PARAM: A Model Checker for Parametric Markov Models

Ernst Moritz Hahn[1], Holger Hermanns[1,2], Björn Wachter[1], and Lijun Zhang[3]

[1] Saarland University – Computer Science, Saarbrücken, Germany
[2] INRIA Grenoble – Rhône-Alpes, France
[3] DTU Informatics, Technical University of Denmark, Denmark

Abstract. We present PARAM 1.0, a model checker for parametric discrete-time Markov chains (PMCs). PARAM can evaluate temporal properties of PMCs and certain extensions of this class. Due to parametricity, evaluation results are polynomials or rational functions. By instantiating the parameters in the result function, one can cheaply obtain results for multiple individual instantiations, based on only a single more expensive analysis. In addition, it is possible to post-process the result function symbolically using for instance computer algebra packages, to derive optimum parameters or to identify worst cases.

1 Introducing PARAM

Markov processes are applied in computer science, engineering, mathematics, and biology. In the early design phase of a system or for the sake of robust modelling, it can be advantageous to leave certain aspects unspecified and use symbolic parameters rather than fixed values. We consider *parametric* Markov chains (PMCs) [1], where e.g., certain transition probabilities are symbolic parameters. As a result, the analysis of properties, such as the probability of reaching a set of goal states, yields symbolic expressions [2,3] rather than concrete values. These symbolic expressions are represented by multivariate rational functions, i.e. fractions where numerator and denominator are polynomials in the model parameters. To arrive at these results, PARAM uses efficient techniques to manipulate and represent polynomials combined with dedicated state-lumping techniques. The basic analysis provided by PARAM is the computation of step-bounded and unbounded reachability probabilities for PMCs, but it also can handle several extensions of this analysis, as we will explain below.

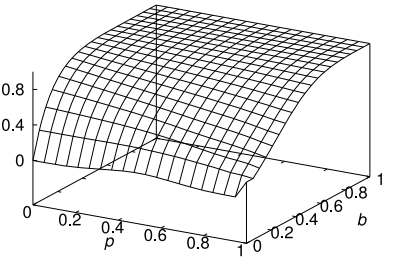

Fig. 1. Crowds Reliability

Consider the Crowds protocol [4,5] where communication is hidden via random routing to protect the anonymity of Internet users. Assume we have n honest Crowd members and m dishonest members. Further, assume that a Crowd member is untrustworthy with probability $b = m/(m+n)$ and denote the probability for random routing (instead of sending directly to the final receiver) by p. We

T. Touili, B. Cook, and P. Jackson (Eds.): CAV 2010, LNCS 6174, pp. 660–664, 2010.

are interested in the probability q that the untrustworthy members observe the original sender more often than any other participant. In practical applications, the Crowds protocol is deployed with many different parameter instantiations, so it is beneficial to explore the resulting range of behaviours with PARAM.

The Crowds protocol can be conveniently modelled in the input language of PARAM, a variation of the language of the PRISM [6] model checker. From the PMC model, PARAM automatically computes a rational function in the parameters of the model. Assume we have $n = 5$ honest members and $r = 7$ rounds of the protocol. Then, the result is a rational function involving multivariate polynomials of degree 21. We can export this function to a computer algebra package for post-processing, e.g. to find optimum parameter settings. In Fig. 1, we plot the dependency of q from the probability b of untrustworthiness of a member and the forwarding probability p. To arrive at this plot, we have evaluated the rational function at 50*60 parameter instances. We could, of course, have solved a total of 3000 non-parametric model checking problems instead to arrive at the same plot (which takes more time). For $n = r = 2$ the formula is:

$$q(p,b) = \frac{p^2b^4 + 2p^2b^3 - 7p^2b^2 + 4p^2b + 12pb^2 - 12pb - 4b^2 + 8b}{4p^2b^2 - 8p^2b + 4p^2 + 8pb - 8p + 4}$$

PARAM can handle the Crowds model up to roughly $n = 10, r = 8$, leading to a state-space of about 2.5 million states.

The method of computing rational functions is inspired by Daws [7], who treats a PMC as a finite automaton and computes the regular expression of its accepted language [8], from which the rational function is obtained. Our method instead works directly on rational functions, and simplifies them on-the-fly. In doing so, we can exploit symmetries, cancellations and simplifications of arithmetic expressions, especially if most of the transition probabilities of the input model are constants. Experimental evidence shows that this results in drastically smaller rational functions during intermediate computations.

2 Architecture

The architecture and components of PARAM are depicted in Fig. 2.

First, the state-space exploration component generates an explicit graph representation from the symbolic description of the model and property. Thereby it leverages property-dependent optimisations to reduce the number of states, in order to accelerate subsequent steps.

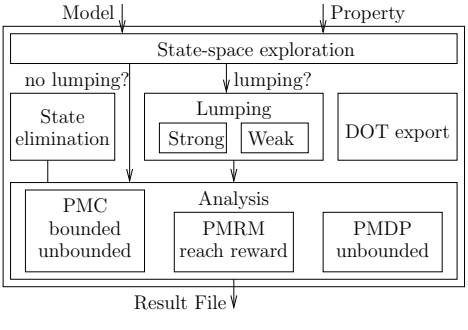

Fig. 2. PARAM Architecture

The lumping [9,10] component leverages bisimulation equivalence to minimise the model. It selects the adequate property-preserving bisimulation relation. The lumping component is

modular and can easily be adapted. It can lead to dramatic speed-ups. For parametric Markov decision processes (PMDPs), lumping maintains only maximal reachability [2,3].

To produce the final result in the form of a rational function, the analysis component uses a variant of state elimination and operations on polynomials. Three dedicated engines cater to specific variants of parametric models: PMCs and extended models with rewards or non-determinism.

3 Selected Features

The algorithmic basis for our tool has been laid out in a previous publication [2] together with a prototypical implementation. PARAM 1.0 [3] improves and enhances this prototype in many aspects. We give an overview of important distinguishing features below.

SPIN-like model exploration. Our prototype suffered from a rather inefficient low-level state-space generation. Using a SPIN-like [11] precompilation technique, we improved the speed of the state-space generation by a factor of about ten. To this end, we first convert the PRISM model into a C++ library, compile it and generate the state-space using the library.

Reward models. In addition to PMCs, PARAM 1.0 can also handle parametric Markov *reward* models (PMRMs) in which states and transitions are additionally equipped with reward structures, and these reward structures can be specified as parameters. For PMRMs, we consider reachability rewards, that is, the accumulated rewards till a certain set of target states is reached.

Nondeterministic models. PARAM 1.0 supports PMDPs, which involve both parametric probabilistic choice and nondeterminism. We are interested in the *maximum* probability of reaching a given set of target states. PARAM first reduces the problem to reachability over PMCs, by encoding the nondeterministic choices as additional parameters, and then submits the problem to the engine for PMCs. Admittedly, this approach does not scale, since the analysis is exponential in the number of nondeterministic states.

Improved data structures. The prototype of PARAM relied on data structures provided by the arithmetic library CoCoALib [12]. PARAM 1.0 instead uses dedicated data structures and a novel, memory-efficient implementation of rational functions, to avoid costly conversions between PARAM and CoCoALib. CoCoALib is still used to cancel rational functions, as algorithms to do so are very complicated to implement in an efficient manner. PARAM 1.0 tries to call the cancellation routine only if cancellation is detected to be possible.

Sharing. PARAM 1.0 uses a global table of rational functions. The same rational function is only stored once. Rational functions attached to state transitions are references into this table. This is advantageous, because experiments show

that rational functions have a tendency of becoming quite large during the state elimination. At the same time, intermediate computations often result in identical rational functions. To exploit sharing, we use a fast hashing mechanism to find such rational functions in the table. This is important, as table lookups are frequent. In the prototype, this feature was not yet implemented.

Value Cache. Arithmetic operations on rational functions are much more costly than for floating-point values. To avoid redundant re-computation, we have implemented a value cache which stores operands and results of operations. Our experiments have shown that, for our current implementation of state elimination, the value cache is only useful during lumping. Because of this, our implementation of rational functions allows to toggle this feature, for different parts of the analysis. This feature existed in the prototype, but is improved in release 1.0.

Lumping Bisimulation. PARAM 1.0 features an efficient implementation of lumping algorithms. Using signature-based lumping [13] is the key to an efficient implementation. Currently, weak and strong bisimulation for PMCs are implemented. The mechanism is written in a modular way, allowing future integration of other lumping techniques into the framework. Since operations on rational functions are rather expensive, this feature is very important in practice. In certain cases, this component can be used for PMDPs.

4 Concluding Remarks

PARAM 1.0 consists of approximately 5000 lines of C++ code, and has been tested on a large number of case studies. It is available for Linux with libc6. The source code is published under the GPL license. The source code and several case studies can be downloaded from `http://depend.cs.uni-sb.de/tools/param`

Acknowledgement. This work is supported by the NWO-DFG bilateral project ROCKS, by the DFG as part of the Transregional Collaborative Research Centre SFB/TR 14 AVACS and the GRK 623 and has received funding from the European Community's Seventh Framework Programme under grant agreement n^o 214755. The work of Lijun Zhang is partially supported by MT-LAB, a VKR Centre of Excellence. Part of this work was done while Lijun Zhang was at Saarland University and Computing Laboratory, Oxford University.

References

1. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Parametric probabilistic transition systems for system design and analysis. FAC 19, 93–109 (2007)
2. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. In: Păsăreanu, C.S. (ed.) SPIN. LNCS, vol. 5578, pp. 88–106. Springer, Heidelberg (2009)
3. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. STTT (2010) doi: 10.1007/s10009-010-0146-x

4. Reiter, M.K., Rubin, A.D.: Crowds: anonymity for web transactions. ACM Trans. Inf. Syst. Secur. 1, 66–92 (1998)
5. Shmatikov, V.: Probabilistic model checking of an anonymity system. Journal of Computer Security 12, 355–377 (2004)
6. Hinton, A., Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 441–444. Springer, Heidelberg (2006)
7. Daws, C.: Symbolic and parametric model checking of discrete-time Markov chains. In: Liu, Z., Araki, K. (eds.) ICTAC 2004. LNCS, vol. 3407, pp. 280–294. Springer, Heidelberg (2005)
8. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation, SIGACT News, 2nd edn., vol. 32, pp. 60–65 (2001)
9. Derisavi, S., Hermanns, H., Sanders, W.H.: Optimal state-space lumping in Markov chains. Inf. Process. Lett. 87, 309–315 (2003)
10. Baier, C., Hermanns, H.: Weak bisimulation for fully probabilistic processes. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 119–130. Springer, Heidelberg (1997)
11. Holzmann, G.: SPIN model checker, The: primer and reference manual. Addison-Wesley Professional, Reading (2003)
12. Abbott, J.: The design of CoCoALib. In: Iglesias, A., Takayama, N. (eds.) ICMS 2006. LNCS, vol. 4151, pp. 205–215. Springer, Heidelberg (2006)
13. Derisavi, S.: Signature-based symbolic algorithm for optimal Markov chain lumping. In: QEST, pp. 141–150 (2007)