Statistical Model Checking of Simulink Models with Plasma Lab

Axel Legay, Louis-Marie Traonouez

Statistical Model Checking of Simulink Models with Plasma Lab

Axel Legay and Louis-Marie Traonouez

Inria Rennes – Bretagne Atlantique

Abstract. We present an extension of the statistical model-checker Plasma Lab capable of analyzing Simulink models.

1 Introduction

Formal methods comprise a wide range of techniques capable of proving or evaluating the safety of a system. Model based techniques, like model-checking, rely on a formal model of the system in order to perform an exhaustive exploration of its state-space. The technique reaches its limit when the state-space of the model is too large to be explored entirely, or when the model mixes heterogeneous data like time, quantities and probabilities. Statistical Model Checking (SMC) is an alternative technique that combines formal analysis with statistical methods. It relies on a finite number of simulations of a formal model in order to compute an evaluation of the system's safety as a probability measure. This lightweight approach can be applied on complex systems, even with infinite state-space.

SMC can be implemented easily for a wide range of formal models or even directly applied to a system simulator. It only depends on three basic components: 1. a simulator of the model or the system, capable of generating random traces, specified as a finite sequence of states; 2. a monitor, that determines if a trace satisfies a property expressed in a formal logic like the Bounded Linear Temporal Logic; 3. an SMC algorithm from the statistic area that evaluates the probability to satisfy the formal property. For instance, the Monte Carlo algorithm computes N executions ρ and it estimates the probability γ that the system satisfies a logical formula φ using the following equation:

$$\tilde{\gamma} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}(\rho \models \varphi)$$

where $\mathbf{1}$ is an indicator function that returns 1 if φ is satisfied and 0 otherwise. It guarantees that the estimate $\tilde{\gamma}$ is close enough to the true probability γ with a probability of error that is controlled by the number N of simulations.

Several model-checking tools have added SMC as a complement to exhaustive model-checking. This includes the model-checker UPPAAL [5] for timed automata, the probabilistic model-checker PRISM [7], and the model-checker Ymer [9] for continuous time Markov chains. Plasma Lab [3] is the first platform entirely dedicated to SMC. Contrary to other tools, that target a specific domain and offer several analysis techniques, including basic SMC algorithms, Plasma Lab is designed as a generic platform that offers several advanced SMC algorithms that can be applied to various models. Indeed to apply Plasma Lab algorithms to a new model or system it is only required to implement a simulator that extends public interfaces from Plasma Lab API. Currently, Plasma Lab can already be used with the PRISM language, biological models, the SystemC language, and Simulink models, the extension presented in this paper.

Simulink is a graphical programming language for multidomain dynamic systems. It is part of the MATLAB environment, a widely used tool in the industry. Simulink models can be formally translated to hybrid automata [1], that interleave discrete state automata with complex dynamic behaviors described by differential equations. Model-checking of these models is however undecidable. It is therefore interesting to use SMC to provide a formal analysis technique for these models. Rather than translating Simulink models to a specific formal language, we have been able to directly interface Plasma Lab and Simulink, and we apply SMC algorithms by using the simulation engine provided by Simulink. This approach facilitates the adoption of formal methods by non experts, who can launch SMC analyses directly from a small MATLAB App.

2 Plasma Lab Architecture

Plasma Lab is a compact, efficient and flexible platform for SMC. The tool offers a series of SMC algorithms, included advanced techniques for rare events simulation, distributed SMC, non-determinism, and optimization. The main difference between Plasma Lab and other SMC tools is that Plasma Lab proposes an API abstraction of the concepts of stochastic model simulator, property checker (monitoring) and SMC algorithm. In other words, the tool has been designed to be capable of using external simulators, input languages, or SMC algorithms. This not only reduces the effort of integrating new algorithms, but also allows us to create direct plug-in interfaces with industry used specification tools. The latter being done without using extra compilers.

Fig. 1 presents Plasma Lab architecture. More specifically, the relations between model simulators, property checkers, and SMC algorithms components. The simulators features include starting a new trace and simulating a model step by step. The checkers decide a property on a trace by accessing to state values. They also control the simulations, with a *state on demand* approach that generates new states only if more states are needed to decide the property. A SMC algorithm component is a runnable object. It collect samples obtained from a checker component. Depending on the property language, their checker either returns Boolean or numerical values. The algorithm then notifies progress and sends its results through the Controller API.

In coordination with this architecture, we use a plugin system to load models and properties components. It is then possible to support new model or property languages. Adding a simulator or a checker is pretty straightforward as they share a similar plugin architecture. Thus, it requires only a few classes and methods

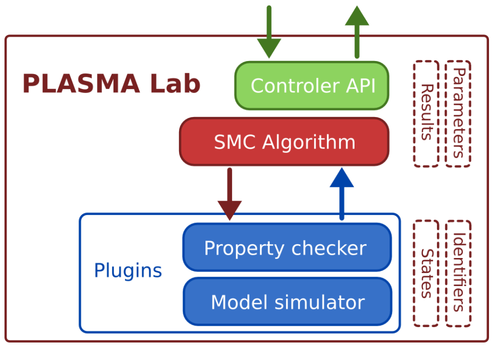

Fig. 1: Plasma Lab architecture

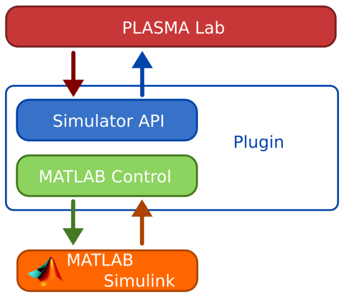

Fig. 2: Interface between Plasma Lab and Simulink

to get a new component running. Each plugin contains a factory class used by Plasma Lab to instantiate component objects. These components implement the corresponding interface defining their behavior. Some companion objects are also required (results, states, identifiers) to allow communication between components and the Controller API.

One of the goal of Plasma Lab is also to benefit from a massive distribution of the simulations, which is one of the advantage of the SMC approach. Therefore Plasma Lab API provides generic methods to define distributed algorithms.

3 Plasma Lab and Simulink Integration

We now show how we have integrated Plasma Lab within Simulink, hence lifting the power of our simulation approaches directly within the tool.

In order to obtain significant results with SMC the Simulink models should include randomly generated events. By default the Simulink library provides some random generators, but these are not compatible with SMC: they always generate the same random sequence of values at each execution. To overcome this limitation we use some custom C-code blocks that generate independent sequences of random draws.

Our objective was to reuse the simulation engine provided with Simulink and to integrate it in Plasma Lab. To do so, we developed a simulator plugin whose architecture is showed in Fig. 2. One of the key points of our integration has been to exploit MATLAB Control[1], a library that allows to interact with MATLAB from Java. This library uses a proxy object connected to a MATLAB session. Function calls and variables access are transmitted and executed on the MATLAB session through the proxy. This allowed us to implement a MATLAB program that controls a Simulink simulation. Calls to this implementation are then done in Java from the Plasma Lab plugin.

Regarding the monitoring of properties, we exploit the simulation output of Simulink. More precisely, BLTL properties are checked over sequences of states

[1] `https://code.google.com/p/matlabcontrol/`

and time stamps, based on a set of state variables defined by declaring some Simulink signals as log output. During the simulation these signals are logged in a data structure containing time stamps and are then retrieved as states in Plasma Lab. One important point is that Simulink discretizes the signals trace, its sample frequency being parameterized by each block. In terms of monitoring this means that the sample frequency must be configured to observe any relevant change in the model. In practice, the frequency can be set as a constant value, or, if the model mixes both continuous data flow and state flow, the frequency can be aligned on the transitions, *i.e.*, when a state is newly visited.

Usage We provide a Simulink plugin for the main interface of Plasma Lab. Simulink models can be loaded in the interface and a MATLAB instance is started to simulate the models. Alternatively we provide PLASMA2Simulink, a MATLAB App that can be installed in MATLAB. It contains all the necessary components to verify Simulink models: the simulator plugin, a BLTL monitor and SMC algorithms. Then, SMC experiments can be directly started in MATLAB from this App: it allows to select a model, a property and an algorithm, to specify the parameters of the experiment and it displays the results. Both Plasma Lab and PLASMA2Simulink can be downloaded from our website [2].

Applications We also describe in this webpage[3] two case-studies developed with Simulink and verified with Plasma Lab. The first is a fuel control system provided by MathWorks. The second described the temperature controller of a pig shed.

In the first one, we replace manual switches, used in the standard model to introduce failures in the system sensors, by random generators that implement a Poisson probability distribution using C-code blocks. We then analyze the probability of a long engine shutdown and compare our results obtained with Plasma Lab with the results from [10].

4 Related Works

A first experiment with SMC and Simulink was presented in [10]. Their approach consists in programming one SMC algorithm within the Simulink toolbox. On the contrary, the flexibility of our tool will allow us to incrementally add new algorithms to the toolbox without new programming efforts.

A few other works consider formal verification of Simulink models via model-checking. None consider adding stochastic behaviors to Simulink, but consider the hybrid automata semantics of these models. However, model-checking hybrid automata is undecidable, and therefore, the existing approaches restrict the type of blocks that can be used in Simulink models: in general by removing continuous behaviors in order to obtain a finite state machine. For instance Honeywell

[2] `https://project.inria.fr/plasma-lab/download/`

[3] `https://project.inria.fr/plasma-lab/examples/`

presents in [8] a tool that translates certain Simulink models to the input language of the model-checker NuSMV. [2] also presents a tool chain that translates Simulink models to the input language of the LTL model-checker DiViNE. This tool chain uses the tool HiLiTe [6], also developed by Honeywell, that can perform semantic analyses of Simulink models. Contrary to these model-checking approaches, SMC techniques are not restricted by the model, and our Simulink plugin for Plasma Lab is able to handle any type of Simulink and Stateflow diagrams, with both continuous and discrete behaviors.

Finally, our approach is also different from the one in [4] that consists in translating parts of Simulink models into the UPPAAL language. This makes it difficult to analyze counter examples as it implies remapping traces from UPPAAL to the Simulink model. Therefore Plasma Lab offers the first integrated verification tool for Simulink models with stochastic information.

Acknowledgement This work was supported by the European Union Seventh Framework Programme under grant agreement number 318490 (SENSATION).

References

1. Agrawal, A., Simon, G., Karsai, G.: Semantic Translation of Simulink/Stateflow Models to Hybrid Automata Using Graph Transformations. Electron. Notes Theor. Comput. Sci. 109, 43–56 (Dec 2004)
2. Barnat, J., Beran, J., Brim, L., Kratochvíla, T., Ročkai, P.: Tool Chain to Support Automated Formal Verification of Avionics Simulink Designs. In: Formal Methods for Industrial Critical Systems, LNCS, vol. 7437, pp. 78–92. Springer (2012)
3. Boyer, B., Corre, K., Legay, A., Sedwards, S.: PLASMA-lab: A Flexible, Distributable Statistical Model Checking Library. In: Proceedings of QEST. LNCS, vol. 8054, pp. 160–164. Springer (2013)
4. David, A., Du, D., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Statistical Model Checking for Stochastic Hybrid Systems. In: Proceedings of HSB. EPTCS, vol. 92, pp. 122–136 (2012)
5. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for Statistical Model Checking of Real-Time Systems. In: Proceedings of CAV. vol. 6806, pp. 349–355. Springer (2011)
6. Devesh Bhatt, Gabor Madl, David Oglesby, Kirk Schloegel: Towards Scalable Verification of Commercial Avionics Software. In: AIAA Infotech@Aerospace (2010)
7. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: Verification of Probabilistic Real-Time Systems. In: Proceedings of CAV. LNCS, vol. 6806, pp. 585–591. Springer (2011)
8. Meenakshi, B., Bhatnagar, A., Roy, S.: Tool for Translating Simulink Models into Input Language of a Model Checker. In: Formal Methods and Software Engineering, LNCS, vol. 4260, pp. 606–620. Springer (2006)
9. Younes, H.L.S.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis, Carnegie Mellon (2005)
10. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to Stateflow/Simulink verification. Formal Methods in System Design 43(2), 338–367 (2013)