Efficient Modelling and Generation of Markov Automata[*]

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

[1]Formal Methods and Tools, Faculty of EEMCS
University of Twente, The Netherlands
timmer@cs.utwente.nl

[2]Software Modeling and Verification Group
RWTH Aachen University, Germany
katoen@cs.rwth-aachen.de

This presentation introduces a process-algebraic framework with data for modelling and generating Markov automata. We show how an existing linearisation procedure for process-algebraic representations of probabilistic automata can be reused to transform systems in our new framework to a special format. This format enables easy state space generation and facilitates the definition of syntactic reduction techniques. We introduce several such techniques, which treat data as well as Markovian and interactive behaviour in a fully symbolic manner. In this way, reductions are obtained on the specification level instead of the model level, reducing state spaces prior to their construction.

1 Introduction

In the past decade, must 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 security, distributed systems, systems biology and networking have been successfully used for dependability and performance analysis.

Recently, a new type of model that captures much more behaviour was introduced: Markov automata (MA) [5, 4, 3]. In addition to nondeterministic and probabilistic choices, MAs also contain Markovian transitions, i.e., transitions subject to an exponentially distributed delay in continuous time. Hence, MAs can be seen as a unification of probabilistic automata (PAs) [13, 14] (containing nondeterministic and probabilistic transitions) and interactive Markov chains (IMCs) [8] (containing nondeterministic and Markovian transitions). They can be used to provide a natural semantics for Generalized Stochastic Petri Nets [11], the domain-specific language AADL [1] and (dynamic) fault trees [2]; i.e., MAs are very general and can be used to describe most behaviour that is modelled these days.

Example 1. Consider a buffer with infinite capacity, that has a $\frac{1}{3}$ probability of having an arrival with rate λ_1 and a $\frac{2}{3}$ probability of having an arrival with rate λ_2. Due to multiple processors the service rate is assumed to increase linearly when the buffer fills, i.e., if there are n tasks in the buffer then the service rate is $n \cdot \mu$. The corresponding MA representation is given in Figure 1.

Note that this example does not yet show the full expressivity of MAs, since there is no nondeterminism present. Also, the probabilistic choices could be encoded in the rates, reducing the model to a CTMC. We chose to keep things simple, however, to be able to demonstrate our language on an easy specification. □

Although several formalisms to specify PAs and IMCs exist, 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 clearly also applies to MAs; therefore, specifications on a higher level than the actual MAs might be essential for syntactic optimisations that already reduce their corresponding state spaces before construction.

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

Submitted to:
QAPL 2012

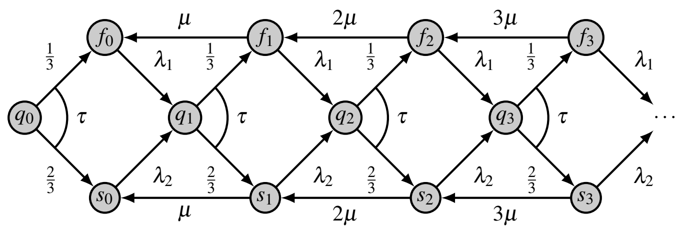

Figure 1: State space of a buffer with probabilistic arrival behaviour.

2 A symbolic representation of Markov Automata

To fulfill the needs described above, we extended the process-algebraic framework for PAs that was introduced in [9, 10]. We show how this framework, that allows the combination of data and probabilistic choice, can easily be generalised to MAs. The process-algebraic specification language for MAs we obtain this way is called MAPA. Due to space limitations of this abstract, we do not show its precise syntax and semantics here, but illustrate the language by means of the following example.

Example 2. The MAPA specification below corresponds to the MA in Figure 1. We chose the concrete values $\lambda_1 = 2$ and $\lambda_2 = 4$, as well as $\mu = 5$. Also, for clarity we added some labelled actions.

$$\begin{aligned}
&\mathit{Arrival} = \mathit{chooseType} \sum_{i:\{1,2\}} \frac{i}{3} : (2\cdot i) \;.\; \mathit{arrive} \;.\; \mathit{Arrival}\\
&\mathit{Queue}(n:\mathbb{N}) = \mathit{incoming} \;.\; \mathit{Queue}(n+1)\\
&\qquad\qquad + n > 0 \Rightarrow (n\cdot 5) \;.\; \mathit{completion} \;.\; \mathit{Queue}(n-1)\\
&\mathit{System} = \partial_{\{\mathit{incoming},\mathit{arrive}\}}(\mathit{Queue}(0) \;||\; \mathit{Arrival})\\
&\gamma(\mathit{arrive},\mathit{incoming}) = \mathit{arrival}
\end{aligned}$$

Notice that the queue itself and the arrival process have been declared separately. The arrival process continuously chooses a type of task, by taking a value i out of the set $\{1,2\}$. Then, if $i = 1$ was chosen (which happens with probability $\frac{i}{3} = \frac{1}{3}$), the process is subject to an exponential delay with rate $2 \cdot i = 2$, after which a task arrives and the process is repeated. If $i = 2$ is chosen (which happens with probability $\frac{2}{3}$), a task arrives after a delay governed by an exponential rate of 4. The queue has a process variable to count the number of tasks. It always accepts incoming tasks, after which it increases its counter. If the counter is positive, it also works on tasks and completes them after a Markovian delay with rate $n \cdot 5$.

The system as a whole consists of the queue (initially empty) in parallel to the arrival process. The γ-statement indicates that the *arrive* and *incoming* actions can synchronise: they can happen at the same time, resulting in the action *arrival*. The ∂-operator encapsulates the individual actions *arrive* and *incoming*, meaning that they cannot occur by themselves in *System*, but only together as an *arrival*.

Note that our data-rich language allowed us to model an infinite system with a finite specification. If a finite system is required, the queue size could be restricted to a certain `max` and an additional nondeterministic choice $n = \texttt{max} \Rightarrow \mathit{drop} \;.\; \mathit{Queue}(\texttt{max})$ could be added. □

Our new language has at least two purposes. First, it enables writing down Markov automata on a higher level. Due to the possibility to use data types as well as parallel composition, these specifications are often much smaller than the automata they model. Second, our language enables several syntactic reductions, similar to those defined for the language μCRL [7] and the probabilistic language prCRL [10]. There, each specification is first *linearised* to a carefully-chosen subset of the language, which simplifies both state space generation and the definition of many syntactic reduction techniques. In prCRL, a specification in this subset is called an LPPE (linear probabilistic process equation), and we defined a similar subset of MAPA, called a Markovian LPPE (MLPPE). Similar to the LPPE, an MLPPE always consists of precisely one process (so parallel processes are encoded into one) that is a nondeterministic choice between process terms containing exactly one action and one recursive process call: the *summands*. Such

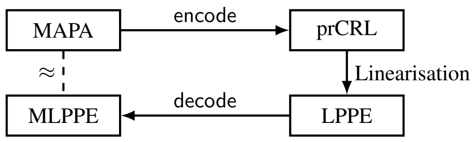

Figure 2: Linearising MAPA specifications using prCRL linerarisation.

a summand can be seen as a symbolic transition, as it unfolds to many concrete transitions.

In [10] we gave a rather complicated algorithm for transforming prCRL specifications to bisimilar LPPEs. Instead of redoing all this work for MAPA, we managed to encode MAPA specifications into prCRL, such that the resulting prCRL specifications can be linearised to LPPEs and these LPPEs can be decoded to MLPPEs. We now proved the correctness of this procedure, i.e., we showed that an original MAPA specification and the MLPPE obtained in this way are strongly bisimilar. Figure 2 illustrates the approach. This has been implemented in the SCOOP tool [16], enabling the specification of MAs in MAPA and exporting the state spaces to the Alderaban format (e.g., to visualise them with CADP [6]).

2.1 Reduction techniques

We showed in [12] and [15] how the LPPE format allows significant optimisations using various reduction techniques, and implemented these in the tool SCOOP [16]. This presentation describes several reduction techniques for MLPPEs; we generalised four existing reduction techniques from LPPEs to MLPPEs, and defined two additional techniques to apply specifically to MAs. An implementation is currently in progress. The generalisation of the more involved confluence reduction [15] is a topic of future work.

Generalisation of existing techniques. *Constant elimination* [10] detects if a parameter of an LPPE never changes its value. Then, the parameter is omitted and every reference to it is replaced by its initial value. Clearly, this reduction technique can be used unchanged for MLPPEs. The same holds for *expression simplification* [10]: we can easily evaluate functions for which all parameters are constants and applying basic laws from logic. *Summation elimination* [10] can also be applied to MLPPEs, but only in a careful manner. A term of the form $\sum_{d:\mathbb{N}} d = 5 \Rightarrow \mathit{send}(d)$ can clearly be changed to $\mathit{send}(5)$, but the LPPE reduction from for instance $\sum_{d:\{1,2,3\}} (\lambda) \cdot \mathit{finish}$ to $(\lambda) \cdot \mathit{finish}$ is not valid anymore.

The techniques above do not change the state space, but improve readability and speed up state space generation. *Dead-variable reduction* [12] 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. Then, it is reset to its initial value. This technique works for MLPPEs just as for LPPEs.

Novel reduction techniques. Due to the maximal progress assumption, no Markovian transition can be taken from a state that also allows a τ-transition. Hence, such Markovian transitions can safely be omitted, including all states that are only reachable by such a transition. This *maximal progress reduction* can be applied during state space generation, but is more efficient on the MLPPE level: we just omit all Markovian summands that are always enabled together with non-Markovian summands. Note that, to detect such scenarios, some kind of automatic theorem proving has to be applied, as in [12].

Additionally, if a Markovian transition is followed by an invisible transition, these can be merged. For instance, in Figure 1 each transition $f_i \xrightarrow{\lambda_1} q_{i+1}$ can be merged with the τ-transition leaving q_{i+1}, replacing it by the two transitions $f_i \xrightarrow{1/3\lambda_1} f_{i+1}$ and $f_i \xrightarrow{2/3\lambda_1} s_{i+1}$. Doing the same for the λ_2-transitions, all states q_i for $i > 0$ can be omitted. Such *transition merging* can also be performed on an MLPPE. Moreover, terms such as $\sum_{d:\{1,2,3\}} (\lambda) \cdot \mathit{finish}$ can be changed to $(3\lambda) \cdot \mathit{finish}$: the variable d is not used in the argument of the summation, but does cause a multiplication of the Markovian transitions.

3 Conclusions

We introduced MAPA, a new process-algebraic language for specifying Markov automata. It enables easy modelling and state space generation, as well as reduction techniques that are simplified by first linearising to an MLPPE. We defined an encoding of MAPA to the existing probabilistic process-algebraic language prCRL, and proved that the linearisation procedure of prCRL can be reused to linearise MAPA specifications. This shows the versatility of our framework.

Based on the MLPPE format, several reduction techniques can be defined. We already generalised four such techniques from prCRL to MAPA, and defined two novel ones. Note that these techniques can now also be applied to any GSPN, IMC, DTMC or CTMC, as each of these can be modelled as an MA.

References

[1] International Society of Automotive Engineers (2004): *Architecture Analysis and Design Language (AADL)*. SAE Standard AS5506.

[2] H. Boudali, P. Crouzen & M. I. A. Stoelinga (2007): *Dynamic Fault Tree Analysis Using Input/Output Interactive Markov Chains*. In: *Proc. of the 37th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, IEEE Computer Society, pp. 708–717.

[3] Y. Deng & M. Hennessy (2011): *On the Semantics of Markov Automata*. In: *Proc of the 38th Int. Coll. on Automata, Languages and Programming (ICALP)*, *LNCS* 6756, Springer, pp. 307–318.

[4] C. Eisentraut, H. Hermanns & L. Zhang (2010): *Concurrency and Composition in a Stochastic World*. In: *Proc. of the 21th Int. Conf. on Concurrency Theory (CONCUR)*, *LNCS* 6269, Springer, pp. 21–39.

[5] C. Eisentraut, H. Hermanns & L. Zhang (2010): *On Probabilistic Automata in Continuous Time*. In: *Proc. of the 25th Annual IEEE Symp. on Logic in Computer Science (LICS)*, IEEE Computer Society, pp. 342–351.

[6] H. Garavel, F. Lang, R. Mateescu & W. Serwe (2011): *CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes*. In: *Proc. of the 17th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, *LNCS* 6605, Springer, pp. 372–387.

[7] J. F. Groote & A. Ponse (1995): *The syntax and semantics of μCRL*. In: *Proc. of Algebra of Communicating Processes*, Workshops in Computing, Springer, pp. 26–62.

[8] H. Hermanns (2002): *Interactive Markov Chains: The Quest for Quantified Quality*. *LNCS* 2428, Springer.

[9] J.-P. Katoen, J. C. van de Pol, M. I. A. Stoelinga & M. Timmer (2010): *A linear process-algebraic format for probabilistic systems with data*. In: *Proc. of the 10th Int. Conf. on Application of Concurrency to System Design (ACSD)*, IEEE, pp. 213–222.

[10] J.-P. Katoen, J.C. van de Pol, M.I.A. Stoelinga & M. Timmer (2012): *A linear process-algebraic format with data for probabilistic automata*. *Theoretical Computer Science* 413(1), pp. 36–57.

[11] M. A. Marsan, G. Conte & G. Balbo (1984): *A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems*. *ACM Transactions on Computer Systems (TOCS)* 2(2), pp. 93–122.

[12] J. C. van de Pol & M. Timmer (2009): *State Space Reduction of Linear Processes using Control Flow Reconstruction*. In: *Proc. of the 7th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, *LNCS* 5799, Springer, pp. 54–68.

[13] R. Segala (1995): *Modeling and Verification of Randomized Distributed Real-Time Systems*. Ph.D. thesis, MIT.

[14] M. I. A. Stoelinga (2002): *An introduction to probabilistic automata*. *Bulletin of the EATCS* 78, pp. 176–198.

[15] M. Timmer, M. I. A. Stoelinga & J. C. van de Pol (2011): *Confluence Reduction for Probabilistic Systems*. In: *Proc. of the 17th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, *LNCS* 6605, Springer, pp. 311–325.

[16] Mark Timmer (2011): *SCOOP: A Tool for SymboliC Optimisations of Probabilistic Processes*. In: *Proc. of the 8th Int. Conf. on Quantitative Evaluation of Systems (QEST)*, IEEE Computer Society, pp. 149–150.