Efficient Modelling and Generation of Markov Automata (extended version)*

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) [6, 5, 4]. 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) [17, 20] (containing nondeterministic and probabilistic transitions) and interactive Markov chains (IMCs) [9] (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 [13], the domain-specific language AADL [3] and (dynamic) fault trees [2]; 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 [18]). 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 [11, 7], 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 [11], which was in turn based on μCRL [8]. MAPA supports the use of data for efficient modelling in the presence of nondeterministic

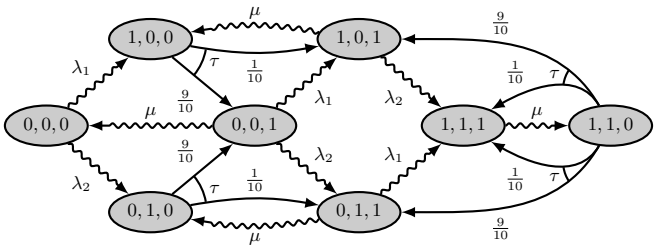

Fig. 1. A queueing system, consisting of a server and two stations. The two stations have incoming requests with rates λ_1, λ_2, which are stored until fetched by the server. If both stations contain a job, the server chooses nondeterministically. Jobs are processed with rate μ, and when polling a station, 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.

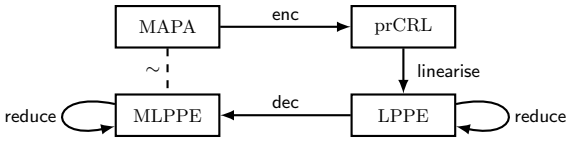

Fig. 2. Linearising MAPA specifications using prCRL linerarisation.

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 [11]; 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 [22], 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. The (straightforward) definition of parallel composition has been placed in Appendix B, and all proofs in Appendix A.

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 [6], 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 [6], 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 [9, 6], 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 [17].

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 [5]. 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 [11] is a MAPA specification without rates.

$$\textbf{constant } \mathit{queueSize} = 10, \mathit{nrOfJobTypes} = 3$$
$$\textbf{type } \mathit{Stations} = \{1, 2\},\ \mathit{Jobs} = \{1, \ldots, \mathit{nrOfJobTypes}\}$$

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

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

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

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 [11] 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. For completeness, we present the technical details in Appendix B.

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 [11].

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*

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

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

$$\textsc{NChoiceL}\ \frac{p \xrightarrow{\alpha}_{\mathcal{D}} \beta}{p + q \xrightarrow{\alpha}_{\textsc{NChoiceL}+\mathcal{D}} \beta} \qquad \textsc{NChoiceR}\ \frac{q \xrightarrow{\alpha}_{\mathcal{D}} \beta}{p + q \xrightarrow{\alpha}_{\textsc{NChoiceR}+\mathcal{D}} \beta}$$

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

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

Fig. 4. SOS rules for MAPA.

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 $\sum_{d:\{1,2\}} a(d) \cdot p \xrightarrow{a(d_2)}_{\text{NSum}(d_2)} p$.

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 [12], multi-transition systems [10], and derivation-labelled transitions [16]. 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 [6]. See Appendix B 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 [11], which in turn was based on the LPE format for μCRL [8]. 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 $\boldsymbol{g'} \xrightarrow{a(\boldsymbol{q})} \mu$ 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,

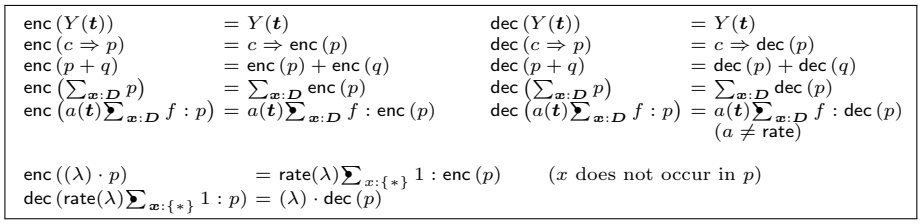

Fig. 5. Encoding and decoding rules for process terms.

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.

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*

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

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 Appendix A.1.

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. The full proof, consisting of several lemmas, can be found in Appendix A.2.

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) $m \hookrightarrow^{a} \mu$ (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 $\mathcal{D}'$

[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.

obtained from $\mathcal{D}$ by substituting PSum for MStep. Using these two observations, and taking R as the derivation-preserving bisimulation relation for $f(P) \sim_{\text{dp}} P$, 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 [11] (here referred to by 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 Appendix A.3), it is an immediate consequence of Theorem 2. The fact that M' is an MLPPE follows from the proof in [11] 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.*

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 [15].

Summation elimination. Summation elimination [11] 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 [11] 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* [11] 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* [15] 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 [22], 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; `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 [14]. 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 [15] 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.

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

	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).

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 [21].

References

1. Bergstra, J.A., Klop, J.W.: ACP$_\tau$: A universal axiom system for process specification. In: Algebraic Methods: Theory, Tools and Applications. LNCS, vol. 394, pp. 447–463 (1989)
2. Boudali, H., Crouzen, P., Stoelinga, M.I.A.: Dynamic fault tree analysis using Input/Output interactive Markov chains. In: DSN. pp. 708–717 (2007)
3. 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)
4. Deng, Y., Hennessy, M.: On the semantics of Markov automata. In: ICALP. LNCS, vol. 6756, pp. 307–318 (2011)
5. Eisentraut, C., Hermanns, H., Zhang, L.: Concurrency and composition in a stochastic world. In: CONCUR. LNCS, vol. 6269, pp. 21–39 (2010)
6. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: LICS. pp. 342–351 (2010)
7. 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)
8. Groote, J.F., Ponse, A.: The syntax and semantics of μCRL. In: Algebra of Communicating Processes. pp. 26–62. Workshops in Computing (1995)
9. Hermanns, H.: Interactive Markov Chains: The Quest for Quantified Quality, LNCS, vol. 2428. Springer (2002)
10. Hillston, J.: Process algebras for quantitative analysis. In: LICS. pp. 239–248 (2005)

11. 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)
12. 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
13. 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)
14. Pnueli, A., Zuck, L.D.: Verification of multiprocess probabilistic protocols. Distributed Computing 1(1), 53–72 (1986)
15. 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)
16. Priami, C.: Stochastic pi-calculus. The Computer Journal 38(7), 578–589 (1995)
17. Segala, R.: Modeling and Verification of Randomized Distributed Real-Time Systems. Ph.D. thesis, MIT (1995)
18. Srinivasan, M.M.: Nondeterministic polling systems. Management Science 37(6), 667–681 (1991)
19. Stoelinga, M.I.A.: Alea jacta est: Verification of Probabilistic, Real-time and Parametric Systems. Ph.D. thesis, University of Nijmegen (2002)
20. Stoelinga, M.I.A.: An introduction to probabilistic automata. Bulletin of the EATCS 78, 176–198 (2002)
21. 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)
22. Timmer, M.: SCOOP: A tool for symbolic optimisations of probabilistic processes. In: QEST. pp. 149–150 (2011)

A Proofs

A.1 Proof of Theorem 1

To prove Theorem 1, we first need the following elementary lemma.

Lemma 1. *Let S be any set and R, R' two equivalence relations over $S \times S$ such that $R \subseteq R'$. Let $[r]_{R'} \in S/R'$ be an arbitrary equivalence class of R'. Then, $[r]_{R'} \in S/R'$ can be partitioned into equivalence classes of R.*

Proof. Let $[p]_R \in S/R$ be one of the equivalence classes of R. We show that either $[p]_R \subseteq [r]_{R'}$ or $[p]_R \cap [r]_{R'} = \emptyset$. To see this, let $s \in [p]_R$, so $(s, p) \in R$. Since $R \subseteq R'$, this implies $(s, p) \in R'$. Now, we make a case distinction based on whether or not $p \in [r]_{R'}$.

- Let $p \in [r]_{R'}$, and hence, $(p, r) \in R'$. Since $(s, p) \in R'$, by transitivity we obtain $(s, r) \in R'$ and thus $s \in [r]_{R'}$.
- Let $p \notin [r]_{R'}$. If $s \in [r]_{R'}$, then $(s, r) \in R'$ and hence by transitivity also $(p, r) \in R'$ and thus $p \in [r]_{R'}$. As this is a contradiction, $s \notin [r]_{R'}$.

Hence, if $p \in [r]_{R'}$ then every element of $[p]_R$ is in $[r]_{R'}$ and thus $[p]_R \subseteq [r]_{R'}$, and if $p \notin [r]_{R'}$ then no element of $[p]_R$ is in $[r]_{R'}$ and thus $[p]_R \cap [r]_{R'} = \emptyset$. Since every equivalence class of R is either fully contained in $[r]_{R'}$ or does not overlap with it at all, $[r]_{R'}$ indeed can be partitioned into equivalence classes of R. □

We now show that derivation-preserving bisimulation is a congruence for all prCRL operators. We allow prCRL process terms to contain free variables. In that case, we require the bisimulation relation to be valid under all possible valuations for these variables. For instance, $d > 5 \Rightarrow p$ and $d > 5 \wedge d > 3 \Rightarrow p$ are clearly bisimilar for every value of d, but $d > 5 \Rightarrow p$ and $d > 3 \Rightarrow p$ are not bisimilar if for example d is substituted by 4.

Theorem 1. *Let $p, p', q,$ and q' be (possibly open) prCRL process terms such that $p \sim_{\mathrm{dp}} p'$ and $q \sim_{\mathrm{dp}} q'$ for every valuation of their free variables. Then, for every such valuation and every $\boldsymbol{D}$, c, a, $\boldsymbol{t}$ and f, also*

$$p + q \sim_{\mathrm{dp}} p' + q' \tag{1}$$

$$\sum_{\boldsymbol{x}:\boldsymbol{D}} p \sim_{\mathrm{dp}} \sum_{\boldsymbol{x}:\boldsymbol{D}} p' \tag{2}$$

$$c \Rightarrow p \sim_{\mathrm{dp}} c \Rightarrow p' \tag{3}$$

$$a(\boldsymbol{t}) \blacktriangleright \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p \sim_{\mathrm{dp}} a(\boldsymbol{t}) \blacktriangleright \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p' \tag{4}$$

$$Y(\boldsymbol{t}) \sim_{\mathrm{dp}} Y'(\boldsymbol{t}) \tag{5}$$

where $Y(\boldsymbol{g} : \boldsymbol{G}) = p$ and $Y'(\boldsymbol{g} : \boldsymbol{G}) = p'$.

Proof. Let R_p and R_q be the derivation-preserving bisimulation relations relating p and p', and q and q', respectively. Also, assume some arbitrary valuation for all free variables of p, p', q and q'.

For each of the statements above, we construct a relation R and show that it is a derivation-preserving bisimulation relation. In each case, we first prove that (a) R is a bisimulation relation relating the left-hand side and right-hand side of the equation, and then that (b) it is derivation preserving.

(1) We choose R to be the symmetric, reflexive, transitive closure of the set

$$R_p \cup R_q \cup \{(p+q, p'+q')\}$$

(a) Let $p+q \xrightarrow{\alpha} \mu$. We show that $p'+q' \xrightarrow{\alpha} \mu'$ such that $\mu \equiv_R \mu'$. By the operational semantics, either $p \xrightarrow{\alpha} \mu$ or $q \xrightarrow{\alpha} \mu$. We assume the first possibility without loss of generality. Since $p \sim_{\mathrm{dp}} p'$ (by the bisimulation relation R_p), we know that $p' \xrightarrow{\alpha} \mu'$ for some μ' such that $\mu \equiv_{R_p} \mu'$, and therefore, by the operational semantics, also $p'+q' \xrightarrow{\alpha} \mu'$. Since $R_p \subseteq R$, by Proposition 5.2.1 of [19] we obtain that $\mu \equiv_R \mu'$. The fact that transitions of $p'+q'$ can be mimicked by $p+q$ follows by symmetry.

For any other element $(s,t) \in R$, the required implications follow from the assumption that R_p and R_q are bisimulation relations. Since R is the smallest set containing R_p, R_q and $(p+q, p'+q')$ such that $(s,s) \in R$, $(s,t) \in R \implies (t,s) \in R$ and $(s,t) \in R \wedge (t,u) \in R \implies (s,u) \in R$, we can do induction over the number of applications of these closure rules for (s,t) to be in R. The base case, $(s,t) \in R_p$ or $(s,t) \in R_q$, follows immediate from the fact that R_p and R_q are bisimulation relations and Proposition 5.2.1 of [19]. Otherwise, $(s,t) \in R$ is due to reflexivity, symmetry or transitivity. For reflexivity, $s = t$, and they trivially mimic each other. For symmetry, $(t,s) \in R$ can mimic each other by the induction hypothesis, and therefore (s,t) also satisfy the requirements because of symmetry of mimicking. If $(s,t) \in R$ since $(s,u) \in R$ and $(u,t) \in R$, then by the induction hypothesis any transition $s \xrightarrow{\alpha} \mu$ can be mimicked by a transition $u \xrightarrow{\alpha} \mu'$ such that $\mu \equiv_R \mu'$, which in turn can be mimicked by a transition $t \xrightarrow{\alpha} \mu''$ such that $\mu' \equiv_R \mu''$. By transitivity of $\equiv_R$, indeed $\mu \equiv_R \mu''$.

(b) Let $[r]_R$ be any equivalence class of R, and λ an arbitrary rate. Also, let

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

be the sets of all derivations from $p+q$ and $p'+q'$, respectively, with action $\mathsf{rate}(\lambda)$ to a state in $[r]_R$. We need to show that $|X| = |X'|$. Note that $|X| < \infty$ and $|X'| < \infty$ since infinite outgoing rates are prohibited.

Note that X can be partitioned into two sets: $X = X_p \cup X_q$, where X_p contains all derivations that start with NChoiceL (and hence correspond to transitions of p), and X_q contains all derivations that start with NChoiceR (corresponding to transitions of q). That is:

$$X_p = \{\mathcal{D} \in \Delta \mid \exists \mathcal{D}' \in \Delta \,.\, \mathcal{D} = \mathrm{NChoiceL} + \mathcal{D}' \wedge \exists r' \in [r]_R \,.\, p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{r'}\}$$
$$X_q = \{\mathcal{D} \in \Delta \mid \exists \mathcal{D}' \in \Delta \,.\, \mathcal{D} = \mathrm{NChoiceR} + \mathcal{D}' \wedge \exists r' \in [r]_R \,.\, q \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{r'}\}$$

Similarly, we can partition X' into two sets X'_p and X'_q. Since every derivation in X_p corresponds to exactly one derivation of p, it follows that the size of X_p is given by

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

and similarly for X_q, X'_p and X'_q.

Since $R_p \subseteq R$, by Lemma 1 we know that $[r]_R$ can be partitioned into $[p_1]_{R_p}, [p_2]_{R_p}, \ldots, [p_n]_{R_p}$ for some $p_1, p_2, \ldots p_n$. Therefore:

$$\begin{aligned} |X_p| &= \sum_{i=1}^{n} |\{\mathcal{D} \in \Delta \mid \exists r' \in [p_i]_{R_p} \,.\, p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| \\ &= \sum_{i=1}^{n} |\{\mathcal{D} \in \Delta \mid \exists r' \in [p_i]_{R_p} \,.\, p' \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| = |X'_p| \end{aligned}$$

where the second equality is due to the fact that $(p, p') \in R_p$ and R_p is derivation preserving. In the same way, we find that $|X_q| = |X'_q|$, and hence $|X| = |X'|$.

The fact that all other elements of R satisfy the derivation preservation property follows from an easy inductive argument and the fact that R_p and R_q are derivation preserving (in the same way as above for (a)).

(2) We choose R to be the symmetric, reflexive, and transitive closure of the set

$$R_p \cup \left\{ \left(\sum_{\boldsymbol{x}:\boldsymbol{D}} p, \sum_{\boldsymbol{x}:\boldsymbol{D}} p' \right) \right\}$$

(a) Let $\sum_{\boldsymbol{x}:\boldsymbol{D}} p \xrightarrow{\alpha} \mu$. Then, by the operational semantics, there is a $\boldsymbol{d} \in \boldsymbol{D}$ such that $p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\alpha} \mu$. From the assumption that $p \sim_{\mathrm{dp}} p'$ for all valuations, it immediately follows that $p[\boldsymbol{x} := \boldsymbol{d}] \sim_{\mathrm{dp}} p'[\boldsymbol{x} := \boldsymbol{d}]$ for any $\boldsymbol{d} \in \boldsymbol{D}$, so if we have $p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\alpha} \mu$, then also $p'[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\alpha} \mu'$ and hence $\sum_{\boldsymbol{x}:\boldsymbol{D}} p' \xrightarrow{\alpha} \mu'$ with $\mu \equiv_{R_p} \mu'$ and thus $\mu \equiv_R \mu'$ due to $R \supseteq R_p$ and Proposition 5.2.1 of [19]. The fact that transitions of $\sum_{\boldsymbol{x}:\boldsymbol{D}} p'$ can be mimicked by $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$ follows by symmetry. For all other elements of R, the required implications follow from the assumption that R_p is a bisimulation relation as above.

(b) Let $[r]_R$ be any equivalence class of R, and λ an arbitrary rate. Also, let

$$\begin{aligned} X &= \{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, \sum_{\boldsymbol{x}:\boldsymbol{D}} p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\} \\ X' &= \{\mathcal{D} \in \Delta \mid \exists r' \in [r]_R \,.\, \sum_{\boldsymbol{x}:\boldsymbol{D}} p \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\} \end{aligned}$$

be the sets of all derivations from $\sum_{\boldsymbol{x}:\boldsymbol{D}} p$ and $\sum_{\boldsymbol{x}:\boldsymbol{D}} p'$, respectively, with action $\mathsf{rate}(\lambda)$ to a state in $[r]_R$. We need to show that $|X| = |X'|$. Again, neither X nor X' can be infinite.

Note that X can be partitioned into as many sets as there are elements in the set $\boldsymbol{D}$: $X = \bigcup_{\boldsymbol{d} \in \boldsymbol{D}} X_{\boldsymbol{d}}$, where $X_{\boldsymbol{d}}$ contains all derivations that start with

$\text{NSUM}(\boldsymbol{d})$ (and hence correspond to transitions of p with $\boldsymbol{d}$ substituted for $\boldsymbol{x}$). That is:

$$X_{\boldsymbol{d}} = \{\mathcal{D} \in \Delta \mid \exists \mathcal{D}' \in \Delta \,.\, \mathcal{D} = \text{NSUM}(\boldsymbol{d}) + \mathcal{D}' \wedge \exists r' \in [r]_R \,.\, p[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{r'}\}$$

Similarly, we can partition X' into sets $X'_{\boldsymbol{d}}$. Since every derivation in $X_{\boldsymbol{d}}$ corresponds precisely to one derivation of $p[\boldsymbol{x} := \boldsymbol{d}]$, it follows that the size of $X_{\boldsymbol{d}}$ is given by

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

and similarly for $X'_{\boldsymbol{d}}$.

Since $R_p \subseteq R$, by Lemma 1 we know that $[r]_R$ can be partitioned into $[p_1]_{R_p}, [p_2]_{R_p}, \ldots, [p_n]_{R_p}$ for some $p_1, p_2, \ldots p_n$. Therefore:

$$\begin{aligned} |X_{\boldsymbol{d}}| &= \sum_{i=1}^{n} |\{\mathcal{D} \in \Delta \mid \exists r' \in [p_i]_{R_p} \,.\, p\,[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| \\ &= \sum_{i=1}^{n} |\{\mathcal{D} \in \Delta \mid \exists r' \in [p_i]_{R_p} \,.\, p'[\boldsymbol{x} := \boldsymbol{d}] \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}} \mathbb{1}_{r'}\}| = |X'_{\boldsymbol{d}}| \end{aligned}$$

where the second equality is due to the fact that $(p, p') \in R_p$ and R_p is derivation preserving for every valuation. As this holds for all $X_{\boldsymbol{d}}$, we obtain $|X| = |X'|$.

The fact that all other elements of R satisfy the derivation preservation property follows again from the fact that R_p is derivation preserving.

(3) We choose R to be the symmetric, reflexive, and transitive closure of the set

$$R_p \cup \{(c \Rightarrow p, c \Rightarrow p')\}$$

(a) Let $(c \Rightarrow p) \xrightarrow{\alpha} \mu$. By the operational semantics, this implies that c holds for the given valuation and $p \xrightarrow{\alpha} \mu$. Now, since $p \sim_{\text{dp}} p'$ (by the bisimulation relation R_p), we know that $p' \xrightarrow{\alpha} \mu'$, and therefore also $(c \Rightarrow p') \xrightarrow{\alpha} \mu'$, such that $\mu \equiv_{R_p} \mu'$. Since $R_p \subseteq R$, by Proposition 5.2.1 of [19] we obtain that $\mu \equiv_R \mu'$. The fact that transitions of $c \Rightarrow p'$ can be mimicked by $c \Rightarrow p$ follows by symmetry. For all other elements of R, the required implications follow from the assumption that R_p is a bisimulation relation, as above.

(b) If c does not hold for the given valuation, then both $c \Rightarrow p$ and $c \Rightarrow p'$ have no derivations at all. If c does hold, the proof is analogous to the proof of 1(b) and 2(b).

(4) We choose R to be the symmetric, reflexive, and transitive closure of the set

$$R_p \cup \left\{ \left(a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p, a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p' \right) \right\}$$

(a) Let $(a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p) \xrightarrow{\alpha} \mu$. Then, by the operational semantics $\alpha = a(\boldsymbol{t})$, and

$$\forall \boldsymbol{d} \in \boldsymbol{D} \,.\, \mu(p[\boldsymbol{x} := \boldsymbol{d}]) = \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ p[\boldsymbol{x}:=\boldsymbol{d}]=p[\boldsymbol{x}:=\boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}']$$

Then, also $(a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : p') \xrightarrow{\alpha} \mu'$, where $\alpha = a(\boldsymbol{t})$ and

$$\forall \boldsymbol{d} \in \boldsymbol{D} \,.\, \mu'(p'[\boldsymbol{x} := \boldsymbol{d}]) = \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ p'[\boldsymbol{x}:=\boldsymbol{d}]=p'[\boldsymbol{x}:=\boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}']$$

From the assumption that $p \sim_{\mathrm{dp}} p'$ for all valuations (by the bisimulation relation R_p), it immediately follows that $p[\boldsymbol{x} := \boldsymbol{d}] \sim_{\mathrm{dp}} p'[\boldsymbol{x} := \boldsymbol{d}]$ for any $\boldsymbol{d} \in \boldsymbol{D}$, so $(p[\boldsymbol{x} := \boldsymbol{d}], p'[\boldsymbol{x} := \boldsymbol{d}]) \in R_p$. Since μ and μ' both assign probability $f[\boldsymbol{x} := \boldsymbol{d}]$ to these process terms, they assign equal probabilities to each equivalence class of R_p; hence, $\mu \equiv_{R_p} \mu'$ and thus $\mu \equiv_R \mu'$ due to $R \supseteq R_p$ and Proposition 5.2.1 of [19]. (Note that for instance μ might have $p[\boldsymbol{x} := \boldsymbol{d}] = p[\boldsymbol{x} := \boldsymbol{d}']$ and therefore assign probability $f[\boldsymbol{x} := \boldsymbol{d}] + f[\boldsymbol{x} := \boldsymbol{d}']$ to this term. However, even if $p'[\boldsymbol{x} := \boldsymbol{d}] \neq p'[\boldsymbol{x} := \boldsymbol{d}']$ and therefore μ' does not combine these probabilities, still all terms are in the same equivalence class, and therefore everything still matches.)

Again, the mimicking the other way around follows by symmetry. For all other elements of R, the required implications follow from the assumption that R_p is a bisimulation relation, as above.

(b) The proof is analogous to the proof of 1(b) and 2(b).

(5) We choose R to be the symmetric, reflexive, transitive closure of the set

$$R_p \cup \{(Y(\boldsymbol{t}), Y'(\boldsymbol{t})\}$$

(a) Let $Y(\boldsymbol{t}) \xrightarrow{\alpha} \mu$. Then, by the operational semantics, also $p[\boldsymbol{x} := \boldsymbol{t}] \xrightarrow{\alpha} \mu$. From the assumption that $p \sim_{\mathrm{dp}} p'$ for all valuations, it immediately follows that $p[\boldsymbol{x} := \boldsymbol{t}] \sim_{\mathrm{dp}} p'[\boldsymbol{x} := \boldsymbol{t}]$. Therefore, also $p'[\boldsymbol{x} := \boldsymbol{t}] \xrightarrow{\alpha} \mu'$ with $\mu \equiv_{R_p} \mu'$ and thus $\mu \equiv_R \mu'$ due to $R \supseteq R_p$ and Proposition 5.2.1 of [19]. The fact that transitions of $Y'(\boldsymbol{t})$ can be mimicked by $Y(\boldsymbol{t})$ follows by symmetry. For all other elements of R, the required implications follow from the assumption that R_p is a bisimulation relation.

(b) The proof is analogous to the proof of 1(b) and 2(b). □

A.2 Proof of Theorem 2

The following fundamental result is needed in the proofs later on. It states that, if $\mu \equiv_R \mu'$, then also $\mu_f \equiv_{R_f} \mu'_f$, where R_f is the lifting of R over a bijective function f.

Lemma 2. *Let S, T be countable sets, $\mu, \mu' \in \mathsf{Distr}(S)$, and $R \subseteq S \times S$ an equivalence relation such that $\mu \equiv_R \mu'$. Given a bijective function $f\colon S \to T$, the set*

$$R_f = \{(t, t') \in T^2 \mid (f^{-1}(t)), f^{-1}(t')) \in R\}$$

is an equivalence relation and $\mu_f \equiv_{R_f} \mu'_f$.

Proof. For any $t \in T$, we have $(t, t) \in R_f$ since $(f^{-1}(t), f^{-1}(t)) \in R$ due to reflexivity of R; hence, R_f is also reflexive. For any $(t, t') \in R_f$ it holds that $(f^{-1}(t), f^{-1}(t')) \in R$, so by symmetry of R also $(f^{-1}(t'), f^{-1}(t)) \in R$ and hence $(t', t) \in R_f$. Therefore, R_f is also symmetric. For any $(t, t') \in R_f$ and $(t', t'') \in R_f$, we find $(f^{-1}(t), f^{-1}(t')) \in R$ and $(f^{-1}(t'), f^{-1}(t'')) \in R$, so $(f^{-1}(t), f^{-1}(t'')) \in R$ by transitivity of R, and hence also $(t, t'') \in R_f$. Therefore, R_f is also transitive.

Now, let $[t]_{R_f}$ be an arbitrary equivalence class of R_f, then

$$\begin{aligned}
&\mu_f([t]_{R_f}) && \{ \text{ Def. of probability of sets } \}\\
&= \sum_{t' \in [t]_{R_f}} \mu_f(t') && \{ \text{ Def. of lifting of distributions } \}\\
&= \sum_{t' \in [t]_{R_f}} \mu(f^{-1}(t')) && \{ \text{ Disjointness of inverses } \}\\
&= \mu\Big(\bigcup_{t' \in [t]_{R_f}} \{f^{-1}(t')\} \Big) && \{ \text{ Def. of inverse } \}\\
&= \mu\Big(\bigcup_{t' \in [t]_{R_f}} \{s \in S \mid f(s) = t'\} \Big) && \{ \text{ Easy rewriting } \}\\
&= \mu(\{s \in S \mid f(s) \in [t]_{R_f}\}) && \{ \text{ See below } \}\\
&= \sum_{\substack{[s]_R \in S/R \\ f(s) \in [t]_{R_f}}} \mu([s]_R)
\end{aligned}$$

To see why the final equality holds, we show that $f(s) \in [t]_{R_f}$ if and only if $f(s') \in [t]_{R_f}$ for every $s' \in [s]_R$ (note that the 'if' part of this statement is trivial, since $s \in [s]_R$). Then, the total probability of all states s such that $f(s) \in [t]_{R_f}$ clearly corresponds to the total probability of all classes of states for which at least one state has this property.

Let $s \in S$ such that $f(s) \in [t]_{R_f}$, and let $s' \in [s]_R$. So, by definition of equivalence classes, $(s, s') \in R$. Hence, by definition of R_f also $(f(s), f(s')) \in R_f$. Since $f(s) \in [t]_{R_f}$, therefore by definition of equivalence classes $(f(s), t) \in R_f$. Finally, by symmetry and transitivity of R_f we obtain $(f(s'), t) \in R_f$ and thus $f(s') \in [t]_{R_f}$.

In exactly the same way as above, we can show that

$$\mu'_f([t]_{R_f}) = \sum_{\substack{[s]_R \in S/R \\ f(s) \in [t]_{R_f}}} \mu'([s]_R)$$

Now, since $\mu([s]_R) = \mu'([s]_R)$ for every $s \in S$ (by definition of $\equiv$ and due to the assumption $\mu \equiv_R \mu'$), we obtain $\mu_f([t]_{R_f}) = \mu'_f([t]_{R_f})$ and hence $\mu_f \equiv_{R_f} \mu'_f$. □

Based on the encoding and decoding rules, we can prove the following results. Note that the Lemma 3 implies that dec and enc are bijective.

Lemma 3. *Restricting to MAPA specifications without any* rate *actions, the functions* dec *and* enc *are each others' inverse. That is,*

$$\mathsf{dec} \circ \mathsf{enc} = \mathsf{id}_{\mathsf{m}} \qquad \textit{and} \qquad \mathsf{enc} \circ \mathsf{dec} = \mathsf{id}_{\mathsf{p}}$$

where id_{m} *is the identity function on MAPA process terms and* id_{p} *is the identity function on prCRL process terms.*

Proof. We show that $\mathsf{dec}\,(\mathsf{enc}\,(p)) = p$ for every MAPA process term p, by induction on the structure of p. It can be shown similarly that $\mathsf{enc}\,(\mathsf{dec}\,(p)) = p$ for every prCRL term p.

Base case Let $p = Y(\boldsymbol{t})$. Then, $\mathsf{dec}\,(\mathsf{enc}\,(p)) = \mathsf{dec}\,(\mathsf{enc}\,(Y(\boldsymbol{t}))) = \mathsf{dec}\,(Y(\boldsymbol{t})) = Y(\boldsymbol{t}) = p$.

Inductive case Let $\mathsf{dec}\,(\mathsf{enc}\,(p)) = p$ and $\mathsf{dec}\,(\mathsf{enc}\,(q)) = q$. Now:

$$\begin{aligned}
&\mathsf{dec}\,(\mathsf{enc}\,(c \Rightarrow p)) && \{ \text{ Def. of } \mathsf{enc}\,() \ \} \\
&= \mathsf{dec}\,(c \Rightarrow \mathsf{enc}\,(p)) && \{ \text{ Def. of } \mathsf{dec}\,() \ \} \\
&= c \Rightarrow \mathsf{dec}\,(\mathsf{enc}\,(p)) && \{ \text{ Induction hypothesis } \} \\
&= c \Rightarrow p
\end{aligned}$$

We can show in exactly the same way that

$$\begin{aligned}
\mathsf{dec}\,(\mathsf{enc}\,(p + q)) &= p + q \\
\mathsf{dec}\,(\mathsf{enc}\,(\textstyle\sum_{\boldsymbol{x}:\boldsymbol{D}} p)) &= \textstyle\sum_{\boldsymbol{x}:\boldsymbol{D}} p \\
\mathsf{dec}\,(\mathsf{enc}\,(a(\boldsymbol{t}) \textstyle\sum_{\boldsymbol{x}:\boldsymbol{D}} f : p)) &= a(\boldsymbol{t}) \textstyle\sum_{\boldsymbol{x}:\boldsymbol{D}} f : p
\end{aligned}$$

where for the last equation, we need the assumption that $a \neq \mathsf{rate}$. Finally,

$$\begin{aligned}
\mathsf{dec}\,(\mathsf{enc}\,((\lambda) \cdot p)) &= \mathsf{dec}\,(\mathsf{rate}(\lambda) \sum_{x:\{*\}} 1 : \mathsf{enc}\,(p)) \\
&= (\lambda) \cdot \mathsf{dec}\,(\mathsf{enc}\,(p)) = (\lambda) \cdot p
\end{aligned}$$ □

The following lemma states that enc is similar to a functional bisimulation, except that it relates MAPA process terms to prCRL process terms.

Lemma 4. *Let m be a MAPA process term. Then, for every action $a \neq \mathsf{rate}$ and distribution μ,*

$$m \overset{a}{\hookrightarrow} \mu \quad \iff \quad \mathsf{enc}\,(m) \overset{a}{\to} \mu_{\mathsf{enc}}$$

Proof. Let $m \overset{a}{\hookrightarrow} \mu$. We prove that $\mathsf{enc}\,(m) \overset{a}{\to} \mu_{\mathsf{enc}}$ by induction on the structure of m. The reverse can be proven symmetrically, noting that dec indeed decodes a transition like $\mathsf{enc}\,(m) \overset{a}{\to} \mu_{\mathsf{enc}}$ to an interactive transition if $a \neq \mathsf{rate}$.

Base case. Let $m = b(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : m'$. Since $m \overset{a}{\hookrightarrow} \mu$, by the SOS rules it must hold that $a = b(\boldsymbol{t})$ and

$$\forall \boldsymbol{d} \in \boldsymbol{D} \,.\, \mu(m'[\boldsymbol{x} := \boldsymbol{d}]) = \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ m'[\boldsymbol{x}:=\boldsymbol{d}]=m'[\boldsymbol{x}:=\boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}']$$

Now, by definition of enc we have $\mathsf{enc}\,(m) = b(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : \mathsf{enc}\,(m')$. Hence, by the SOS rules for prCRL it holds that $\mathsf{enc}\,(m) \overset{a}{\to} \mu'$, where

$$\forall \boldsymbol{d} \in \boldsymbol{D} \,.\, \mu'(\mathsf{enc}\,(m')\,[\boldsymbol{x} := \boldsymbol{d}]) = \sum_{\substack{\boldsymbol{d}' \in \boldsymbol{D} \\ \mathsf{enc}(m')[\boldsymbol{x}:=\boldsymbol{d}]=\mathsf{enc}(m')[\boldsymbol{x}:=\boldsymbol{d}']}} f[\boldsymbol{x} := \boldsymbol{d}']$$

Since the enc function does neither introduce nor remove variables, it follows that, for every $\boldsymbol{d}' \in \boldsymbol{D}$, $\mathsf{enc}\,(m')\,[\boldsymbol{x} := \boldsymbol{d}] = \mathsf{enc}\,(m')\,[\boldsymbol{x} := \boldsymbol{d}']$ holds if and only if $m'[\boldsymbol{x} := \boldsymbol{d}] = m'[\boldsymbol{x} := \boldsymbol{d}']$ holds. Hence, the right-hand sides of the two equations coincide. Also, note that $\mathsf{enc}\,(m')\,[\boldsymbol{x} := \boldsymbol{d}] = \mathsf{enc}\,(m'[\boldsymbol{x} := \boldsymbol{d}])$. Therefore $\mu'(\mathsf{enc}\,(m'')) = \mu(m'')$ for every MAPA process term m''. By definition, this implies that $\mu' = \mu_{\mathsf{enc}}$.

Inductive case. Let $m = m' + m''$. Since $m \overset{a}{\hookrightarrow} \mu$, by the SOS rules it must hold that either $m' \overset{a}{\hookrightarrow} \mu$ or $m'' \overset{a}{\hookrightarrow} \mu$. By induction, this implies that either $\mathsf{enc}\,(m') \overset{a}{\to} \mu_{\mathsf{enc}}$ or $\mathsf{enc}\,(m'') \overset{a}{\to} \mu_{\mathsf{enc}}$. Since $\mathsf{enc}\,(m) = \mathsf{enc}\,(m') + \mathsf{enc}\,(m'')$, the SOS rules for prCRL imply that $\mathsf{enc}\,(m) \overset{a}{\to} \mu_{\mathsf{enc}}$.

The cases where $m = Y(\boldsymbol{t})$, $m = c \Rightarrow p$ or $m = \sum_{\boldsymbol{x}:\boldsymbol{D}} p$ are proven in the same way. □

Lemma 5. *Let m be a MAPA process term. Then, for every process term m', rate λ and Markovian derivation $\mathcal{D}$,*

$$m \overset{\lambda}{\to}_{\mathcal{D}} m' \quad \iff \quad \mathsf{enc}\,(m) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(m')}$$

where $\mathcal{D}'$ is obtained from $\mathcal{D}$ by substituting PSum *for* MStep.

Proof. Let $m \overset{\lambda}{\to}_{\mathcal{D}} m'$. We prove that $\mathsf{enc}\,(m) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(m')}$, by induction on the structure of m. The reverse can be proven symmetrically.

Base case. Let $m = (\kappa) \cdot m'$. Since $m \xrightarrow{\lambda}_{\mathcal{D}} m'$, by the SOS rules it must hold that $\kappa = \lambda$ and $\mathcal{D} = \langle \text{MSTEP} \rangle$. Hence, $\mathsf{enc}\,(m) = \mathsf{rate}(\lambda) \sum_{x:\{*\}} 1 : \mathsf{enc}\,(m')$.

The derivation $\mathcal{D}'$, corresponding to $\mathcal{D}$, is $\langle \text{PSUM} \rangle$. Note that, by the SOS rules of prCRL and the fact that x does not occur in $\mathsf{enc}\,(m')$ by definition of enc, indeed $\mathsf{enc}\,(m) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(m')}$.

Inductive case. Let $m = m_1 + m_2$. Since $m \xrightarrow{\lambda}_{\mathcal{D}} m'$, by the SOS rules it must hold that either $m_1 \xrightarrow{\lambda}_{\mathcal{D}_1} m'$ and $\mathcal{D} = \langle \text{NCHOICEL} \rangle + \mathcal{D}_1$ or $m_2 \xrightarrow{\lambda}_{\mathcal{D}_2} m'$ and $\mathcal{D} = \langle \text{NCHOICER} \rangle + \mathcal{D}_2$. Assume the first (the proof for the other option is symmetrical).

By induction, this implies that $\mathsf{enc}\,(m_1) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}_1'} \mathsf{enc}\,(m')$. Since $\mathsf{enc}\,(m) = \mathsf{enc}\,(m_1) + \mathsf{enc}\,(m_2)$, the SOS rules for prCRL imply that $\mathsf{enc}\,(m) \xrightarrow{\mathsf{rate}(\lambda)}_{\mathcal{D}_1''} \mathsf{enc}\,(m')$, where $\mathcal{D}_1'' = \langle \text{NCHOICEL} \rangle + \mathcal{D}_1'$. Since we already saw that $\mathcal{D} = \langle \text{NCHOICEL} \rangle + \mathcal{D}_1$, indeed $\mathcal{D}_1'' = \mathcal{D}'$.

The cases where $m = Y(\boldsymbol{t})$, $m = c \Rightarrow p$ or $m = \sum_{\boldsymbol{x}:\boldsymbol{D}} p$ are proven in the same way. □

Lemma 6. *Let P_1, P_2 be prCRL specifications. Then,*

$$P_1 \sim_{\text{dp}} P_2 \quad \Rightarrow \quad \mathsf{dec}\,(P_1) \sim \mathsf{dec}\,(P_2)\,.$$

Proof. Assume that $P_1 \sim_{\text{dp}} P_2$, and let $\mathcal{M}_1 = \langle S, s_1^0, A, \hookrightarrow, \rightsquigarrow \rangle$ and $\mathcal{M}_2 = \langle S, s_2^0, A, \hookrightarrow, \rightsquigarrow \rangle$ be the MAs that represent the semantics of $\mathsf{dec}\,(P_1)$ and $\mathsf{dec}\,(P_2)$. Let R be the derivation-preserving bisimulation relating P_1 and P_2.

Now, consider the bisimulation relation R' over MAPA terms, given by $R' = \{(\mathsf{dec}\,(p)\,, \mathsf{dec}\,(q)) \mid (p, q) \in R\}$. It is easy to see that R' is an equivalence relation, since R is one. We now show that it is a bisimulation relation relating $\mathcal{M}_1$ and $\mathcal{M}_2$, and therefore proving the result.

First, since the initial states of P_1 and P_2 are related by R, the initial states of $\mathsf{dec}\,(P_1)$ and $\mathsf{dec}\,(P_2)$ are related by R' by definition of dec. Second, let $(s, t) \in R'$ and assume that $s \xrightarrow{\alpha} \mu$. We show that $t \xrightarrow{\alpha} \mu'$ such that $\mu \equiv_{R'} \mu'$. Note that either (1) $\alpha \in A$ and $s \xhookrightarrow{\alpha} \mu$, or (2) $\alpha = \chi(\mathit{rate}(s))$, $\mathit{rate}(s) > 0$, $\mu = \mathbb{P}_s$ and there is no μ' such that $s \xhookrightarrow{\tau} \mu'$. Also note that $\alpha \neq \mathsf{rate}$, by definition of dec.

(1) Let $s \xhookrightarrow{\alpha} \mu$ for some $\alpha \in A$ such that $\alpha \neq \mathsf{rate}$. We need to show that $t \xhookrightarrow{\alpha} \mu'$ such that $\mu \equiv_{R'} \mu'$. First note that, by Lemma 4, we have $\mathsf{enc}\,(s) \xrightarrow{\alpha} \mu_{\mathsf{enc}}$. We know that $(s, t) \in R'$, so $(\mathsf{enc}\,(s)\,, \mathsf{enc}\,(t)) \in R$. Since $\mathsf{enc}\,(s) \xrightarrow{\alpha} \mu_{\mathsf{enc}}$ and R is a bisimulation relation, this implies that $\mathsf{enc}\,(t) \xrightarrow{\alpha} \nu$ such that $\mu_{\mathsf{enc}} \equiv_R \nu$. Then, $t \xhookrightarrow{\alpha} \nu_{\mathsf{dec}}$ by Lemma 4.

Now, note that R' can be seen as R_{dec} as defined in Lemma 2. Hence, by this lemma $\mu_{\mathsf{enc}} \equiv_R \nu$ implies $\mu_{(\mathsf{dec} \circ \mathsf{enc})} \equiv_{R'} \nu_{\mathsf{dec}}$. By Lemma 3, this reduces to $\mu \equiv_{R'} \nu_{\mathsf{dec}}$, which is what we wanted to show.

Figure 6 illustrates this part of the proof.

(2) Let $\alpha = \chi(\mathit{rate}(s))$, $\mathit{rate}(s) > 0$, $\mu = \mathbb{P}_s$ and let there be no μ' such that

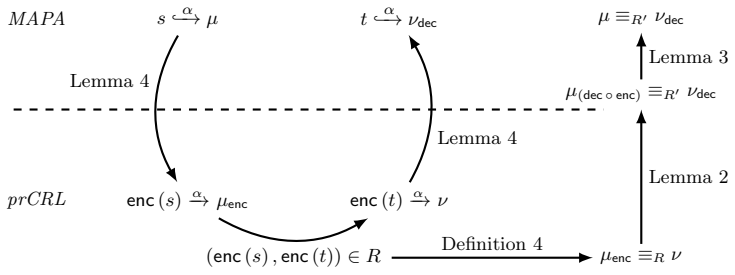

Fig. 6. Visualisation of the proof of Lemma 6 part (1).

$s \xhookrightarrow{\tau} \mu'$. We need to show that $t \xrightarrow{\alpha} \nu$ such that $\mu \equiv_{R'} \nu$, i.e., that (a) $\mathit{rate}(t) = \mathit{rate}(s)$, that (b) $\mathbb{P}_s \equiv_{R'} \mathbb{P}_t$ and that (c) there is no μ' such that $t \xhookrightarrow{\tau} \mu'$.

For (a), note that

$$\mathit{rate}(s) = \sum_{s' \in S} \mathit{rate}(s, s') = \sum_{s' \in S} \sum_{(s,\lambda,s') \in \rightsquigarrow} \lambda$$

by Definition 2, and that by the operational semantics, we have $(s, \lambda, s') \in \rightsquigarrow$ if and only if $\mathsf{MD}(s, s') \neq \emptyset$ and $\lambda = \sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}(s,s')} \lambda_i$. Combining this, we obtain

$$\mathit{rate}(s) = \sum_{s' \in S} \sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}(s,s')} \lambda_i = \sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}(s)} \lambda_i$$

where $\mathsf{MD}(s) = \bigcup_{s' \in S} \mathsf{MD}(s, s') = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists s' \in S \,.\, s \xrightarrow{\lambda_i}_{\mathcal{D}} s'\}$. This rewriting is valid since all sets $\mathsf{MD}(s, s')$ are disjoint, because every Markovian derivation $\mathcal{D}$ yields a single target state s'.

Now, let

$$\mathsf{MD}'(\mathsf{enc}(s)) = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists s' \in S \,.\, \mathsf{enc}(s) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(s')}\}$$

where $\mathcal{D}'$ is again obtained from $\mathcal{D}$ by substituting PSum for MStep. By Lemma 5, it follows that $\mathsf{MD}(s) = \mathsf{MD}'(\mathsf{enc}(s))$. Hence, $\mathsf{enc}(s)$ has the same outgoing transitions as s, except that the derivations are slightly different and that the target states are encoded too.

Similarly, $\mathit{rate}(t) = \sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}(t)} \lambda_i$ with $\mathsf{MD}(t) = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists t' \in S \,.\, t \xrightarrow{\lambda_i}_{\mathcal{D}} t'\}$, and $\mathsf{MD}(t) = \mathsf{MD}'(\mathsf{enc}(t)) = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists t' \in S \,.\, \mathsf{enc}(t) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}'} \mathbb{1}_{\mathsf{enc}(t')}\}$.

To show that $\mathit{rate}(s) = \mathit{rate}(t)$, it therefore remains to show that

$$\sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}'(\mathsf{enc}(s))} \lambda_i = \sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}'(\mathsf{enc}(t))} \lambda_i$$

To see why this is the case, first note that, since the bisimulation relation R is derivation-preserving, by definition we have

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

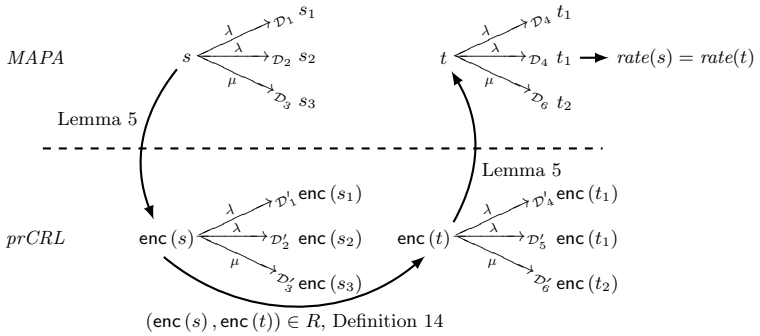

Fig. 7. Visualisation of the proof of Lemma 6 part (2).

for every $(p, q) \in R$, every equivalence equivalence class $[r]_R$ and every rate λ. Hence, as $(\mathsf{enc}(s), \mathsf{enc}(t)) \in R$, this also holds for $\mathsf{enc}(s), \mathsf{enc}(t)$, and therefore

$$|\{\mathcal{D} \in \Delta \mid \exists \mathsf{enc}(r') \in [\mathsf{enc}(r)]_R \,.\, \mathsf{enc}(s) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}} \mathsf{enc}(r')\}| =$$
$$|\{\mathcal{D} \in \Delta \mid \exists \mathsf{enc}(r') \in [\mathsf{enc}(r)]_R \,.\, \mathsf{enc}(t) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}} \mathsf{enc}(r')\}|$$

for every action $\mathsf{rate}(\lambda_i)$ and every equivalence class $[\mathsf{enc}(r)]_R$.

Note that the size of each of these sets is equal to the total number of derivations from $\mathsf{enc}(s)$ and $\mathsf{enc}(t)$ with action $\mathsf{rate}(\lambda_i)$ to a certain equivalence class $[\mathsf{enc}(r)]_R$. This equality immediately implies that, for every action $\mathsf{rate}(\lambda_i)$, the total number of $\mathsf{rate}(\lambda_i)$-derivations from $\mathsf{enc}(s)$ and $\mathsf{enc}(t)$ is also equal. Clearly, if there are as many $\mathsf{rate}(\lambda_i)$-derivations from $\mathsf{enc}(s)$ and $\mathsf{enc}(t)$ for every $\mathsf{rate}(\lambda_i)$, then by definition

$$\sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}'(\mathsf{enc}(s))} \lambda_i = \sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}'(\mathsf{enc}(t))} \lambda_i$$

which is what we needed to show.

Figure 7 illustrates this part of the proof, and can be helpful for next part as well.

For (b), note that $\mathbb{P}_s(s') = \frac{rate(s,s')}{rate(s)}$ and $\mathbb{P}_t(s') = \frac{rate(t,s')}{rate(t)}$. To show $\mathbb{P}_s \equiv_{R'} \mathbb{P}_t$, we need to prove that $\mathbb{P}_s([p]_{R'}) = \mathbb{P}_t([p]_{R'})$ for every equivalence class $[p]_{R'} \in S/R'$.

Let $[p]_{R'} \in S/R'$, then

$$\mathbb{P}_s([p]_{R'}) = \sum_{p' \in [p]_{R'}} \mathbb{P}_s(p') = \sum_{p' \in [p]_{R'}} \frac{rate(s,p')}{rate(s)} = \frac{\sum_{p' \in [p]_{R'}} rate(s,p')}{rate(s)}$$

Similarly, $\mathbb{P}_t([p]_{R'}) = \frac{\sum_{p' \in [p]_{R'}} rate(t,p')}{rate(t)}$. Since we already showed in part (b) that $rate(s) = rate(t)$, it remains to show that the numerators of these two fractions coincide.

As above, we can derive

$$\sum_{p' \in [p]_{R'}} rate(s, p') = \sum_{p' \in [p]_{R'}} \sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}(s, p')} \lambda_i = \sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}(s, [p]_{R'})} \lambda_i$$

where $\mathsf{MD}(s, [p]_{R'}) = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists s' \in [p]_{R'} \,.\, s \xrightarrow{\lambda_i}_{\mathcal{D}} s'\}$. Using Lemma 5 we find that $\mathsf{MD}(s, [p]_{R'}) = \mathsf{MD}'(\mathsf{enc}\,(s)\,, [\mathsf{enc}\,(p)]_R)$, where

$$\begin{aligned}&\mathsf{MD}'(\mathsf{enc}\,(s)\,, [\mathsf{enc}\,(p)]_R)\\ &\quad = \{(\lambda_i, \mathcal{D}) \in \mathbb{R} \times \Delta \mid \exists s' \in [\mathsf{enc}\,(p)]_R \,.\, \mathsf{enc}\,(s) \xrightarrow{\lambda_i}_{\mathcal{D}'} \mathsf{enc}\,(s')\}\end{aligned}$$

Similar derivations can be made for t, so it remains to show that

$$\sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}'(\mathsf{enc}(s), [\mathsf{enc}(p)]_R)} \lambda_i = \sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}'(\mathsf{enc}(t), [\mathsf{enc}(p)]_R)} \lambda_i$$

Just as in part (b), by assumption we have

$$\begin{aligned}&|\{\mathcal{D} \in \Delta \mid \exists \mathsf{enc}\,(p') \in [\mathsf{enc}\,(p)]_R \,.\, \mathsf{enc}\,(s) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}} \mathsf{enc}\,(p')\}| =\\ &|\{\mathcal{D} \in \Delta \mid \exists \mathsf{enc}\,(p') \in [\mathsf{enc}\,(p)]_R \,.\, \mathsf{enc}\,(t) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}} \mathsf{enc}\,(p')\}|\end{aligned}$$

for every action $\mathsf{rate}(\lambda_i)$.

Note again that the size of each of these sets is equal to the total number of derivations from $\mathsf{enc}\,(s)$ and $\mathsf{enc}\,(t)$ with action $\mathsf{rate}(\lambda_i)$ to a certain equivalence class $[\mathsf{enc}\,(p)]_R$. Hence, the fact that these two sets are of equal size implies that every transition $\mathsf{enc}\,(s) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}} \mathsf{enc}\,(s')$ with $\mathsf{enc}\,(s') \in [\mathsf{enc}\,(p)]_R$ corresponds one-to-one to a transition $\mathsf{enc}\,(t) \xrightarrow{\mathsf{rate}(\lambda_i)}_{\mathcal{D}'} \mathsf{enc}\,(t')$ such that $\mathsf{enc}\,(t') \in [\mathsf{enc}\,(p)]_R$.

This immediately implies that, for every action $\mathsf{rate}(\lambda_i)$, the total number of $\mathsf{rate}(\lambda_i)$-derivations to $[\mathsf{enc}\,(p)]_R$ from $\mathsf{enc}\,(s)$ and $\mathsf{enc}\,(t)$ is also equal. Therefore, by definition

$$\sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}'(\mathsf{enc}(s), [\mathsf{enc}(p)]_R)} \lambda_i = \sum_{(\lambda_i, \mathcal{D}) \in \mathsf{MD}'(\mathsf{enc}(t), [\mathsf{enc}(p)]_R)} \lambda_i$$

which is what we needed to show.

For (c), note that $(\mathsf{enc}\,(s)\,, \mathsf{enc}\,(t)) \in R$ since $(s, t) \in R'$. As there is no τ-transition from s, by Lemma 4 there is also no τ-transition from $\mathsf{enc}\,(s)$. Since R is a bisimulation relation, also $\mathsf{enc}\,(t)$ does not have a τ-transition, and applying Lemma 4 again also t does not have a τ-transition. □

Theorem 2. *Let f: 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. Let M be an arbitrary MAPA specification without any rate action. Since $f(P) \sim_{\mathrm{dp}} P$ for every prCRL specification, also $f(\mathsf{enc}\,(M)) \sim_{\mathrm{dp}} \mathsf{enc}\,(M)$. Lemma 6 therefore yields $\mathsf{dec}\,(f(\mathsf{enc}\,(M))) \sim \mathsf{dec}\,(\mathsf{enc}\,(M))$. Moreover, $M = \mathsf{dec}\,(\mathsf{enc}\,(M))$ by Lemma 3, and thus also $M \sim \mathsf{dec}\,(\mathsf{enc}\,(M))$. By transitivity of strong bisimulation, the result follows. □

A.3 Proof of Theorem 3

We now show that the linearisation procedure indeed preserves derivations. Note that the proof that we present here is very similar to the correctness proof of the algorithm, as shown in [11]. We only had to take derivation preservation into account at a few places in the proof.

The basic intuition is that linearisation never changes something of the form $p + p$ to p or vice versa, and neither does it remove nor introduce summations. Therefore, the bisimulation relation used in the original proof can just as well be applied to show that the procedure is also derivation preserving.

Lemma 7. *Let P be a prCRL specification, then* $\mathsf{linearise}(P) \sim_{\mathrm{dp}} P$.

Proof. Linearisation is defined in [11] as first applying Algorithm 1 (which uses Algorithm 2 and 3) and then applying Algorithm 4. The algorithms can be found on page 43, 44 and 46 of [11]. The fact that $\mathsf{linearise}(P) \sim P$ was already proven in Theorems 19 and 22 of [11], but it remains to prove that linearisation preserves derivations. We first show this for Algorithm 1 (based on the proof of Lemma 37 from [11]) and then for Algorithm 4 (based on the proof of Theorem 22 from [11]).

Algorithm 1 preserves derivations.

Let $P = (E, X_1(\boldsymbol{v}))$ be an arbitrary input prCRL specification for Algorithm 1 of [11] (having unique variable names), and let $\boldsymbol{v'}$ be the computed new initial vector. Let P' be the specification it returns. We show that $P \sim_{\mathrm{dp}} P'$ by showing that, before and after an arbitrary iteration of the algorithm's while loop, $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v'})) \sim_{\mathrm{dp}} P$. Since we already showed in Lemma 36 of [11] that Algorithm 1 terminates, and in Theorem 19 of [11] that at the end of the algorithm only $(\mathit{done}, X'_1(\boldsymbol{v'}))$ is returned and that the equations in *done* at that moment do not depend on any of the process equations in $E \cup \mathit{toTransform}$ anymore, this is sufficient.

For brevity, in this proof we write 'bisimilar' if we mean 'strongly derivation-preserving bisimilar for all valuations'. Also, the notation $p \sim_{\mathrm{dp}} q$ will be used for this.

We prove that $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v'})) \sim_{\mathrm{dp}} P$ before and after an arbitrary iteration of the algorithm's while loop, by induction on the number of iterations that have already been performed. We let $E = \{X_1(\boldsymbol{x_1} : \boldsymbol{D_1}) = p_1, \ldots, X_n(\boldsymbol{x_n} : \boldsymbol{D_n}) = p_n\}$, and hence we have $P = (\{X_1(\boldsymbol{x_1} : \boldsymbol{D_1}) = p_1, \ldots, X_n(\boldsymbol{x_n} : \boldsymbol{D_n}) = p_n\}, X_1(\boldsymbol{v}))$.

Base case. Before the first iteration, the parameters of the new processes are determined. Every process will have the same parameters: $\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}$. This is the union of all process variables of the original processes, extended with a parameter for every nondeterministic or probabilistic sum binding a variable that is used later on. Also, the new initial state vector $\boldsymbol{v'}$ is computed by taking the

original initial vector $\boldsymbol{v}$, and appending dummy values for all added parameters. Furthermore, *done* is set to $\emptyset$ and *toTransform* to $\{X'_1(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p_1\}$.

Clearly, $X'_1(\boldsymbol{v'})$ is identical to $X_1(\boldsymbol{v})$, except that it has more global variables (without overlap, as we assumed specifications to have unique variable names). However, these additional global variables are not used in p_1, otherwise they would be free in $X_1(\boldsymbol{x_1} : \boldsymbol{D_1}) = p_1$ (which is not allowed by Definition 9). Therefore, $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v'}))$ and P are obviously bisimilar (by the trivial bisimulation relation that relates $X_1(\boldsymbol{v})$ to $X'_1(\boldsymbol{v'})$ and contains the identity relation).

Since the additional unused variables do not affect the possible derivations in any way, this bisimulation is derivation preserving.

Inductive case. Now assume that $k \geq 0$ iterations have passed. Without loss of generality, assume that each time a process $(X'_i(\mathit{pars}) = p_i) \in \mathit{toTransform}$ had to be chosen, it was the one with the smallest i. Then, after these k iterations, $\mathit{done} = \{X'_1(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_1, \ldots, X'_k(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_k\}$. Also, $\mathit{toTransform} = \{X'_{k+1}(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_{k+1}, \ldots, X'_l(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_l\}$ for some $l \geq k$. We have $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v'})) \sim_{\mathrm{dp}} P$ by the induction hypothesis.

We prove that after $k+1$ iterations, $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v'}))$ is still derivation-preserving bisimilar to P. During iteration $k+1$ three things happen: (1) the process equation $X'_{k+1}(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_{k+1}$ is removed from *toTransform*; (2) an equation $X'_{k+1}(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p''_{k+1}$ is added to *done*; (3) potentially, one or more equations of the form $X'_{l+1}(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_{l+1}, \ldots, X'_m(\boldsymbol{x_1} : \boldsymbol{D_1}, \boldsymbol{x'} : \boldsymbol{D'}) = p'_m$ are added to *toTransform*.

As the other equations in $E \cup \mathit{done} \cup \mathit{toTransform}$ do not change, Theorem 1 implies that $(E \cup \mathit{done} \cup \mathit{toTransform}, X'_1(\boldsymbol{v'})) \sim_{\mathrm{dp}} P$ still holds if and only if $p'_{k+1} \sim_{\mathrm{dp}} p''_{k+1}$. We show this by induction on the structure of p'_{k+1}.

The base case is $p'_{k+1} = a(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : q$. We now make a case distinction based on whether there already is a process equation in either *done* or *toTransform* whose right-hand side is an IRF corresponding to the normal form of q (which is just q when q is not a process instantiation, otherwise it is the right-hand side of the process it instantiates), as indicated by the variable *bindings*.

Case 1a: *There does not already exist a process equation* $X'_j(\mathit{pars}) = q'$ *in* bindings *such that* q' *is the normal form of* q.

> In this case, a new process equation $X'_{l+1}(\mathit{pars}) = q'$ is added to *toTransform* via line 6 of Algorithm 2, and $p''_{k+1} = a(\boldsymbol{t})\sum_{\boldsymbol{x}:\boldsymbol{D}} f : X'_{l+1}(\mathit{actualPars})$.
>
> When q was *not* a process instantiation, the actual parameters for X'_{l+1} are just the unchanged global variables, with those that are not used in q reset (line 4 of Algorithm 3). Since (by definition of the normal form) the right-hand side of X'_{l+1} is identical to q and *actualPars* takes care that all data parameters keep the same value, clearly $X'_{l+1}(\mathit{actualPars})$ is derivation-preserving bisimilar to q: every derivation $\mathcal{D}$ of q corresponds to the derivation $\textsc{Inst} + \mathcal{D}$ of $X'_{l+1}(\mathit{actualPars})$. Therefore, by Theorem 1 indeed also $p''_{k+1} \sim_{\mathrm{dp}} p'_{k+1}$.

When $q = Y(t_1, t_2, \ldots, t_n)$, there should occur some substitutions to ascertain that $X'_{l+1}(\mathit{actualPars})$ is bisimilar to q. Since $X'_{l+1}(\mathit{actualPars}) = q'$, with q' the right-hand side of Y, the actual parameters to be provided to X'_{l+1} should include $t_1, t_2, \ldots, t_n$ for the global variables of X'_{l+1} that correspond to the original global variables of Y. All other global variables can be reset, as they cannot be used by Y anyway. This indeed happens in line 2 of Algorithm 3, so all behaviours of q are present in $X'_{l+1}(\mathit{actualPars})$ are vice versa, and all derivations of q map one-to-one to the derivations of $X'_{l+1}(\mathit{actualPars})$. So, $q \sim_{\mathrm{dp}} X'_{l+1}(\mathit{actualPars})$, and therefore, by Theorem 1 indeed also $p''_{k+1} \sim_{\mathrm{dp}} p'_{k+1}$.

Case 1b: *There exists a process equation* $X'_j(\mathit{pars}) = q'$ *in* bindings *such that* q' *is the normal form of* q.

In this case, we obtain $p''_{k+1} = a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : X'_j(\mathit{actualPars})$ from line 4 of Algorithm 2. By Theorem 1, we only need to show that q is derivation-preserving bisimilar to $X'_j(\mathit{actualPars})$.

Note that the fact that $X'_j(\mathit{pars}) = q'$ is in *bindings* implies that at some point $X'_j(\mathit{pars}) = q'$ was in *toTransform*. In case it was already transformed in an earlier iteration there is now a process $X'_j(\mathit{pars}) = q''$ in *done* such that $q'' \sim_{\mathrm{dp}} q'$ (by induction). Otherwise, $X'_j(\mathit{pars}) = q'$ is still in *toTransform*.

In both cases, $\mathit{done} \cup \mathit{toTransform} \cup P$ contains a process $X'_j(\mathit{pars}) = q''$ such that $q'' \sim_{\mathrm{dp}} q'$, and therefore it is correct to take $p''_{k+1} = a(\boldsymbol{t}) \sum_{\boldsymbol{x}:\boldsymbol{D}} f : X'_j(\mathit{actualPars})$. The reasoning to see that indeed $p''_{k+1} \sim p'_{k+1}$ then only depends on the choice of *actualPars*, and is the same as for Case 1a.

Now, assume that q_1 and q_2 are process terms for which Algorithm 2 provided the bisimilar process terms p'''_{k+1} and p''''_{k+1}. We prove that p''_{k+1} (as obtained from Algorithm 2) is bisimilar to p'_{k+1} for the remaining possible structures of p'_{k+1}.

Case 2: $p'_{k+1} = c \Rightarrow q_1$.

In this case, Algorithm 2 yields $p''_{k+1} = c \Rightarrow p'''_{k+1}$, which according to Theorem 1 is bisimilar to p'_{k+1}, since $q_1 \sim_{\mathrm{dp}} p'''_{k+1}$.

Case 3: $p'_{k+1} = q_1 + q_2$.

In this case, Algorithm 2 yields $p''_{k+1} = p'''_{k+1} + p''''_{k+1}$, which according to Theorem 1 is bisimilar to p'_{k+1}, since $q_1 \sim_{\mathrm{dp}} p'''_{k+1}$ and $q_2 \sim_{\mathrm{dp}} p''''_{k+1}$.

Case 4: $p'_{k+1} = Y(\boldsymbol{t})$, *where we assume that* $Y(\boldsymbol{x} : \boldsymbol{D}) = q_1$.

In this case, Algorithm 2 yields $p''_{k+1} = p'''_{k+1}$, with $\boldsymbol{x}$ substituted by $\boldsymbol{t}$, which is bisimilar to p'_{k+1} (as it precisely follows the SOS rule Inst). To see that the bisimulation preserves derivations, note that every derivation $\mathcal{D}$ of p'''_{k+1} corresponds one-to-one to a derivation $\textsc{Inst} + \mathcal{D}$ of p'_{k+1}.

Case 5: $p'_{k+1} = \sum_{x:D} q_1$.

In this case, Algorithm 2 yields $p''_{k+1} = \sum_{x:D} p'''_{k+1}$, which according to Theorem 1 is bisimilar to p'_{k+1}, since $q_1 \sim_{\mathrm{dp}} p'''_{k+1}$.

Since in all cases the process term p''_{k+1} obtained from Algorithm 2 is strongly probabilistically derivation-preserving bisimilar to p'_{k+1} for all valuations, the lemma holds.

Algorithm 4 preserves derivations.

In [11], it is shown that the input and output of Algorithm 4 are isomorphic. It is easy to see that the isomorphism that is used to show this is a derivation-preserving bisimulation: the current proof that each transition of $X'_i(\boldsymbol{u})$ can be mapped one-to-one to a transition of $X(i, \boldsymbol{u})$ just as well applies to show that every derivation of $X'_i(\boldsymbol{u})$ maps one-to-one to a derivation of $X(i, \boldsymbol{u})$. □

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.*

Proof. Lemma 7 showed that linearise respects derivation-preserving bisimulation. Hence, Theorem 2 yields $\mathsf{dec}\,(\mathsf{linearise}(\mathsf{enc}\,(M))) \sim M$.

It was already proven in [11] that $\mathsf{linearise}(\mathsf{enc}\,(M))$ is an LPPE. Since decoding does not change the structure of a specification in any way (except for changing rate-actions to λ-actions), M' is indeed in MLPPE format. □

B Parallel composition of MAPA terms

In this section, we generalise the definition of parallel prCRL [11] to parallel MAPA. The technicalities are almost identical; the only difference is in the two additional SOS rules Par-L-Rate and Par-R-Rate, that state that the parallel composition of two processes can still do the same Markovian transitions as those processes individually.

B.1 Parallel MAPA

Definition 18. *A* process term *in parallel MAPA is any term that can be generated by the following grammar:*

$$q \;::=\; p \;\mid\; q \,||\, q \;\mid\; \partial_E(q) \;\mid\; \tau_H(q) \;\mid\; \rho_R(q)$$

Here, p is a MAPA process term, $E, H \subseteq Act$ are sets of actions, and the function $R\colon Act \to Act$ maps actions to actions. A parallel MAPA specification $P = (\{X_i(\boldsymbol{x_i} : \boldsymbol{D_i}) = q_i\}, X_j(\boldsymbol{t}))$ *is a set of* parallel MAPA process equations *(which are like MAPA process equations, but with parallel MAPA process terms as right-hand sides) together with an initial process. The well-formedness criteria of Definition 9 are lifted in the obvious way.*

In a parallel MAPA process term, $q_1 \,||\, q_2$ is parallel composition. Furthermore, $\partial_E(q)$ encapsulates the actions in E, $\tau_H(q)$ hides the actions in H (renaming them to τ and removing their parameters), and $\rho_R(q)$ renames actions using R. Parallel processes by default interleave all actions. However, we assume a partial function $\gamma\colon Act \times Act \to Act$ that specifies which actions can communicate; more precisely, $\gamma(a,b) = c$ denotes that a and b can communicate if their parameters are equal, resulting in the action c with these parameters (as in ACP [1]).

The SOS rules for parallel MAPA are shown in Figure 8 (relying on the SOS rules for MAPA from Figure 4), where for any probability distribution μ, we denote by $\tau_H(\mu)$ the probability distribution μ' such that $\forall p \,.\, \mu'(\tau_H(p)) = \mu(p)$. Similarly, we use $\rho_R(\mu)$ and $\partial_E(\mu)$. Note that there is no Encap-T rule, to remove transitions labelled by an encapsulated action.

Also note that, if $p \xrightarrow{\lambda_1} p$ and $q \xrightarrow{\lambda_2} q$, we obtain a derivation for $p \,||\, q \xrightarrow{\lambda_1} p \,||\, q$ and one for $p \,||\, q \xrightarrow{\lambda_2} p \,||\, q$. By the operational semantics, as defined in Section 3.1, this means that the underlying MA will have a transition $p \,||\, q \overset{\lambda_1+\lambda_2}{\rightsquigarrow} p \,||\, q$. This precisely corresponds to the definition of parallel composition in [6].

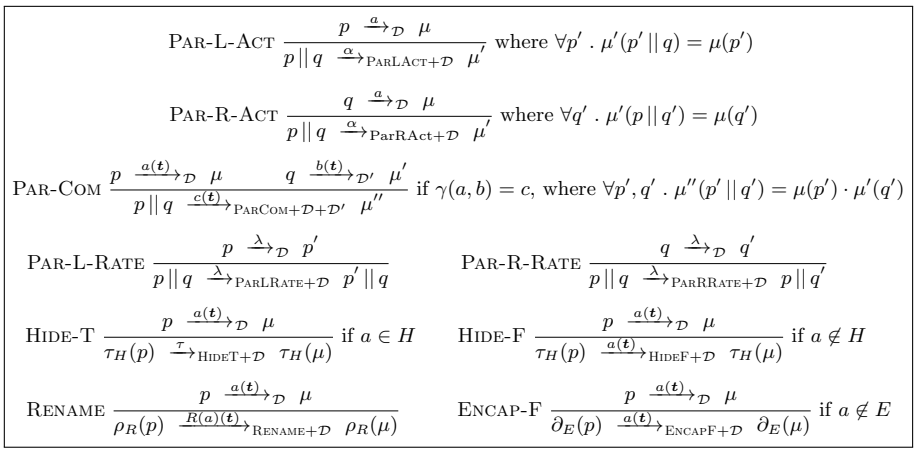

Fig. 8. SOS rules for parallel MAPA.

B.2 Linearisation of parallel processes

The LPPE format allowed processes to be put in parallel very easily, and for the MLPPE format this is no different. Since strong bisimulation is a congruence for parallel composition [6], we can therefore first linearise the components of a parallel composition and then compose the resulting MLPPEs.

Although the MLPPE size is worst-case exponential in the number of parallel processes (when all summands have different actions and all of them communi-

cate), in practice we see only linear growth (having only some actions communicate).

Assume the following two MLPPEs (omitting the initial states):

$$\begin{aligned} X(\boldsymbol{g} : \boldsymbol{G}) = &\sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \;\Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : X(\boldsymbol{n_i}) \\ &+ \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \;\Rightarrow (\lambda_j) \cdot X(\boldsymbol{n_j}) \\ Y(\boldsymbol{g'} : \boldsymbol{G}) = &\sum_{i \in I'} \sum_{\boldsymbol{d'_i} : \boldsymbol{D'_i}} c'_i \;\Rightarrow a'_i(\boldsymbol{b'_i}) \sum_{\boldsymbol{e'_i} : \boldsymbol{E'_i}} f'_i : Y(\boldsymbol{n'_i}) \\ &+ \sum_{j \in J'} \sum_{\boldsymbol{d'_j} : \boldsymbol{D'_j}} c'_j \Rightarrow (\lambda'_j) \cdot Y(\boldsymbol{n'_j}) \end{aligned}$$

Also assuming (without loss of generality) that all global and local variables are named uniquely, the product $Z(\boldsymbol{g} : \boldsymbol{G}, \boldsymbol{g'} : \boldsymbol{G'}) = X(\boldsymbol{g}) \,||\, Y(\boldsymbol{g'})$ is constructed as follows (imitating the construction for the product of two LPPEs):

$$\begin{aligned} Z(\boldsymbol{g} : \boldsymbol{G}, \boldsymbol{g'} : \boldsymbol{G'}) = &\sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : Z(\boldsymbol{n_i}, \boldsymbol{g'}) \\ &+ \sum_{i \in I'} \sum_{\boldsymbol{d'_i} : \boldsymbol{D'_i}} c'_i \Rightarrow a'_i(\boldsymbol{b'_i}) \sum_{\boldsymbol{e'_i} : \boldsymbol{E'_i}} f'_i : Z(\boldsymbol{g}, \boldsymbol{n'_i}) \\ &+ \sum_{(k,l) \in I \gamma I'} \sum_{(\boldsymbol{d_k}, \boldsymbol{d'_l}) : \boldsymbol{D_k} \times \boldsymbol{D'_l}} c_k \wedge c'_l \wedge \boldsymbol{b_k} = \boldsymbol{b'_l} \Rightarrow \\ &\qquad \gamma(a_k, a'_l)(\boldsymbol{b_k}) \sum_{(\boldsymbol{e_k}, \boldsymbol{e'_l}) : \boldsymbol{E_k} \times \boldsymbol{E'_l}} f_k \cdot f'_l : Z(\boldsymbol{n_k}, \boldsymbol{n'_l}) \\ &+ \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot X(\boldsymbol{n_j}, \boldsymbol{g'}) \\ &+ \sum_{j \in J'} \sum_{\boldsymbol{d'_j} : \boldsymbol{D'_j}} c'_j \Rightarrow (\lambda'_j) \cdot Y(\boldsymbol{g}, \boldsymbol{n'_j}) \end{aligned}$$

Here, $I \gamma I'$ is the set of all combinations of summands $(k, l) \in I \times I'$ such that the action a_k of summand k and the action a'_l of summand l can communicate. Formally, $I \gamma I' = \{(k, l) \in I \times I' \mid (a_k, a'_l) \in \text{domain}(\gamma)\}$.

Note that the parallel composition of MLPPEs is almost the same as the parallel composition of LPPEs. The first three sets of summands of Z are even identical; only the last two sets are new. They work in the same way for the Markovian summands as the first two sets work for the interactive summands: all rates can interleave. As there is no synchronisation on rates, no Markovian variant of the third set of summands needs to be added.

The following proposition states that the above construction is indeed correct. The proof is analogous to the proof of the corresponding proposition in [11].

Proposition 1. *For all $\boldsymbol{v} \in \boldsymbol{G}, \boldsymbol{v'} \in \boldsymbol{G'}$, it holds that $Z(\boldsymbol{v}, \boldsymbol{v'}) \equiv X(\boldsymbol{v}) \,||\, Y(\boldsymbol{v'})$.*

B.3 Linearisation of hiding, encapsulation and renaming

For hiding, renaming, and encapsulation, again we can first linearise and then manipulate the MLPPE. For the MLPPE

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

let the MLPPEs $U(\boldsymbol{g})$, $V(\boldsymbol{g})$, and $W(\boldsymbol{g})$, for $\tau_H(X(\boldsymbol{g}))$, $\rho_R(X(\boldsymbol{g}))$, and $\partial_E(X(\boldsymbol{g}))$, respectively, be given by

$$\begin{aligned} U(\boldsymbol{g} : \boldsymbol{G}) = &\sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a'_i(\boldsymbol{b'_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : U(\boldsymbol{n_i}) \\ &+ \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot U(\boldsymbol{n_j}) \\ V(\boldsymbol{g} : \boldsymbol{G}) = &\sum_{i \in I} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a''_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : V(\boldsymbol{n_i}) \\ &+ \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot V(\boldsymbol{n_j}) \\ W(\boldsymbol{g} : \boldsymbol{G}) = &\sum_{i \in I'} \sum_{\boldsymbol{d_i} : \boldsymbol{D_i}} c_i \Rightarrow a_i(\boldsymbol{b_i}) \sum_{\boldsymbol{e_i} : \boldsymbol{E_i}} f_i : W(\boldsymbol{n_i}) \\ &+ \sum_{j \in J} \sum_{\boldsymbol{d_j} : \boldsymbol{D_j}} c_j \Rightarrow (\lambda_j) \cdot W(\boldsymbol{n_j}) \end{aligned}$$

where

$$a'_i = \begin{cases} \tau & \text{if } a_i \in H \\ a_i & \text{otherwise} \end{cases} \quad \boldsymbol{b'_i} = \begin{cases} () & \text{if } a_i \in H \\ \boldsymbol{b_i} & \text{otherwise} \end{cases} \quad a''_i = R(a_i) \quad I' = \{i \in I \mid a_i \notin E\}$$

Note that, again, this construction is completely analogous to the construction of the corresponding LPPEs for prCRL. The only difference is the set of Markovian summands, which is just copied to $U(\boldsymbol{g} : \boldsymbol{G})$, $V(\boldsymbol{g} : \boldsymbol{V})$ and $W(\boldsymbol{g} : \boldsymbol{W})$. After all, hiding, renaming and encapsulation only affect the interactive transitions, and the way they do is identical to the way they affect the transitions of an LPPE. Hence, the following proposition immediately follows from its counterpart in [11].

Proposition 2. *For all* $\boldsymbol{v} \in \boldsymbol{G}$, $U(\boldsymbol{v}) \equiv \tau_H(X(\boldsymbol{v}))$, $V(\boldsymbol{v}) \equiv \rho_R(X(\boldsymbol{v}))$, *and* $W(\boldsymbol{v}) \equiv \partial_E(X(\boldsymbol{v}))$.