(\lambda,\mathcal{D})\in\mathsf{MD}(p,p')}\lambda < \infty$ for every process term p. One way of enforcing this syntactically is to require all data types in Markovian summands to be finite. □

4 Encoding in prCRL

To apply MLPPE-based reductions while modelling in the full MAPA language, we need an automated way for transforming MAPA specifications to strongly bisimilar MLPPEs. Instead of defining such a *linearisation* procedure for MAPA, we exploit the existing linearisation procedure for prCRL. That is, we show how to encode a MAPA specification into a prCRL specification and how to decode a MAPA specification from a prCRL specification. That way, we can apply the existing linearisation procedure, as depicted earlier in Figure 2. Additionally, the encoding enables us to immediately apply many other useful prCRL transformations to MAPA specifications. In this section we explain the encoding and decoding procedures, and prove the correctness of our method.

$$
\begin{array}{lll}
\mathsf{enc}\,(Y(\boldsymbol{t})) & = Y(\boldsymbol{t}) & \mathsf{dec}\,(Y(\boldsymbol{t})) = Y(\boldsymbol{t}) \\
\mathsf{enc}\,(c \Rightarrow p) & = c \Rightarrow \mathsf{enc}\,(p) & \mathsf{dec}\,(c \Rightarrow p) = c \Rightarrow \mathsf{dec}\,(p) \\
\mathsf{enc}\,(p + q) & = \mathsf{enc}\,(p) + \mathsf{enc}\,(q) & \mathsf{dec}\,(p + q) = \mathsf{dec}\,(p) + \mathsf{dec}\,(q) \\
\mathsf{enc}\,\left(\sum_{\boldsymbol{x}:\boldsymbol{D}} p\right) & = \sum_{\boldsymbol{x}:\boldsymbol{D}} \mathsf{enc}\,(p) & \mathsf{dec}\,\left(\sum_{\boldsymbol{x}:\boldsymbol{D}} p\right) = \sum_{\boldsymbol{x}:\boldsymbol{D}} \mathsf{dec}\,(p) \\
\mathsf{enc}\,\left(a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p\right) & = a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : \mathsf{enc}\,(p) & \mathsf{dec}\,\left(a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p\right) = a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : \mathsf{dec}\,(p) \\
 & & \quad (a \neq \mathsf{rate}) \\
 & & \\
\mathsf{enc}\,((\lambda) \cdot p) & = \mathsf{rate}(\lambda) \sum_{x:\{*\}} 1 : \mathsf{enc}\,(p) & \quad (x \text{ does not occur in } p) \\
\mathsf{dec}\,(\mathsf{rate}(\lambda) \sum_{\boldsymbol{x}:\{*\}} 1 : p) & = (\lambda) \cdot \mathsf{dec}\,(p) &
\end{array}
$$

Fig. 5. Encoding and decoding rules for process terms.

4.1 Encoding and decoding

The encoding of MAPA terms is straightforward. The $(\lambda)\cdot p$ construct of MAPA is the only one that has to be encoded, since the other constructs all are also present in prCRL. We chose to encode exponential rates by an action $\mathsf{rate}(\lambda)$ (which is assumed not to occur in the original specification). Since actions in prCRL require a probabilistic choice for the next state, we use $\sum_{x:\{*\}} 1 : p$ such that x is not used in p. Here, $\{*\}$ is a singleton set with an arbitrary element. Figure 5 shows the appropriate encoding and decoding functions.

Definition 13 (Encoding). *Given a MAPA specification* $M = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = p_i\}, X_j(\boldsymbol{t}))$ *and a prCRL specification* $P = (\{Y_i(\boldsymbol{y_i} : \boldsymbol{E_i}) = q_i\}, Y_j(\boldsymbol{u}))$*, let*

$$
\begin{aligned}
\mathsf{enc}\,(M) &= (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = \mathsf{enc}\,(p_i)\}, X_j(\boldsymbol{t})) \\
\mathsf{dec}\,(P) &= (\{Y_i(\boldsymbol{y_i} : \boldsymbol{E_i}) = \mathsf{dec}\,(q_i)\}, Y_j(\boldsymbol{u}))
\end{aligned}
$$

where the functions enc *and* dec *for process terms are given in Figure 5.*

Remark 3. It may appear that, given the above encoding and decoding rules, bisimilar prCRL specifications always decode to bisimilar MAPA specifications. However, this is not the case. Consider the bisimilar prCRL terms $\mathsf{rate}(\lambda) \cdot X + \mathsf{rate}(\lambda) \cdot X$ and $\mathsf{rate}(\lambda) \cdot X$. The decodings of these two terms, $(\lambda) \cdot X + (\lambda) \cdot X$ and $(\lambda) \cdot X$, are clearly not bisimilar in the context of MAPA.

An obvious solution may seem to encode each rate by a unique action, yielding $\mathsf{rate}_1(\lambda)\cdot X + \mathsf{rate}_2(\lambda)\cdot X$, preventing the above erroneous reduction. However, this does not work in all occasions either. Take for instance a MAPA specification consisting of two processes $X = Y + Y$ and $Y = (\lambda) \cdot X$. Encoding this to $X = Y + Y$ and $Y = \mathsf{rate}_1(\lambda) \cdot X$ enables the reduction to $X = Y$ and $Y = \mathsf{rate}_1(\lambda) \cdot X$, which is incorrect since it halves the rate of X.

Note that an 'encoding scheme' that does yield bisimilar MAPA specifications for bisimilar prCRL specifications exists. We could generate the complete state space of a MAPA specification, determine the total rate from p to p' for every pair of process terms p, p', and encode each of these as a unique action in the prCRL specification. When decoding, potential copies of this action that may arise when looking at bisimilar specifications can then just be ignored. However, this clearly renders useless the whole idea of reducing a linear specification before generation of the entire state space. □

Derivation-preserving bisimulation. The observations above suggest that we need a stronger notion of bisimulation if we want two bisimilar prCRL specifications to decode to bisimilar MAPA specifications: all bisimilar process terms should have an equal number of $\mathsf{rate}(\lambda)$ derivations to every equivalence class (as given by the bisimulation relation). We formalise this by means of a *derivation-preserving bisimulation*. It is defined on prCRL terms instead of states in a PA.

Definition 14 (Derivation preservation[1]). *Let R be a bisimulation relation over prCRL process terms. Then, R is* derivation preserving *if for every pair $(p, q) \in R$, every equivalence equivalence class $[r]_R$ and every rate λ:*

$$|\{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| = |\{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, q \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}|.$$

Two prCRL terms p, q are derivation-preserving bisimilar*, denoted $p \sim_{\mathrm{dp}} q$, if there exists a derivation-preserving bisimulation relation R such that $(p, q) \in R$.*

The next theorem states that derivation-preserving bisimulation is a congruence for every prCRL operator. The proof can be found in [19].

Theorem 1. *Derivation-preserving bisimulation is a congruence for prCRL.*

Our encoding scheme and notion of derivation-preserving bisimulation allow us to reuse prCRL transformations for MAPA specifications. The next theorem confirms that a function $\mathsf{dec} \circ f \circ \mathsf{enc}\colon \text{MAPA} \to \text{MAPA}$ respects bisimulation if $f\colon \text{prCRL} \to \text{prCRL}$ respects derivation-preserving bisimulation.

Theorem 2. *Let $f\colon$ prCRL $\to$ prCRL such that $f(P) \sim_{\mathrm{dp}} P$ for every prCRL specification P. Then, $\mathsf{dec}\,(f(\mathsf{enc}\,(M))) \sim M$ for every MAPA specification M without any* rate *action.*

Proof (sketch). It can be shown that (a) $m \overset{a}{\hookrightarrow} \mu$ (with $a \neq \mathsf{rate}$) is a transition in an MA if and only if $\mathsf{enc}\,(m) \xrightarrow{a} \mu_{\mathsf{enc}}$, and that (b) every derivation $m \xrightarrow{\lambda}_{\mathcal{D}} m'$ in an MA corresponds one-to-one to a derivation $\mathsf{enc}\,(m) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(m')}$, with $\mathcal{D}'$ obtained from $\mathcal{D}$ by substituting PSum for MStep. Using these two observations, and taking R as the derivation-preserving bisimulation relation for $f(P) \sim_{\mathrm{dp}} P$, it can be shown that $R' = \{(\mathsf{dec}\,(p), \mathsf{dec}\,(q)) \mid (p, q) \in R\}$ is a bisimulation relation, and hence $\mathsf{dec}\,(f(P)) \sim \mathsf{dec}\,(P)$. Taking $P = \mathsf{enc}\,(M)$, and noting that $\mathsf{dec}\,(\mathsf{enc}\,(M)) = M$, the theorem follows. □

We can now state that the linearisation procedure from [10] (here referred to by $\mathsf{linearise}$) can be used to transform a MAPA specification to an MLPPE. Under the observation that a prCRL specification P and its linearisation are derivation-preserving bisimilar (proven in [19]), it is an immediate consequence of Theorem 2. The fact that M' is an MLPPE follows from the proof in [10] that $\mathsf{linearise}(\mathsf{enc}\,(M))$ is an LPPE, and the observation that decoding does not change the structure of a specification.

Theorem 3. *Let M be a MAPA specification without any* rate *action, and let $M' = \mathsf{dec}\,(\mathsf{linearise}(\mathsf{enc}\,(M)))$. Then, $M \sim M'$ and M' is an MLPPE.*

[1] We could even be a bit more liberal (although technically slightly more involved), only requiring equal sums of the λs of all rate-transitions to each equivalence class.

5 Reductions

We discuss three symbolic prCRL reduction techniques that, by Theorem 2, can directly be applied to MAPA specifications. Also, we discuss two new techniques that are specific to MAPA. Note that, since MAs generalise LTSs, CTMCs, DTMCs, PAs and IMCs, all techniques also are applicable to these subclasses.

5.1 Novel reduction techniques

Maximal progress reduction. No Markovian transitions can be taken from states that also allow a τ-transition. Hence, such Markovian transitions (and their target states) can safely be omitted. This maximal progress reduction can be applied during state space generation, but it is more efficient to already do this on the MLPPE level: we can just omit all Markovian summands that are always enabled together with non-Markovian summands. Note that, to detect such scenarios, some heuristics or theorem proving have to be applied, as in [14].

Summation elimination. Summation elimination [10] aims to remove unnecessary summations, transforming $\sum_{d:\mathbb{N}} d = 5 \Rightarrow \mathit{send}(d) \cdot X$ to $\mathit{send}(5) \cdot X$ (as there is only one possible value for d) and $\sum_{d:\{1,2\}} a \cdot X$ to $a \cdot X$ (as the summation variable is not used). This technique would fail for MAPA, as the second transformation changes the number of a-derivations; for $a = \mathsf{rate}(\lambda)$, this would change behaviour. Therefore, we generalise summation elimination to MLPPEs. Interactive summands are handled as before, but for Markovian summands the second kind of reduction is altered. Instead of reducing $\sum_{d:D}(\lambda) \cdot X$ to $(\lambda) \cdot X$, we now reduce to $(|D| \times \lambda) \cdot X$. That way, the total rate to X remains the same.

5.2 Generalisation of existing techniques

Constant elimination [10] detects if a parameter of an LPPE never changes value. Then, the parameter is omitted and every reference to it replaced by its initial value. *Expression simplification* [10] evaluates functions for which all parameters are constants and applies basic laws from logic. These techniques do not change the state space, but improve readability and speed up state space generation. *Dead-variable reduction* [14] additionally reduces the number of states. It takes into account the control flow of an LPPE and tries to detect states in which the value of some data variable is irrelevant. Basically, this is the case if that variable will be overwritten before being used for all possible futures.

It is easy to see that all three techniques are derivation preserving. Hence, by Theorem 2 we can reuse them unchanged for MAPA using $\mathsf{dec}\,(\mathsf{reduce}(\mathsf{enc}\,(M))$.

6 Case Study and Implementation

We extended our tool SCOOP [21], enabling it to handle MAPA. We implemented the encoding scheme, linked it to the original linearisation and derivation-preserving reduction techniques, and implemented the novel reductions. Table 1 shows statistics of the MAs generated from several variations of Figure 3;

	Original				Reduced				
Spec.	States	Trans.	MLPPE	Time	States	Trans.	MLPPE	Time	Red.
queue-3-5	316,058	581,892	15 / 335	87.4	218,714	484,548	8 / 224	20.7	76%
queue-3-6	1,005,699	1,874,138	15 / 335	323.3	670,294	1,538,733	8 / 224	64.7	80%
queue-3-6'	1,005,699	1,874,138	15 / 335	319.5	74	108	5 / 170	0.0	100%
queue-5-2	27,659	47,130	15 / 335	4.3	23,690	43,161	8 / 224	1.9	56%
queue-5-3	1,191,738	2,116,304	15 / 335	235.8	926,746	1,851,312	8 / 224	84.2	64%
queue-5-3'	1,191,738	2,116,304	15 / 335	233.2	170	256	5 / 170	0.0	100%
queue-25-1	3,330	5,256	15 / 335	0.5	3,330	5,256	8 / 224	0.4	20%
queue-100-1	50,805	81,006	15 / 335	8.9	50,805	81,006	8 / 224	6.6	26%
mutex-3-2	17,352	40,200	27 / 3,540	12.3	10,560	25,392	12 / 2,190	4.6	63%
mutex-3-4	129,112	320,136	27 / 3,540	95.8	70,744	169,128	12 / 2,190	30.3	68%
mutex-3-6	425,528	1,137,048	27 / 3,540	330.8	224,000	534,624	12 / 2,190	99.0	70%
mutex-4-1	27,701	80,516	36 / 5,872	33.0	20,025	62,876	16 / 3,632	13.5	59%
mutex-4-2	360,768	1,035,584	36 / 5,872	435.9	218,624	671,328	16 / 3,632	145.5	67%
mutex-4-3	1,711,141	5,015,692	36 / 5,872	2,108.0	958,921	2,923,300	16 / 3,632	644.3	69%
mutex-5-1	294,882	1,051,775	45 / 8,780	549.7	218,717	841,750	20 / 5,430	216.6	61%

Table 1. State space generation using SCOOP on a 2.4 GHz 8 GB Intel Core 2 Duo MacBook (MLPPE in number of parameters / symbols, time in seconds).

`queue-i-j` denotes the variant with buffers of size `i` and `j` types of jobs[2]. The primed specifications were modified to have a single rate for all types of jobs. Therefore, dead-variable reduction detects that the queue contents are irrelevant.

We also modelled a probabilistic mutex exclusion protocol, based on [13]. Each process is in the critical section for an amount of time governed by an exponential rate, depending on a nondeterministically chosen job type. We denote by `mutex-i-j` the variant with `i` processes and `j` types of jobs.

Note that the MLPPE optimisations impact the MA generation time significantly, even for cases without state space reduction. Also note that earlier case studies for prCRL or μCRL would still give the same results; e.g., the results in [14] that showed the benefits of dead-variable reduction are still applicable.

7 Conclusions and Future Work

We introduced a new process-algebraic framework with data, called MAPA, for modelling and generating Markov automata. We defined a special restricted format, the MLPPE, that allows easy state space generation and parallel composition. We showed how MAPA specifications can be encoded in prCRL, an existing language for probabilistic automata. Based on the novel concept of derivation-preservation bisimulation, we proved that many useful prCRL transformations can directly be used on MAPA specifications. This includes a linearisation procedure to turn MAPA processes into strongly bisimilar MLPPEs, and several existing reduction techniques. Also, we introduced two new reduction techniques. A case study demonstrated the use of the framework and the strength of the reduction techniques. Since MAs generalise LTS, DTMCs, CTMCs, IMCs and PAs, we can use MAPA and all our reduction techniques on all such models.

Future work will focus on developing more reduction techniques for MAPA. Most importantly, we will investigate a generalisation of confluence reduction [20].

[2] See `fmt.cs.utwente.nl/~timmer/scoop/papers/concur/` for the tool and models.

References

1. Boudali, H., Crouzen, P., Stoelinga, M.I.A.: Dynamic fault tree analysis using Input/Output interactive Markov chains. In: DSN. pp. 708–717 (2007)
2. Bozzano, M., Cimatti, A., Katoen, J.P., Nguyen, V.Y., Noll, T., Roveri, M.: Safety, dependability and performance analysis of extended AADL models. The Computer Journal 54(5), 754–775 (2011)
3. Deng, Y., Hennessy, M.: On the semantics of Markov automata. In: ICALP. LNCS, vol. 6756, pp. 307–318 (2011)
4. Eisentraut, C., Hermanns, H., Zhang, L.: Concurrency and composition in a stochastic world. In: CONCUR. LNCS, vol. 6269, pp. 21–39 (2010)
5. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS. pp. 342–351 (2010)
6. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2010: A toolbox for the construction and analysis of distributed processes. In: TACAS. LNCS, vol. 6605, pp. 372–387 (2011)
7. Groote, J.F., Ponse, A.: The syntax and semantics of μCRL. In: Algebra of Communicating Processes. pp. 26–62. Workshops in Computing (1995)
8. Hermanns, H.: Interactive Markov Chains: The Quest for Quantified Quality, LNCS, vol. 2428. Springer (2002)
9. Hillston, J.: Process algebras for quantitative analysis. In: LICS. pp. 239–248 (2005)
10. Katoen, J.P., van de Pol, J., Stoelinga, M., Timmer, M.: A linear process-algebraic format with data for probabilistic automata. TCS 413(1), 36–57 (2012)
11. Latella, D., Massink, M., de Vink, E.P.: Bisimulation of labeled state-to-function transition systems of stochastic process languages. In: ACCAT (2012), to appear
12. Marsan, M.A., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Transactions on Computer Systems 2(2), 93–122 (1984)
13. Pnueli, A., Zuck, L.D.: Verification of multiprocess probabilistic protocols. Distributed Computing 1(1), 53–72 (1986)
14. van de Pol, J.C., Timmer, M.: State space reduction of linear processes using control flow reconstruction. In: ATVA. LNCS, vol. 5799, pp. 54–68 (2009)
15. Priami, C.: Stochastic pi-calculus. The Computer Journal 38(7), 578–589 (1995)
16. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. Ph.D. thesis, MIT (1995)
17. Srinivasan, M.M.: Nondeterministic polling systems. Management Science 37(6), 667–681 (1991)
18. Stoelinga, M.I.A.: An introduction to probabilistic automata. Bulletin of the EATCS 78, 176–198 (2002)
19. Timmer, M., Katoen, J.P., van de Pol, J.C., Stoelinga, M.I.A.: Efficient modelling and generation of Markov automata (extended version). Tech. Rep. TR-CTIT-12-16, CTIT, University of Twente (2012)
20. Timmer, M., Stoelinga, M.I.A., van de Pol, J.C.: Confluence reduction for probabilistic systems. In: TACAS. LNCS, vol. 6605, pp. 311–325 (2011)
21. Timmer, M.: SCOOP: A tool for symbolic optimisations of probabilistic processes. In: QEST. pp. 149–150 (2011)