ISCASMC: A Web-Based Probabilistic Model Checker *

Ernst Moritz Hahn[1], Yi Li[2], Sven Schewe[3], Andrea Turrini[1], and Lijun Zhang[1]

[1] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences
[2] LMAM & Department of Information Science, School of Math. Sciences, Peking University
[3] Department of Computer Science, University of Liverpool

Abstract. We introduce the web-based model checker ISCASMC for probabilistic systems (see `http://iscasmc.ios.ac.cn/IscasMC`). This Java application offers an easy-to-use web interface for the evaluation of Markov chains and decision processes against PCTL and PCTL* specifications. Compared to PRISM or MRMC, ISCASMC is particularly efficient in evaluating the probabilities of LTL properties.

1 Introduction

Markov decision processes (MDP) are widely used to model and analyse systems that exhibit both probabilistic and nondeterministic choices. To reason about such systems, one often specifies properties in the popular probabilistic temporal logics PCTL, PLTL, or PCTL* [2]. While PCTL* is more expressive, it suffers from a higher complexity compared to PCTL [4]: model checking MDPs against PCTL specifications is linear, but against PCTL* specifications is 2EXPTIME complete, and the doubly exponential cost is usually incurred through the translation of the LTL fragments to deterministic automata. Several probabilistic model checkers have been developed for verifying Markov chains and MDPs. The state-of-the-art probabilistic model checker PRISM [10] supports both PCTL and PCTL*. Another model checker MRMC [8] is predominantly used for model checking PCTL properties with reward extensions. On the other side, LIQUOR [3] is a probabilistic model checker for PLTL properties.

In this paper, we present a new model checker for probabilistic models, called ISCASMC. ISCASMC supports Markov chains and MDPs, and properties specified in PCTL*. It implements the efficient heuristics in [7] particularly tuned to handle linear time properties. ISCASMC is written in Java, while including a few off-the-shelf components like SPOT [5] on the server side. The web interface on the client side is written in HTML and JavaScript, such that ISCASMC enjoys full portability: it can be run from any machine with internet access and a browser, be it a laptop or a mobile phone.

In the web interface, one can easily import or create examples, analyse it and track the results. The computation is performed on the server, thus making the evaluation of Markov chains and MDPs very easy and readily available. The main features of the tool include modularity, support of linear time properties, and specification of linear time properties using pattern formulas.

* Supported by the National Natural Science Foundation of China (NSFC) under grant No. 61361136002, 61350110518, 61202069, the Chinese Academy of Sciences Fellowship for International Young Scientists (Grant No. 2013Y1GB0006), Research Fund for the Doctoral Program of Higher Education of China (Grant No. 20120001120103), and Engineering and Physical Science Research Council (EPSRC) through the grant EP/H046623/1.

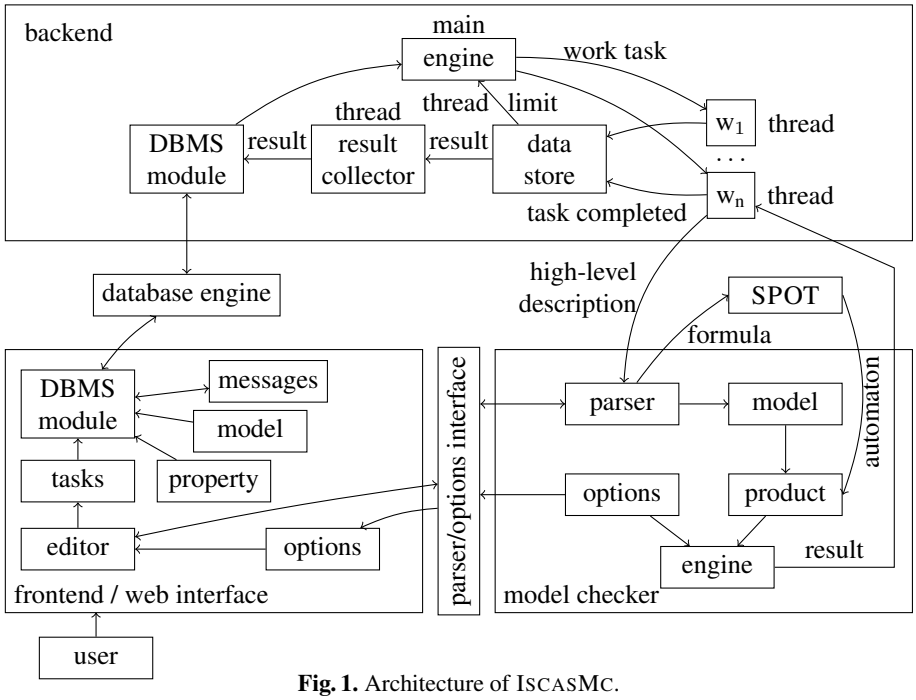

Fig. 1. Architecture of ISCASMC.

Outline. We describe the architecture and usage of the tool in Section 2. The main features of ISCASMC are given in Section 3, and Section 4 concludes the paper.

2 Architecture and Usage

The architecture and components of ISCASMC are depicted in Figure 1. It has three main components: the frontend web interface, the backend for handling requests from the frontend, and the model checker engine. A database engine is used to store information. We describe these components in detail.

2.1 The Frontend Web Interface

The frontend allows for logging into the system, either as a guest or as a registered user. Guest users can experiment with most of the features of the tool, but they have limited resources, for instance small timeout values.

After logging into the system, it offers several views, including:

- *Message Centre*. The message centre provides the user recent news. Particularly, one can post messages, send models to other users as well as receive models shared by other users.
- *Model Centre*. The model management centre lists available models, their type (currently only PRISM models are supported), comments, options, last updated snapshot and all available operations for the model. From the menu above one can also upload or create new models. The properties are associated to models. For each model, one can create and analyse these properties. Currently, ISCASMC supports Markov chains and MDPs and properties in PCTL*. Once one clicks on one of the models in the list, one enters the editing page.

In the editing page, models can be edited, and properties can be added, modified, or removed. A model may have more than one associated properties.

– *Task Centre*. In the model centre, the user can choose to check selected properties or all properties. This is referred to as a *task*. Note that a task is created as a snapshot of the current model, (selected) properties and options. This allows the user to modify the model/properties and submit several tasks without having to wait for the termination of the previous submitted tasks. The task, together with the corresponding options, will be sent to the server side to be handled. In the task centre, one can find a list of all submitted tasks from the user. For each task, one can track the current status, find the final results once available, see the complete log generated by the model checker, or remove the task.
– *Option Centre*. From the option centre the user can set the user level options. Moreover, for each model to be analysed, one may modify certain options and get model level options. The model level options have higher priority and will overwrite user level options.
– *Example Centre*. From the example centre, the user can directly import several examples together with associated properties into her own account.

The Interface. While the frontend does not play a role in the evaluation of the model, it includes a fast syntax check that allows for checking the syntactical correctness of the model while interacting with the editor. As shown in Figure 1, the parser and the options interface are shared between the model checker and the frontend.

The part available to the frontend is a stand-alone program (on the server side) that makes use of only a small part of the classes in the model checker engine. This lightweight version is only used for checking the syntactical correctness of models from the client side, while the full version on the model checker site also constructs the respective models and automata. These syntactical correctness checks are simple and can thus be done efficiently on-demand, bypassing the scheduling queue.

2.2 The Database

The database, powered by MySQL™, contains all information needed to elaborate the models: besides the `user` details, it stores the `models` and the relative `properties` defined by the user, as well as the `tasks` the user creates by requiring an operation on the model. Each task is created by the frontend DBMS module as snapshot of the model and the corresponding properties and options, such that it is not affected by subsequent changes. Once a task is completed, it is updated by the backend DBMS module with the evaluation of the properties (or with an error message), according to the model checker outcome. The tasks are kept until the user explicitly removes them via the task centre.

2.3 The Backend

The main job of the backend is to poll tasks from the database and evaluate them. It currently adopts a FIFO approach to retrieve the tasks from the DBMS module. These tasks are then sent to several instances of ISCASMC that run in parallel in independent worker threads $w_1, \ldots, w_n$. Once a worker w_i completes her task, she sends the outcome to the data store, whose main jobs are to keep track of busy workers and to collect the results. Since the evaluation of a task may take some time, the worker periodically sends status updates to the data store. The result collector retrieves the available results from the data store and forwards them to the database via the backend DBMS module.

2.4 The Model Checker Engine

The model checker is the working horse of the system. Each work thread will parse and translate the model and the specification it is going to be checked against. For complex LTL subformulas (that is, for each linear part φ of PCTL* outside of the simple PCTL operators), we first use SPOT [5] to generate the generalised Büchi automaton. Unless this is already deterministic, we then use the layered approach from [7], which uses first subset constructions (with an over- and underapproximation of the acceptance condition), and subsequently refines them (where necessary) first to a breakpoint construction (again with an over- and underapproximation of the acceptance condition) and then to the deterministic Rabin automata [11]. The product of the automaton and model is an MDP equipped with accepting conditions. These accepting conditions are used to identify accepting states in the product, after which the problem reduces to a *probabilistic reachability* problem for MDPs. The reachability problem is a central problem in probabilistic model checking, and is well studied, see [1] for a survey. One can apply value iteration or policy iteration to solve it. Currently, ISCASMC uses a value iteration approach based on Gauss-Seidel or Jacobi method. After the evaluation, it returns the results to the backend.

3 Main Features

ISCASMC supports models and properties described in the PRISM input language. A nice feature of ISCASMC is that it provides a plain web interface that allows users to easily try the probabilistic model checker. Since the computation is performed on the server side, the user can conduct her experiments using computers, but also smart phones and any device with a browser and internet access. Besides its performance, the main features of ISCASMC are *modularity* and the handling of *linear time properties*.

- *Modularity.* The three main components of ISCASMC are essentially independent of each other. This allows for distributing the computation, i.e., it allows for centring the power-hungry evaluation on powerful servers, while using simple machines like smartphones for the initiation and control of the experiments.
- *Pattern formulas.* ISCASMC provides pattern formula specifications. This allows one to easily add and check PLTL properties based on the absence and response LTL patterns proposed in [6] by simply choosing basic events among existing labels in the model or by writing her own events. All occurrences of the same event identifier are automatically replaced by actual content.
- *Error tracking.* ISCASMC being a web-based tool, we are able to keep track of all internal assertion failures and runtime errors which occur during any model checking run. This way, we are able to reproduce the according bug and thus can quickly fix the problem.
- *Comparison Platform.* ISCASMC can be easily extended for providing a comparing platform for off-the-shelf probabilistic model checkers. We shall leave it as our future work.
- *Linear time properties.* Comparing to existing model checkers such as PRISM, MRMC, and LIQUOR, ISCASMC builds on efficient heuristics in [7] tailored to linear time properties.

Example 1. In Table 1, we consider a variant of the quasi birth-death process described in [9] together with some example properties. For each of the properties, we compare

	ISCASMC				PRISM		
property	time (s)	prod. states	autom. states	type	time (s)	prod. states	autom. states
$prop\mathbf{U}$	1	805	2	subset	24	808	12
$prop\mathbf{GF}\wedge$	1	825	6	breakp.	365	825	77775
$prop\mathbf{GF}\vee$	1	1634	8	rabin	34	823	4110

Table 1. Quasi birth-death process. The example and properties can be loaded on the webpage.

performance results of ISCASMC and PRISM. For both tools we provide the total runtime in seconds, the number of states of the property automaton constructed for the analysis, and the size of the product of this automaton with the model. For ISCASMC, we also state with which method from [7] (subset, breakpoint, or Rabin construction) the property can be decided.

For this set of properties, ISCASMC terminated considerably faster. The reason is that the time required to construct the complete Rabin determinisation is very high. The approach in [7] completely avoids this construction for the first two properties. In the third one, it employs an on-the-fly implementation of the state-of-the-art Rabin determinisation algorithm [11]. We can therefore restrict the use of the full Rabin construction to refining those parts of the product, where this is required for deciding the respective property.

4 Future Work

We plan to extend ISCASMC to support more model types such as continuous-time Markov decision processes and Markov games, recursive Markov chains and quantum Markov chains. On the property side, we plan to incorporate more general properties such as reward properties and ω-regular languages. To allow handling larger models, we plan to extend our implementation to use symbolic rather than explicit-state methods.

References

1. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
2. Bianco, A., de Alfaro, L.: Model checking of probabalistic and nondeterministic systems. In: FSTTCS. LNCS, vol. 1026, pp. 499–513. Springer (1995)
3. Ciesinski, F., Baier, C.: LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems. In: QEST. pp. 131–132 (2006)
4. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. JACM 42(4), 857–907 (1995)
5. Duret-Lutz, A.: LTL translation improvements in SPOT. In: VECoS. pp. 72–83. BCS (2011)
6. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: FMSP. pp. 7–15 (1998)
7. Hahn, E.M., Li, G., Schewe, S., Zhang, L.: Lazy determinisation for quantitative model checking (2013), arXiv:1311.2928
8. Katoen, J.P., Khattri, M., Zapreev, I.S.: A Markov reward model checker. In: QEST. pp. 243–244 (2005)
9. Katoen, J.P., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for continuous-time Markov chains. In: CAV. pp. 311–324 (2007)
10. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV. LNCS, vol. 6806, pp. 585–591 (2011)
11. Schewe, S., Varghese, T.: Tight bounds for the determinisation and complementation of generalised Büchi automata. In: ATVA. LNCS, vol. 7561, pp. 42–56. Springer (2012)