Comparative Analysis of Statistical Model Checking Tools

Mehmet Emin Bakir[1], Marian Gheorghe[2], Savas Konur[2], and Mike Stannett[1]

[1] Department of Computer Science, The University of Sheffield
Regent Court, 211 Portobello, Sheffield, S1 4DP, UK
mebakir1@sheffield.ac.uk, m.stannett@sheffield.ac.uk
[2] School of Electrical Engineering and Computer Science, University of Bradford
West Yorkshire, Bradford, BD7 1DP, UK
m.gheorghe@bradford.ac.uk, s.konur@bradford.ac.uk

Abstract. Statistical model checking is a powerful and flexible approach for formal verification of computational models, e.g. P systems, which can have very large search spaces. Various statistical model checking tools have been developed, but choosing the most efficient and appropriate tool requires a significant degree of experience, not only because different tools have different modelling and property specification languages, but also because they may be designed to support only a certain subset of property types. Furthermore, their performance can vary depending on the property types and membrane systems being verified. In this paper, we evaluate the performance of various common statistical model checkers based on a pool of biological models. Our aim is to help users select the most suitable SMC tools from among the available options, by comparing their modelling and property specification languages, capabilities and performances.

Keywords: membrane computing, P systems, statistical model checking, biological models, performance benchmarking

1 Introduction

In order to understand the structure and functionality of biological systems, we need methods which can highlight the spatial and time-dependent evolution of systems. To this end, researchers have started to utilize the computational power of machine-executable models, including implementations of membrane system models, to get a better and deeper understanding of the spatial and temporal features of biological systems [21]. In particular, the executable nature of computational models enables scientists to conduct experiments, in silico, in a fast and cheap manner.

The vast majority of models used for describing biological systems are based on ordinary differential equations (ODEs) [10], but researchers have recently started to use *computational models* as an alternative to mathematical modelling. The basis of such models is *state machines*, which can be used to model numerous variables and relate different system states (configurations) to one another [21]. There have been various attempts to model biological systems from a computational point of view, including the use of Boolean networks [31], Petri nets [45], the π-calculus [39], interacting state machines [25], L-systems [38] and variants of P systems (membrane systems) [5, 17, 23, 29, 33, 42]. These techniques are useful for investigating the qualitative features, as are their stochastic counterparts (e.g., stochastic Petri Nets [26] and stochastic P systems [8, 43]) are useful for investigating the quantitative features of computation models. More updated details regarding the use of membrane systems in modelling systems and synthetic biology applications can be found in [22].

Having built a model, the goal is typically to *analyse* it, so as to determine the underlying system's properties. Various approaches have been devised for analysing computational models. One widely used method, for example, based on generating the execution traces of a model, is *simulation*. Although the simulation approach is widely applicable, the large number of potential execution paths in models of realistic systems means that we can often exercise only a fraction of the complete trace set using current techniques. Especially for non-deterministic and stochastic systems each state may have more than one possible successor, which means that different runs of the same basic model may produce different outcomes [6]. Consequently, some computational paths may never be exercised, and their conformance to requirements never assessed.

Model checking is another widely recognized approach for analysis and verification of models, which has been successfully applied both to computer systems and biological system models. This technique involves representing each (desired or actual) property as a temporal logic formula, which is then verified against a model. It formally demonstrates the correctness of a system by means of strategically investigating the whole of the model's state space, considering all paths and guaranteeing their correctness [4, 15, 28]. Model checking has advantages over conventional approaches like simulation and testing, because it checks all computational paths and if the specified property is not satisfied it provides useful feedback by generating a counter-example (i.e. execution path) that demonstrates how the failure can occur [28].

Initially, model checking was employed for analysing *transition systems* used for describing discrete systems. A transition system regards time as discrete, and describes a set of states and the possible transitions between them, where each state represents some instantaneous configuration of the system. More recently, model checking has been extended by adding probabilities to state transitions (*probabilistic model checking*); in practice, such systems include discrete-time Markov chains (DTMC), continuous-time Markov chains (CTMC), and Markov decision processes (MDP). Probabilistic models are useful for verifying quantitative features of systems.

Typically, the model checking process comprises the following steps [4, 28]:

1. Describing the system model in a high-level modelling language, so as to provide an unambiguous representation of the input system.

2. Specifying the desired properties (using a property specification language) as a set of logical statements, e.g., temporal logic formulas.
3. Verifying whether each property is valid on the model. For non-probabilistic models the response is either 'yes' or 'no'. For probabilistic systems the response may instead be some estimate of the 'probability of correctness'.

"Exact" model checking considers whole state spaces while verifying a property, but if the model is relatively large, the verification process can be prohibitively resource intensive and time consuming which is known as 'state-space explosion' problem, so this approach can only be applied to a small number of biological models. Nonetheless, the intrinsic power of the approach has gained a good deal of attention from researchers, and model checking has been applied to various biological phenomena, including, for example, gene regulator networks (GRNs) and signal-transduction pathways [8, 13] (see [20] for a recent survey of the use of model checking in systems biology).

To overcome the state-space explosion problem, the *statistical* model checking (SMC) approach does not analyse the entire state space, but instead generates a number of independent simulation traces and uses statistical (e.g., Monte Carlo) methods to generate an approximate measure of system correctness. This approach does not guarantee the absolute correctness of the system, but it allows much larger models be verified (within specified confidence limits) in a faster manner [12, 37, 49, 51]. This approach allows verifying much larger models with significantly improved performance.

The number of tools using statistical model checking has been increasing steadily, as has their application to biological systems [14, 53]. Although the variety of SMC tools gives a certain amount of flexibility and control to users, each model checker has its own specific pros and cons. One tool may support a large set of property operators but perform property verifications slowly, while another may be more efficient at analysing small models, and yet another may excel at handling larger models. In such cases, the user may need to cover all of their options by using more than one model checker, but unfortunately the different SMCs generally use different modelling and property specification languages. Formulating properties using even a single SMC modelling language can be a cumbersome, error-prone, and time wasting experience for non-experts in computational verification (including many biologists), and the difficulties multiply considerably when more than one SMC needs to be used.

In order to facilitate the modelling and analysis tasks, several software suites have been proposed, such as Infobiotics Workbench [9] (based on stochastic P systems [10]) and kPWorkbench framework (based on kernel P systems [17]) [17, 34]. As part of the computational analysis, these tools employ more than one model checker. Currently, they allow only a manual selection of the tools, relying on the user expertise for the selection mechanism. These systems automatically translate the model and queries into the target model checker's specification language. While this simplifies the checking process considerably, one still has to know which target model checker best suits ones needs, and this requires a significant degree of experience. It is desirable, therefore, to introduce another processing layer, so as to reduce human intervention by *automatically* selecting the best model checker for any given combination of P system and property query.

As part of this wider project (Infobiotics Workbench) to provide machine assistance to users, by automatically identifying the best model checker, we evaluate the performance of various statistical model checkers against a pool of biological models. The results reported here can be used to help select the most suitable SMC tools from the available options, by comparing their modelling and property specification languages, capabilities and performances (see also [7]).

Paper structure. We begin in Section 2 by describing some of the most commonly used SMC tools, together with their modelling and property-specification languages. Section 3 compares the usability of these tools in terms of expressibility of their property specification languages. In Section 4 we benchmark the performance of these tools when verifying biological models, and describe the relevant experiment settings. We conclude in Section 5 with a summary of our findings, and highlight open problems that warrant further investigation.

2 A Brief Survey of Current Statistical Model Checkers

In this section, we review some of the most popular and well-maintained statistical model checking tools, together with their modelling and property specification languages.

2.1 Tools

PRISM. PRISM (Probabilistic and Symbolic Model Checker) is a widely-used, powerful probabilistic model checker tool [27, 35]. It has been used for analysing a range of systems including biological systems, communication, multimedia and security protocols and many others [46]. It allows building and analysing several types of probabilistic systems including discrete-time Markov chains (DTMCs) and continuous-time Markov chains (CTMCs) with their 'reward' extension. PRISM can carry out both probabilistic model checking based on numerical techniques with exhaustive traversal of model, and statistical model checking with a discrete-event simulation engine [36, 46]. The associated modelling language, the PRISM language (a high-level state-based language), is the probabilistic variant of Reactive Modules [1, 35] (for a full description of PRISMs modelling language, see [46]), which subsumes several property specification languages, including PCTL, PCTL*, CSL, probabilistic LTL. However, statistical model checking can only be applied to a limited subset of properties; for example, it does not support steady-state and LTL-style path properties.

PRISM can be run via both a Graphical User Interface (GUI) or directly from the command line. Both options facilitate model checking process by allowing to modify a large set of parameters. The command line option is particularly useful when users need to run a large number of models. PRISM is open source software and is available for Windows, Linux and Mac OS X platforms.

PLASMA-Lab. PLASMA-Lab is a software platform for statistical model checking of stochastic systems. It provides a flexible plug-in mechanism which allows users to personalise their own simulator, and it also facilitates distributed simulations [11]. The tool has been applied to a range of problems, such as systems biology, rare events, motion planning and systems of systems [44].

The platform supports four modelling languages: Reactive Module Language (RML) implementation of the PRISM tool language, with two other variants of RML (see Table 1), and Biological Language [11, 44]. In addition, it provides a few simulator plug-ins which enable external simulators to be integrated with PLASMA-Lab, e.g., MATLAB/Simulink. The associated property specification language is based on Bounded Linear Temporal Logic (B-LTL) which bounds the number of states by number of steps or time units.

PLASMA-Lab can be run from a GUI or command line with plug-in system, and while it is not open source it can be embedded within other software programs as a library. It has been developed using the Java programming language, which provides compatibility with different operating systems.

Ymer. Ymer is a statistical model checking tool for verifying continuous-time Markov chains (CTMCs) and generalized semi-Markov processes (GSMPs). The tool supports parallel generation of simulation traces, which makes Ymer a fast SMC tool [50].

Ymer uses the PRISM language grammar for its modelling and property specification language. It employs the CSL formalism for property specification [48].

Ymer can be invoked via a command line interface only. It has been developed using the C/C++ programming language, and the source code is open to the public.

MRMC. MRMC is a tool for numerical and statistical model checking of probabilistic systems. It supports DTMC, CTMC, and using the reward extension of DTMC and CTMC [30].

The tool does not employ a high-level modelling language, but instead requires a sparse matrix representation of probabilities or rates as input. Describing systems in transition matrix format is very hard, especially for large systems, and external tools should be used to automatically generate the required inputs. Both PRISM and Performance Evaluation Process Algebra (PEPA) have extensions which can generate inputs for the MRMC tool [52]. The matrix representation also requires that state labels with atomic propositions be provided in another structure. Properties can be expressed with PCTL and CSL, and with their reward extensions.

MRMC is a command line tool. It has been developed using the C programming language, and the source code is publicly available. Binary distributions for Windows, Linux and Mac OS X are also available [41].

MC2. The MC2 tool enables statistical model checking of simulation traces, and can perform model checking in parallel.

MC2 does not need a modelling language, instead it imports simulation traces generated by external tools for stochastic and deterministic models. The tool uses probabilistic LTL with numerical constraints (PLTLc) for its property specification language, which enables defining numerical constraints on free variables [16].

MC2 can be executed only through its command line interface. The tool was developed using the Java programming interfaces and is distributed as a `.jar` file, therefore the source code is not available to public. The tool is bundled with a Gillespie simulator, called *Gillespie2*. As will be explained in the following section, it is possible to use Gillespie2 to generate simulation traces for the MC2 tool.

2.2 Modelling Languages

As part of the model checking process the system needs to be described in the target SMC modelling language. If the SMC tool relies on external tools, as in the case of MRMC and MC2, users will also have to learn the usage and modelling language of these external tools as well. For example, if users want to use the MRMC tool, they also have to learn how to use PRISM and how to model in the PRISM language.

Table 1 summarises the modelling languages associated with each SMC tool. The PLASMA and Ymer tools provide fair support for the PRISM language. MRMC expects a transition matrix input, but in practice, for large models, it is not possible to generate the transition matrix manually, so an external tool should be used for generating the matrix. MC2 also relies on external tools, because it does not employ a modelling language, instead it expects externally generated simulation traces. If users want to use the MC2 tool, they first have to learn a modelling language and usage of an appropriate simulation tool. For example, in order to use the Gillespie2 simulator as an external tool for MC2, the user should be able to describe their model using the Systems Biology Markup Language (SBML).

3 Usability

Model checking uses *temporal logics* as property specification languages. In order to query probabilistic features, probabilistic temporal logics should be used. Several probabilistic property specification languages exist, such as Probabilistic Linear Temporal Logic (PLTL) [4], probabilistic LTL with numerical constraints (PLTLc) [16] and Continuous Stochastic Logic (CSL) [2, 3, 36].

In order to ease the property specification process, frequently used properties, called *patterns*, have been identified by previous studies [18, 24]. Patterns represent recurring properties (e.g., something is *always* the case, something is

Table 1. Modelling languages and external dependency of SMC tools.

SMCs	**Modelling Language(s)**	**Needs an External Tool?**	**External Tool Modelling Language**
PRISM	PRISM language	NO	N/A
PLASMA-Lab	RML of PRISM, Adaptive RML (extension of RML for adaptive systems), RML with importance sampling, Biological Language	NO	N/A
Ymer	PRISM language	NO	N/A
MRMC	Transition matrix	YES, e.g., PRISM	PRISM language
MC2	N/A	YES, e.g., Gillespie2	Systems Biology Markup Language (SBML)

possibly the case), and are generally represented by natural language-like keywords. An increasing number of studies have been conducted to identify appropriate pattern systems for biological models [23, 32, 40]. Table 2 lists various popular patterns [24], giving a short description and explaining how they can be represented using existing temporal logic operators.

Table 2. Property patterns

Patterns	**Description**	**Temporal Logic**
Existence	ϕ_1 will eventually hold, within the $\bowtie p$ bounds.	$P_{\bowtie p}[F\ \phi_1]$ or $P_{\bowtie p}[\mathit{true}\ \mathrm{U}\ \phi_1]$
Until	ϕ_1 will hold continuously until ϕ_2 eventually holds, within the $\bowtie p$ bounds.	$P_{\bowtie p}[\phi_1\ \mathrm{U}\ \phi_2]$
Response	If ϕ_1 holds, then ϕ_2 must hold within the $\bowtie p$ bounds.	$P_{\geq 1}[G\ (\phi_1 \rightarrow (P_{\bowtie p}[F\ \phi_2]))]$
Steady-State (Long-run)	In the long-run ϕ_1 must hold, within the $\bowtie p$ bounds.	$S_{\bowtie p}[\phi_1]$ or $P_{\bowtie p}[FG\ (\phi_1)]$
Universality	ϕ_1 continuously holds, within the $\bowtie p$ bounds.	$P_{\bowtie p}[G\ \phi_1]$ or $P_{\overline{\bowtie}(1-p)}[(F\ (\neg\phi_1)]$

Key. ϕ_1, and ϕ_2 are state formulas; $\bowtie$ is one of the relations in $\{<, >, \leq, \geq\}$; $p \in [0, 1]$ is a probability with rational bounds; and $\overline{\bowtie}$ is negation of inequality operators. $P_{\bowtie p}$ is the *qualitative* operator which enables users to query qualitative features, those whose result is either 'yes' or 'no'. In order to query *quantitative* properties, $P_{=?}$ (quantitative operator) can be used to returns a numeric value which is the probability that the specified property is true.

The SMCs investigated here employ different grammar syntaxes for property specification, which makes it harder to use other tools at the same time. Although Ymer uses the same grammar as PRISM, it excludes some operators, such as the Always (G) operator. In addition, different SMCs tools may support different sets of probabilistic temporal logics. In the following, we compare the expressibility of their specification languages, by checking if the properties can be defined using just one temporal logic operator (directly supported (DS)), which will be easier for practitioners to express; or as a combination of multiple operators (indirectly supported (IS)); or not supported at all (not supported (NS)). Qualitative and quantitative operators, with five property patterns which are identified as widely used by [24], are listed in Table 3.

Table 3. Specifying various key patterns using different SMC tools.

SMCs	**Qualitative Operator**	**Quantitative Operator ($\mathbf{P}_{=?}$)**	**Existence**	**Until**	**Response**	**Steady -State**	**Universality**
PRISM	DS	DS	DS	DS	**NS**	**NS**	DS
PLASMA-Lab	**NS**	**NS**	DS	DS	IS	IS	DS
Ymer	DS	DS	DS	DS	**NS**	**NS**	IS
MRMC	DS	**NS**	DS	DS	IS	DS	DS
MC2	DS	DS	DS	DS	IS	IS	DS

Key. DS = Directly Supported; IS = Indirectly Supported; NS = Not Supported.

The PRISM, Ymer and MC2 tools directly support both Qualitative and Quantitative operators, but MRMC supports only the Qualitative operator. While PLASMA-Lab does not allow these operators to be expressed directly with B-LTL, the verification outputs contain information about the probability of the property, hence users can interpret the results. Existence, Until and Universality properties are directly supported by all SMCs, except that Ymer does not employ an operator for Universality patterns (it needs to be interpreted using the Not (!) and Eventually (F) operators, i.e. it is indirectly supported). There is no single operator to represent the Response pattern directly, but it is indirectly supported by PLASMA-Lab, MRMC and MC2. The Steady-State pattern can be either represented by one operator, S, or two operators, F and G. Only the MRMC tool employs the S operator to allow Steady-State to be expressed directly, while PLASMA-Lab and MC2 allow it to be expressed indirectly.

4 Experimental Findings

The wide variety of SMC tools gives a certain flexibility and control to users, but practitioners need to know which of the tools is the most suitable for their particular models and queries. The expressive power of the associated modelling and specification languages is not the only criterion, because SMC performance may also depend on the nature of the models and property specifications. We have therefore conducted a series of experiments to determine the capabilities and performances of the most commonly used tools [7]. The experiments are conducted on an Intel i7-2600 CPU @ 3.40GHz 8 cores, with 16GB RAM running on Ubuntu 14.04.

We tested each of the five tools against a representative selection of 465 biological models (in SBML format) taken from the BioModels database [19] (as modified in [47] to fix the stochastic rate constants of all reactions to 1). The models tested ranged in size from 2 species and 1 reaction, to 2631 species and 2824 reactions. Figure 1 shows the distribution of models size, we take “size” to be the product of species count and reaction count. X-axis (log scale) indicates the model size and Y-axis represents the frequency of models with their sizes represented on the X-axis.

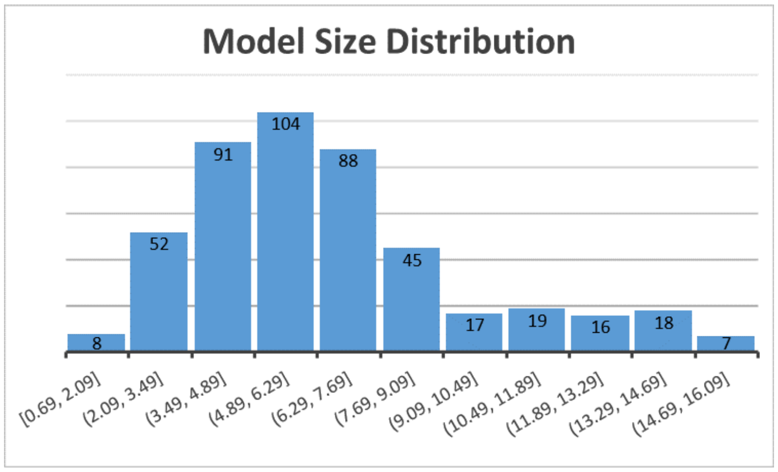

Fig. 1. The distribution of models size in the logarithmic scale.

Each tool/model pair was tested against five different property specification patterns [24], namely Existence, Until, Response, Steady-State and Universality. We have developed a tool for translating SBML models to SMC modelling languages, and translating property patterns to the corresponding SMC specification languages. For each SMC, the number of simulation traces was set to 500, and the depth of each trace was set to 5000.

The time required for each run is taken to be the combined time required for model parsing, simulation and verification. Each SMC/model/pattern combination was tested three times, and the figures reported here give the average total time required. When an SMC depends on external tools, we also added the external tool execution time into the total execution time. In particular, therefore, the total times reported for verifying models with MRMC and MC2 tools are not their execution times only, but include the time consumed for generating transition matrices and simulation traces, respectively. We used the PRISM tool for generating transition matrices requested by MRMC, and the Gillespie2 for generating simulation traces utilised by MC2. When the external tool failed to generate the necessary input for its corresponding SMC, we have recorded the SMC as being incapable of verifying the model. In order to keep the experiment tractable, when an SMC required more than 1 hour to complete the run, we halted the process and again recorded the model as unverifiable.

Table 4 shows the experiment results. The SMCs and the property patterns are represented in the first column and row, respectively. The *Verified* columns under each pattern show the number of models that could be verified by the corresponding SMC. The *Fastest* column shows the number of models for which the corresponding SMC was the fastest tool.

Table 4. The number of model/pattern combinations verified by each SMC tool.

	Existence		**Until**		**Response**		**Steady -State**		**Universality**	
	Verified	**Fastest**	Verified	**Fastest**	Verified	**Fastest**	Verified	**Fastest**	Verified	**Fastest**
PRISM	337	15	435	84	NS	NS	NS	NS	370	57
PLASMA -Lab	465	143	465	54	465	390	465	392	465	80
Ymer	439	304	439	324	NS	NS	NS	NS	439	325
MRMC	75	0	72	0	75	17	57	11	77	0
MC2	458	3	458	3	458	58	458	62	458	3

Key. NS = Not Supported.

The results show that SMC tool capabilities vary depending on the queried properties. For example, PRISM was only able to verify 337 models against Existence, and 435 and 370 models against Until and Universality, respectively. The main reason PRISM failed to verify *all* of the models is that it expects user to increase the depth of the simulation traces, otherwise it cannot verify the unbounded properties with a reliable approximation. In contrast, PLASMA-Lab was able to verify all of the models within 1 hour. Ymer could verify 439 models for those patterns it supports, thus failing to complete 26 models in the time available. MRMC was able to verify relatively few models, because it relied on the PRISM model checker to construct the model and export the associated transition matrices. Especially for relatively large models PRISM crashed while generating these matrices (we believe this is related to its CU Decision Diagram (CUDD) library). MC2 was able to verify 458 models against all of the patterns tested, and only failed for 7 of them.

The second column of the patterns shows the number of models which were verified by the corresponding model checker tools. The distribution of models size across the fastest model checkers for different patterns are shown in the following set of violin plots (Figures 2 – 6). Each of the inner swarm points represents a model. X-axis represents the logarithmic scale of models size. For the models in the white background region, we can uniquely identify the fastest SMC tool for their verification, whereas for the models in grey background region the fastest model checker is not clear.

Ymer was the fastest for most model/pattern pairs (where those patterns were supported). However, it is the fastest tool only for verification of relatively small size models. Ymer was the fastest for verifying 304 models against Existence pattern, the minimum model size verified by Ymer was 2, maximum 2128, mean 256.8 and median 137.5. It was the fastest tool for larger number of models, 324

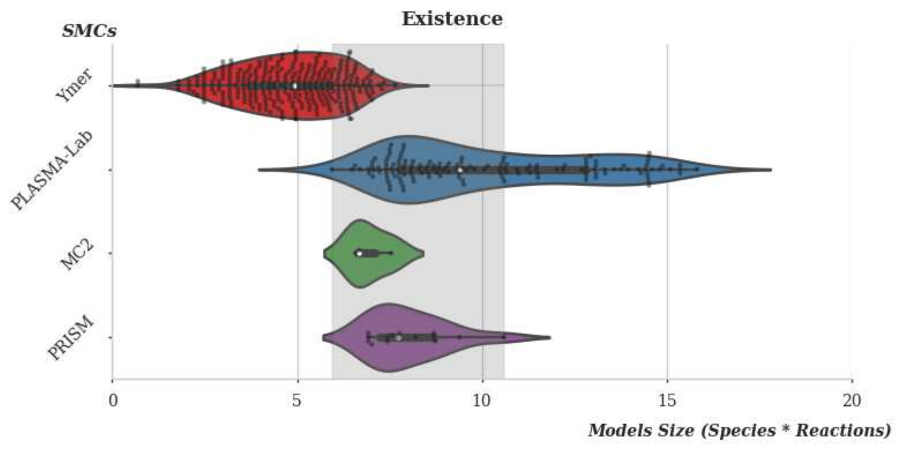

Fig. 2. The distribution of models size across fastest SMC tools for Existence pattern verification.

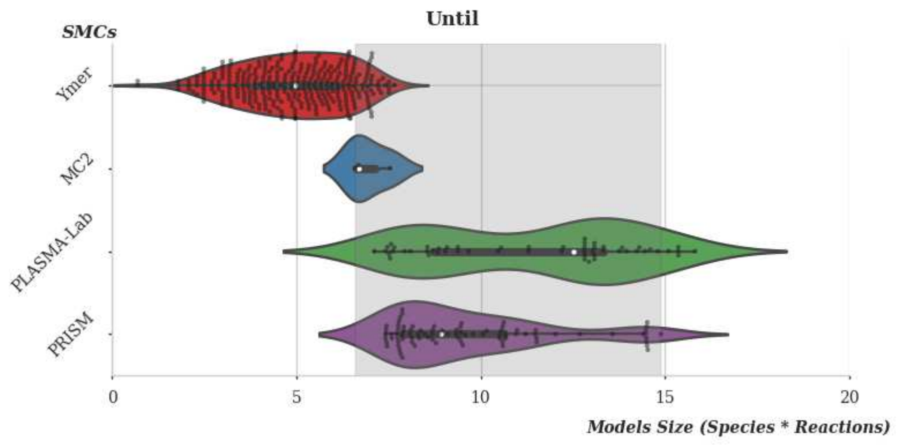

Fig. 3. The distribution of models size across fastest SMC tools for Until pattern verification.

(min = 2, max = 2128, mean = 312.9, median = 144), against Until pattern verification, and 325 models (min = 2, max = 2346, mean = 335, median = 144) against Universality pattern verification. PLASMA-Lab is the fastest tool for relatively large size models. It was the fastest tool for verifying 143 models (min = 380, max = 7429944, mean =464498.9, median = 11875) against Existence pattern, 54 models (min = 1224, max = 7429944, mean = 837193.5, median = 288162) against Until pattern, and 80 models (min = 575, max = 7429944, mean = 773247.5, median = 43143) against Universality pattern verification. It did

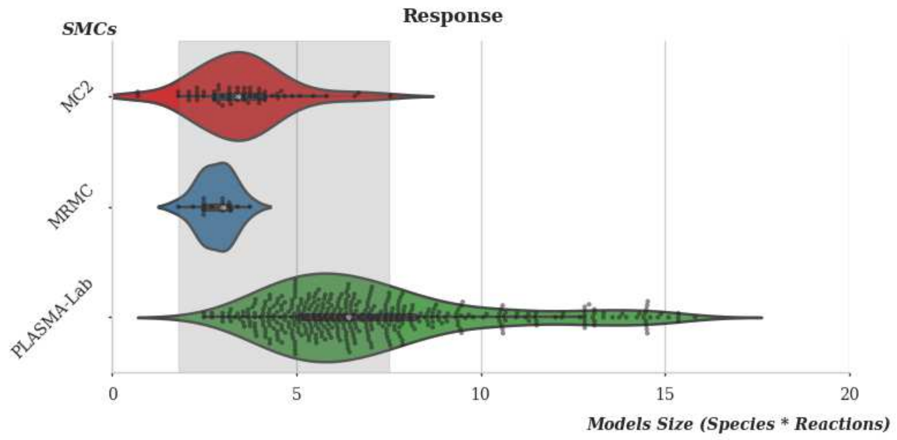

Fig. 4. The distribution of models size across fastest SMC tools for Response pattern verification.

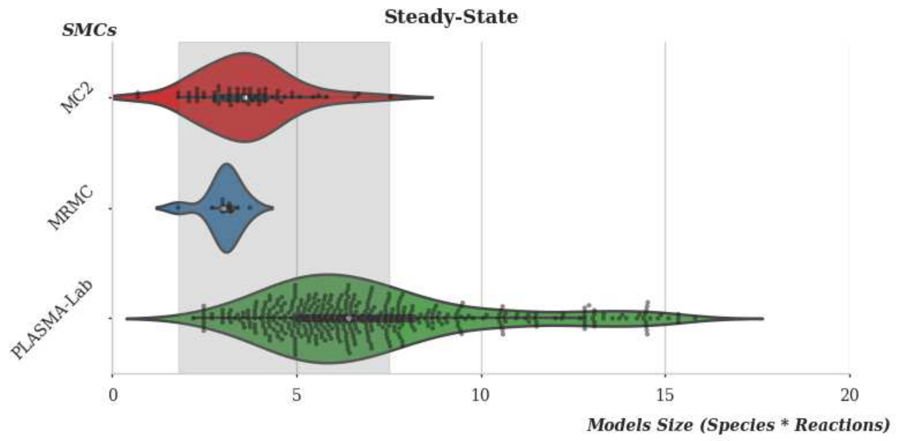

Fig. 5. The distribution of models size across fastest SMC tools for Steady-State pattern verification.

particularly well against Response (390 models (min = 12, max = 7429944, mean = 170734.5, median = 604.5)) and Steady-State patterns (392 models (min = 9, max = 7429944, mean = 169862.1, median = 600), where it was only competing with MRMC and MC2. PRISM is generally the fastest tool for medium to large size models. It was the fastest only for 15 models (min = 1023, max = 39770, mean = 5860.9, median = 2304) against Existence pattern verification, but it was able to verify larger number of models, 84 (min = 1665, max = 2928904, mean = 253327.3, median = 7395), against Until pattern verification and 57 models (min

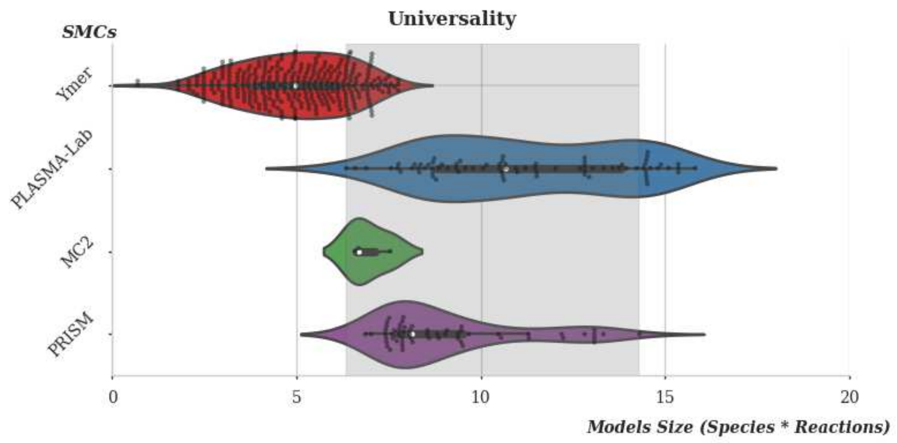

Fig. 6. The distribution of models size across fastest SMC tools for Universality pattern verification.

= 960, max = 1633632, mean = 92998.4, median = 3364) against Universality pattern verification. MC2 (with Gillespie2) is the fastest for relatively small size models. It could verify only 3 models (min = 722, max = 1892, mean = 1138, median = 800) against Existence, Until and Universality patterns, although it did better with 58 models (min = 2, max = 1892, mean = 103.2, median = 30) against Response pattern, and 62 models (min = 2, max = 1892, mean = 105.9, median = 36) against Steady-State patterns. Finally, MRMC (with PRISM dependency) was slower than other tools for Existence, Until and Universality patterns verification, but did better handling Response (fastest for 17 models (min = 6, max = 42, mean = 18.5, median = 20)) and Steady-State (fastest for 11 models (min = 6, max = 42, mean = 22.3, median = 20)).

As we stated previously, the background color of Figures 2 − 6 gives an indication of whether the fastest model checker can be identified for the models within a region of the graph, that is, for models in the white background region, the fastest SMC tool can be identified, but the models in grey background region it is less clear-cut. For verification of Existence pattern, we can uniquely identify the fastest SMC tool for both the 232 smallest models (size ranging from 2 to 380), and the 55 largest models (size = 39984 to 7429944), namely Ymer and PLASMA-Lab respectively, but for remaining 178 medium-sized models (size = 380 to 39770), there is no obvious 'winner'. Similarly, for Until pattern verification, the smallest 283 models (size ranging from 2 to 714), and only for the 5 largest models (size = 3605380 to 7429944) we can identify the fastest SMC tool (Ymer and PLASMA-Lab respectively), but there are more than one candidates for remaining 177 medium-sized models (size = 722 to 2928904). Despite, we have only three SMC tools, namely PLASMA-Lab, MRMC and MC2, which support the verification of Response and Steady-State patterns, their performance on small and medium size models are close to each other, which makes harder to identify the fastest tool. Therefore, only for the smallest 4 models (size = 2 to 6) and for the largest 128 models (size= 1927 to 7429944) the fastest tool (MC2 and PLASMA-Lab respectively) can be identified. Lastly, for Universality pattern verification, the fastest SMC tool for both smallest 262 models (size=2 to 572) and largest 17 models (size =1823582 to 7429944), Ymer and PLASMA-Lab respectively, can be identified, for the remained 186 medium size models we cannot assign a unique model checker tool.

5 Conclusion

The experimental results clearly show that certain SMC tools are best for certain tasks, but there are also situations where the best choice of SMC is far less clear-cut, and it is not surprising that users may struggle to select and use the most suitable SMC tool for their needs. Users need to consider the modelling language of tools and the external tools they may rely on, and need detailed knowledge as to which property specification operators are supported, and how to specify them. Even then, the tool may still fail to complete the verification within a reasonable time, whereas another tool might be able to run it successfully.

These factors make it extremely difficult for users to know which model checker to choose, and point to a clear need for automation of the SMC-selection process. We are currently working to identify novel methods and algorithms to automate the selection of best SMC tool for a given computational model (more specifically for P system models) and property patterns. We aim to enable the integration of our methods within larger software platforms, e.g., IBW and kP-Workbench, and while this is undoubtedly a challenging task, we are encouraged by recent developments in related areas, e.g., the automatic selection of stochastic simulation algorithms [47].

References

1. Alur, R., Henzinger, T.A.: Reactive modules. Form. Methods Syst. Des. 15, 7–48 (Jul 1999), `http://dx.doi.org/10.1023/A:1008739929481`
2. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time Markov chains. ACM Trans. Comput. Logic 1(1), 162–170 (Jul 2000), `http://doi.acm.org/10.1145/343369.343402`
3. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Transactions on Software Engineering 29(6), 524–541 (June 2003)
4. Baier, C., Katoen, J.P.: Principles of model checking. The MIT Press (2008)
5. Bakir, M.E., Ipate, F., Konur, S., Mierla, L., Niculescu, I.: Extended simulation and verification platform for kernel P systems. In: Gheorghe, M., Rozenberg, G., Salomaa, A., Sosík, P., Zandron, C. (eds.) Membrane Computing, pp. 158–178. Lecture Notes in Computer Science, Springer International Publishing (2014), `http://dx.doi.org/10.1007/978-3-319-14370-5\_10`
6. Bakir, M.E., Konur, S., Gheorghe, M., Niculescu, I., Ipate, F.: High performance simulations of kernel P systems. 2014 IEEE 16th International Conference on High Performance Computing and Communications (HPCC) (2014)

7. Bakir, M.E., Stannett, M.: Selection criteria for statistical model checking. In: UCNC'16 Workshop on Membrane Computing (WMC '16) (2016), submitted
8. Bernardini, F., Gheorghe, M., Romero-Campero, F.J., Walkinshaw, N.: A hybrid approach to modeling biological systems. In: Eleftherakis, G., Kefalas, P., Păun, G., Rozenberg, G., Salomaa, A. (eds.) Membrane Computing, Lecture Notes in Computer Science, vol. 4860, pp. 138–159. Springer Berlin Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-77312-2_9
9. Blakes, J., Twycross, J., Romero-Campero, F.J., Krasnogor, N.: The Infobiotics Workbench: An integrated in silico modelling platform for systems and synthetic biology. Bioinformatics 27(23), 3323–3324 (Dec 2011)
10. Blakes, J., Twycross, J., Konur, S., Romero-Campero, F.J., Krasnogor, N., Gheorghe, M.: Infobiotics Workbench: A P systems based tool for systems and synthetic biology. In: Frisco, P., Gheorghe, M., Pérez-Jiménez, M.J. (eds.) Applications of Membrane Computing in Systems and Synthetic Biology, Emergence, Complexity and Computation, vol. 7, pp. 1–41. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-03191-0_1
11. Boyer, B., Corre, K., Legay, A., Sedwards, S.: Plasma-lab: A flexible, distributable statistical model checking library. In: Proceedings of Quantitative Evaluation of Systems - 10th International Conference, QEST 2013, Buenos Aires, Argentina, August 27-30, 2013. pp. 160–164 (2013)
12. Buchholz, P.: A new approach combining simulation and randomization for the analysis of large continuous time Markov chains. ACM Trans. Model. Comput. Simul. 8(2), 194–222 (Apr 1998), http://doi.acm.org/10.1145/280265.280274
13. Carrillo, M., Góngora, P.A., Rosenblueth, D.A.: An overview of existing modeling tools making use of model checking in the analysis of biochemical networks. Frontiers in Plant Science 3(155), 1–13 (2012)
14. Cavaliere, M., Mazza, T., Sedwards, S.: Statistical model checking of membrane systems with peripheral proteins: Quantifying the role of estrogen in cellular mitosis and DNA damage. In: Applications of Membrane Computing in Systems and Synthetic Biology, Emergence, Complexity and Computation, vol. 7, pp. 43–63. Springer International Publishing (2014), http://dx.doi.org/10.1007/978-3-319-03191-0_2
15. Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT Press (1999)
16. Donaldson, R.;Gilbert, D.: A Monte Carlo model checker for Probabilistic LTL with numerical constraints. Tech. rep., University of Glasgow, Department of Computing Science (2008)
17. Dragomir, C., Ipate, F., Konur, S., Lefticaru, R., Mierla, L.: Model checking kernel P systems. In: Membrane Computing, Lecture Notes in Computer Science, vol. 8340, pp. 151–172. Springer Berlin Heidelberg (2014), http://dx.doi.org/10.1007/978-3-642-54239-8_12
18. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 21st International Conference on Software Engineering. pp. 411–420. ICSE '99, ACM, New York, NY, USA (1999), http://doi.acm.org/10.1145/302405.302672
19. The European Bioinformatics Institute. http://www.ebi.ac.uk/, [Online; accessed 08/01/15]
20. Fisher, J., Piterman, N.: Model checking in biology, pp. 255–279. Springer Verlag (2014)
21. Fisher, J., Henzinger, T.A.: Executable cell biology. Nat Biotech 25(11), 1239–1249 (2007)

22. Frisco, P., Gheorghe, M., Pérez-Jiménez, M.J. (eds.): Emergence, Complexity and Computation, vol. 7. Springer International Publishing (2014)
23. Gheorghe, M., Konur, S., Ipate, F., Mierla, L., Bakir, M.E., Stannett, M.: An integrated model checking toolset for kernel P systems. In: Rozenberg, G., Salomaa, A., Sempere, M.J., Zandron, C. (eds.) Membrane Computing: 16th International Conference, CMC 2015, Valencia, Spain, August 17-21, 2015, Revised Selected Papers. pp. 153–170. Springer International Publishing, Cham (2015), `http://dx.doi.org/10.1007/978-3-319-28475-0_11`
24. Grunske, L.: Specification patterns for probabilistic quality properties. In: Proceedings of the 30th International Conference on Software Engineering. pp. 31–40. ICSE '08, ACM, NY, USA (2008), `http://doi.acm.org/10.1145/1368088.1368094`
25. Harel, D.: Statecharts: a visual formalism for complex systems. Science of Computer Programming 8(3), 231 – 274 (1987), `http://www.sciencedirect.com/science/article/pii/0167642387900359`
26. Heiner, M., Gilbert, D., Donaldson, R.: Petri nets for systems and synthetic biology. In: Proceedings of the Formal Methods for the Design of Computer, Communication, and Software Systems 8th International Conference on Formal Methods for Computational Systems Biology. pp. 215–264. SFM'08, Springer-Verlag, Berlin, Heidelberg (2008), `http://dl.acm.org/citation.cfm?id=1786698.1786706`
27. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 441–444. Springer Berlin Heidelberg (2006)
28. Huth, M., Ryan, M.: Logic in computer science: Modelling and reasoning about systems. Cambridge University Press (2004), `http://books.google.co.uk/books?id=eUggAwAAQBAJ`
29. Ibarra, O.H., Păun, G.: Membrane computing: A general view. Ann Eur Acad Sci. EAS Publishing House, Liege pp. 83–101 (2006)
30. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. In: Quantitative Evaluation of Systems (QEST). pp. 167–176. IEEE Computer Society (2009)
31. Kauffman, S.A.: Metabolic stability and epigenesis in randomly constructed genetic nets. Journal of Theoretical Biology 22, 437–467 (1969)
32. Konur, S., Gheorghe, M.: A property-driven methodology for formal analysis of synthetic biology systems. IEEE/ACM Transactions on Computational Biology and Bioinformatics 12(2), 360–371 (March 2015)
33. Konur, S., Gheorghe, M., Dragomir, C., Mierla, L., Ipate, F., Krasnogor, N.: Qualitative and quantitative analysis of systems and synthetic biology constructs using P systems. ACS Synthetic Biology 4(1), 83–92 (2015), `http://dx.doi.org/10.1021/sb500134w`, pMID: 25090609
34. kPWorkbench. `http://kpworkbench.org/`, [Online; accessed 08/01/15]
35. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: Probabilistic symbolic model checker. In: Computer performance evaluation: modelling techniques and tools, pp. 200–204. Springer Berlin Heidelberg (2002)
36. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Proceedings of the 7th International Conference on Formal Methods for Performance Evaluation. pp. 220–270. SFM'07, Springer-Verlag, Berlin, Heidelberg (2007), `http://dl.acm.org/citation.cfm?id=1768017.1768023`
37. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Proceedings of Runtime Verification: First International Conference, RV 2010, St. Julians, Malta, November 1-4. pp. 122–135. Springer, Berlin, Heidelberg (2010), `http://dx.doi.org/10.1007/978-3-642-16612-9_11`

38. Lindenmayer, A., Jürgensen, H.: Grammars of development: Discrete-state models for growth, differentiation, and gene expression in modular organisms. In: Lindenmayer Systems: Impacts on Theoretical Computer Science, Computer Graphics, and Developmental Biology, pp. 3–21. Springer, Berlin, Heidelberg (1992), http://dx.doi.org/10.1007/978-3-642-58117-5_1
39. Milner, R.: Communicating and mobile systems: The Pi-calculus. Cambridge University Press, New York, NY, USA (1999)
40. Monteiro, P.T., Ropers, D., Mateescu, R., Freitas, A.T., de Jong, H.: Temporal logic patterns for querying dynamic models of cellular interaction networks. Bioinformatics 24(16), i227–i233 (Aug 2008), http://dx.doi.org/10.1093/bioinformatics/btn275
41. Markow Reward Model Checker (MRMC). http://www.mrmc-tool.org/, [Online; accessed 18/02/15]
42. Păun, G.: Introduction to membrane computing. In: Ciobanu, G., Păun, G., Pérez-Jiménez, M. (eds.) Applications of Membrane Computing, pp. 1–42. Natural Computing Series, Springer Berlin Heidelberg (2006), http://dx.doi.org/10.1007/3-540-29937-8_1
43. Pérez-Jiménez, M.J., Romero-Campero, F.J.: P systems, a new computational modelling tool for systems biology. In: Priami, C., Plotkin, G. (eds.) Transactions on Computational Systems Biology VI, pp. 176–197. Springer, Berlin, Heidelberg (2006), http://dx.doi.org/10.1007/11880646_8
44. Plasma-Lab. https://project.inria.fr/plasma-lab/, [Online; accessed 18/02/15]
45. Reisig, W.: The basic concepts. In: Understanding Petri Nets, pp. 13–24. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-33278-4_2
46. Probabilistic and Symbolic Model Checker (PRISM). http://www.prismmodelchecker.org/, [Online; accessed 08/01/15]
47. Sanassy, D., Widera, P., Krasnogor, N.: Meta-stochastic simulation of biochemical models for systems and synthetic biology. ACS Synthetic Biology 4(1), 39–47 (2015), http://dx.doi.org/10.1021/sb5001406, pMID: 25152014
48. Ymer website. http://www.tempastic.org/ymer/, [Online; accessed 25/8/15]
49. Younes, H., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. International Journal on Software Tools for Technology Transfer (STTT) 8(3), 216–228 (2006)
50. Younes, H.L.S.: Ymer: A statistical model checker. In: Proceedings of the 17th International Conference on Computer Aided Verification. pp. 429–433. CAV'05, Springer-Verlag, Berlin, Heidelberg (2005), http://dx.doi.org/10.1007/11513988_43
51. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Proceedings of Computer Aided Verification: 14th International Conference, CAV 2002 Copenhagen, Denmark, July 27–31, pp. 223–235. Springer, Berlin, Heidelberg (2002), http://dx.doi.org/10.1007/3-540-45657-0_17
52. Zapreev, I.S., Jansen, C.: Markov reward model checker manual, http://www.mrmc-tool.org/downloads/MRMC/Specs/MRMC_Manual.pdf
53. Zuliani, P.: Statistical model checking for biological applications. International Journal on Software Tools for Technology Transfer 17(4), 527–536 (2014), http://dx.doi.org/10.1007/s10009-014-0343-0