A Platform for High Performance Statistical Model Checking – PLASMA

Cyrille Jegourel, Axel Legay and Sean Sedwards*

INRIA Rennes – Bretagne Atlantique

Abstract. Statistical model checking offers the potential to decide and quantify dynamical properties of models with intractably large state space, opening up the possibility to verify the performance of complex real-world systems. Rare properties and long simulations pose a challenge to this approach, so here we present a fast and compact statistical model checking platform, PLASMA, that incorporates an efficient simulation engine and features an importance sampling engine to reduce the number and length of simulations when properties are rare. For increased flexibility and efficiency PLASMA compiles both model and property into bytecode that is executed on an in-built memory-efficient virtual machine.

1 Introduction

The need to provide accurate predictions about the behaviour of complex systems is increasingly urgent. With computational power ever-more affordable and compact, man-made systems are inevitably becoming increasingly computerised, distributed and concurrent, creating a correspondingly increased burden to check that they function correctly. At the same time, following the success of the human genome project, there is an increased expectation that computers can provide answers to important questions raised by complex systems in the life sciences.

Complex systems tend to pose two particular challenges to formal verification: the non-determinism caused by concurrency and unpredictable environmental conditions and the size of the state space. Our focus here is *model checking*, that can verify the most intricate details of a system's dynamical behaviour and where non-determinism may be handled by assigning probabilistic distributions to unknowns and by quantifying results with a probability - *probabilistic model checking*. 'Exact' probabilistic model checking quantifies these probabilities to the limit of numerical precision by an exhaustive exploration of the state space, but is restricted in practise by what can be conveniently stored in memory. Techniques exist to work with a reduced state space (abstraction, lumping, etc.), but the state space of most real natural and man-made systems remain intractable.

Statistical model checking (SMC) avoids an explicit representation of the state space by building a statistical model of the executions of a system and giving results within confidence bounds. An executable model of the system is

* sean.sedwards@inria.fr

run repeatedly and each simulation trace is verified against a property specified in temporal logic. Examples of tools that have successfully applied this approach are [10, 7]. Knowing a result with less than 100% confidence is often sufficient, since the confidence bounds may be made arbitrarily tight, however the key challenges of this approach are to reduce the length (simulation steps and cpu time) and number of simulation traces necessary to achieve a result with given confidence. The current proliferation of parallel computer architectures (multiple cpu cores, grids, clusters, clouds and general purpose computing on graphics processors, etc.) makes the production of multiple independent simulation runs relatively easy, but it is still necessary to make simulation as efficient as possible. Rare properties pose a particular problem for simulation-based approaches, since they are not only difficult to observe (by definition) but their probability is difficult to bound [2].

In what follows we present the prototype of a flexible SMC platform, PLASMA[1], that incorporates an in-built compiler and virtual machine to perform memory- and time-efficient simulations. PLASMA incorporates an efficient discrete event simulation algorithm and features an importance sampling engine that can reduce the necessary number of simulation runs when properties are rare.

2 Software architecture

PLASMA adopts a modular architecture to facilitate the extension of its features (Fig. 1). Models can currently be specified using the PRISM reactive modules syntax [3], but the implementation of other modelling formalisms, such as timed automata and procedural programming languages such as C and Java, is already planned. The input specification is translated into a common intermediate language based on elements (referred to as *simple commands* because they have no explicit synchronisation and no choice of actions) having the structure (*guard*,*rate*,*actions*), where *guard*, *rate* and *actions* are functions over the current state (constants, variables, clocks) of the system. The intermediate language thus expresses the semantics of a system which advances by discrete events: the *guard* enables the command, the *rate* resolves non-determinism between enabled commands (and controls the delay in continuous time systems) and the *actions* update the state of the system. Different input languages may be facilitated by implementing parsers that construct and fill data structures that reflect simple commands. Once the model is represented in the intermediate language it is compiled into an executable form (the *model program*).

PLASMA uses its own in-built compiler to create bytecode for execution on its own stack-based virtual machine (VM) that comprises a logic VM (LogicVM in Fig. 1) and a simulation VM (SimVM in Fig. 1). PLASMA's bytecode instructions constitute a domain-specific, low level, platform-independent language designed for efficient statistical model checking. This language contains standard low level instructions, such as *push*, *pop*, *add*, *sub*, *mul*, *div*, etc., as well

[1] A demonstration version of PLASMA may be downloaded from https://sites.google.com/site/plasmasmc

as non-standard instructions to construct efficient model checking algorithms. The VM is implemented in a high level procedural programming language (currently Java, but the code uses no features that cannot easily be adapted to other languages) and is efficient because it is optimised for its domain of application: high level instructions are efficient sub-parts of model-checking algorithms and all instructions are optimised with respect to the hardware level. The compiler and VM are also sufficiently compact to allow PLASMA to be implemented as a browser application, a distributed component or in an embedded system etc.

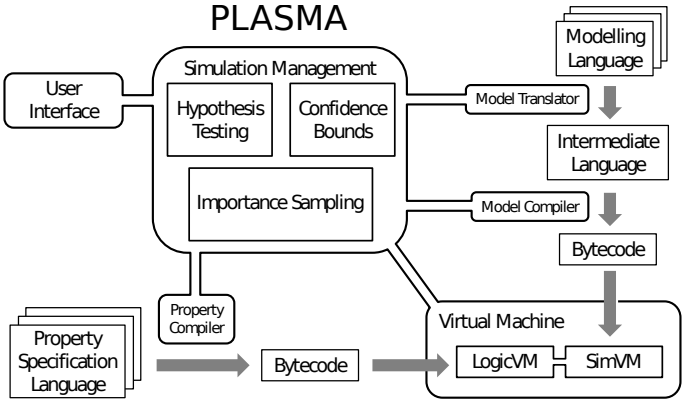

Fig. 1. The architecture of PLASMA

PLASMA verifies properties specified in bounded temporal logic. Such properties are compiled into bytecode programs (*property programs*) and then executed on the logic VM. Our current focus is discrete time, however continuous time and other logics may be easily facilitated by implementing additional logic parser-compilers. Overall control of the verification process is maintained by the simulation management kernel (SMK) according to the options specified by the user. In general, the property program executes the model program until it has seen sufficient steps to decide a result and the SMK executes the property program until it has sufficient results to return an answer to the user. In this way, simulation traces contain the minimum number of states necessary to decide the property and the minimum number of simulations are generated. The logic accepts arbitrarily nested path formulae, however formulae that are not nested are particularly memory efficient: by employing a multivalued logic (*true*, *false*, *undecided*) PLASMA need only store the current state of the system. Nested formulae are also handled efficiently. In general, PLASMA stores only a subset of the full trace, having length equal to the maximum sum of the time bounds of any nested formulae.

2.1 Stochastic simulation algorithm

PLASMA performs discrete event simulation using the 'method of arbitrary partial propensities' (MAPP [6]). The MAPP is based on the Gillespie 'direct method' (DM [8]) but performs significantly better in large-scale practical applications than either the DM or the asymptotically better 'next reaction method' (NRM [1]). In a system of M simple commands, each step of the DM is $\mathcal{O}(M)$ because it iterates through all the commands in the system to find the command to execute and then again to update all the guards and rates following the execution of the chosen command. The MAPP divides the M commands into $\sqrt{M}$ subsets of $\sqrt{M}$ commands and thus divides choosing a command into two operations of $\mathcal{O}(\sqrt{M})$: choosing a subset and choosing a command within the subset. By performing an initial dependency analysis of the system the MAPP avoids updating commands whose guards and rates are not affected during the simulation. Since the average number of dependent commands, D, tends to be smaller than and independent of M, the overall complexity of the MAPP is $\mathcal{O}(\sqrt{M})$. The NRM achieves asymptotic complexity of $\mathcal{O}(\log M)$ by performing a similar dependency analysis and by arranging the commands in an ordered binary tree whose root always contains the next command to be executed. Choosing a reaction is $\mathcal{O}(1)$, but maintaining the invariant property of the tree is proportional to $D \log_2 M$. D is assumed constant, but the fact that it multiplies the complexity of the NRM tends to make the MAPP more efficient in most small to large-scale applications. See Figure 3.

2.2 Rare properties and importance sampling

The process of statistical model checking estimates the probability of a property by verifying the execution paths of multiple independent simulation runs. If Ω is a probability space of traces ($\omega \in \Omega$), $f(\omega)$ the probability measure over Ω and $z(\omega) \in \{0, 1\}$ is a function indicating whether ω satisfies some property, then the expected probability of the property is given exactly by $\gamma = \int_\Omega z(\omega) f(\omega)\ \mathrm{d}\omega$. This leads directly to the standard Monte Carlo estimator: $\tilde{\gamma} = \frac{1}{N} \sum_{i=1}^{N} z(\omega_i^f)$, where ω_i^f are simulation paths under distribution f.

As $\gamma \to 0$, however, it becomes increasingly difficult to bound the relative error in $\tilde{\gamma}$ and N becomes prohibitively large [2]. Importance sampling can be used to reduce N by performing simulations under a 'tilted' (importance sampling) distribution that produces the rare paths more frequently and by compensating for the tilt using the 'likelihood ratio'. If f' is the importance sampling distribution then $l(\omega) = \frac{f(\omega)}{f'(\omega)}$ is the likelihood ratio and $\gamma = \int_\Omega z(\omega) \frac{f(\omega)}{f'(\omega)} f'(\omega)\ \mathrm{d}\omega$. This gives the importance sampling estimator $\tilde{\gamma} = \frac{1}{N} \sum_{i=1}^{N} l(\omega_i^{f'}) z(\omega_i^{f'})$, where $\omega_i^{f'}$ are simulation paths under the the importance sampling distribution and $l(\omega_i^{f'})$ is calculated on the fly.

By individually optimising all the probabilities in the transition system ('state dependent tilting') it is conceivable to create very good importance sampling distributions, however this is not in general tractable. PLASMA therefore adopts a

parameterised (state *independent*) tilting scheme based on its intermediate language representation of the model. For each simple command in the system, an importance sampling parameter taking strictly positive values is introduced to bias the rates. To test the performance of PLASMA's paramterised importance sampling engine we applied it to repair models from [5] that have previously been considered in the context of state dependent tilting and which may be verified by PRISM's numerical algorithm. We have found that our state independent tilting scheme is nevertheless capable of achieving dramatic increases in performance. For instance, example 1 of [5] considers a property with probability 1.17×10^{-7}, requiring an expected 10^8 simulation runs to see a few examples. Using just six parameters PLASMA is able to make a 10^6-fold increase in the frequency of observing the rare event.

3 Results

Figure 2 illustrates typical performance scaling[2] of PLASMA's SMC engine relative to PRISM's numerical algorithm by applying them to increasing instance sizes of a classic probabilistic model checking problem (the randomised dining philosophers protocol of [4]). The state space increases exponentially with respect to instance size, hence PRISM's time scaling is exponential and its maximum instance size is here limited by available memory to about 35 philosophers. By accepting a result with (arbitrarily) bounded error, PLASMA can work with much larger models and its performance scales linearly in time proportional to the length of the property formula. Since PLASMA's memory requirement also scales linearly with instance size, its limit is much higher than the maximum shown in Figure 2.

Figure 3 illustrates typical simulation performance scaling of PLASMA's MAPP algorithm in comparison to the direct method of [8] and PRISM's simulation engine. A stochastic oscillatory model from systems biology [9] was used as the building block to construct plausible biological models of increasing complexity. Using the DM's $\mathcal{O}(M)$ scaling as reference, the lower order scaling of PLASMA's MAPP algoorithm is clear. The performance of PRISM's simulation algorithm is also here limited by memory, but in this case the limit is not related to the state space, which is completely intractable for even the smallest instance.

4 Conclusion and future challenges

PLASMA is a compact, efficient and flexible SMC platform that incorporates a novel importance sampling engine. Its broad goal is to take SMC beyond proof of concept and to tackle the analysis of real-world systems. Since such systems are usually not written in abstract modelling languages, we foresee a need to implement other input languages to avoid errors and make the process of

[2] All results generated under Windows 7 Enterprise 64-bit and Java 1.6.0_26 32-bit on Intel Core i7 CPU M640 @ 2.80Ghz with 4GB RAM. PRISM 4.0.1 was used.

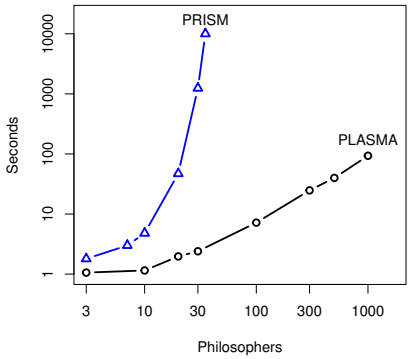

Fig. 2. Performance scaling of PLASMA's SMC engine vs. PRISM's numerical algorithm. PLASMA performed 118595 simulations per point.

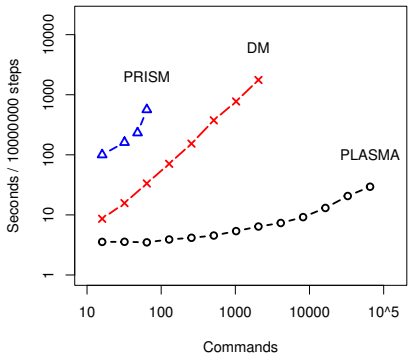

Fig. 3. Simulation algorithm performance scaling using the genetic oscillator of [9] as a building block. The DM was implemented in PLASMA.

verification more convenient. Of particular interest are timed and hybrid systems, since these are commonly used in industrial applications. Importance sampling constitutes a major thread of our research, as it has the potential to dramatically increase the performance of simulation-based techniques. A key challenge is the discovery of good parameterised importance sampling distributions and we are currently developing algorithms to infer these automatically.

References

1. M. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *J. of Physical Chemistry A*, 104:1876, 2000.
2. Philip Heidelberger. Fast simulation of rare events in queueing and reliability models. *ACM Trans. Model. Comput. Simul.*, 5:43–85, January 1995.
3. The PRISM manual. www.prismmodelchecker.org/manual/.
4. A. Pnueli and L. Zuck. Verication of multiprocess probabilistic protocols. *Distributed Computing*, 1:5372, 1986.
5. Ad Ridder. Importance sampling simulations of markovian reliability systems using cross-entropy. *Annals of Operations Research*, 134:119–136, 2005.
6. S Sedwards. A Natural Computation Approach To Biology: Modelling Cellular Processes and Populations of Cells With Stochastic Models of P Systems. PhD thesis, University of Trento, 2009.
7. K Sen, M Viswanathan, and G. A. Agha. Vesta: A statistical model-checker and analyzer for probabilistic systems. pages 251–252. IEEE Computer Society, 2005.
8. Daniel T and Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403 – 434, 1976.
9. M. G. Vilar, H. Y. Kueh, N. Barkai, and S. Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proceedings of the National Academy of Science*, 99, 2002.
10. H. L. S. Younes. Ymer: A statistical model checker. volume 3576, pages 429–433. Springer, 2005.