Statistical Model Checking with Changes and Simulink

Benoît Boyer, Kevin Corre, Axel Legay, Louis-Marie Traonouez

Statistical Model Checking with Changes and Simulink

Benoît Boyer, Kevin Corre, Axel Legay, Louis-Marie Traonouez

Inria Rennes – Bretagne Atlantique

Abstract. Statistical Model Checking (SMC) is a powerful and widely used approach that consists in extracting global information on the system by monitoring some of its executions. In this paper, we add two new stones to the cathedral of results on SMC, that are 1. a new algorithm to detect emergent behaviors at runtime, and 2. an integration of Plasma Lab, a powerful SMC checker, as a library of Simulink. Our results are illustrated on a realistic case study.

1 Introduction, Motivations

Complex systems such as cyber-physical systems are large-scale distributed systems, often viewed as networked embedded systems, where a large number of computational components are deployed in a physical environment. Each component collects information and offers services to its environment (e.g., environmental monitoring and control, health-care monitoring and traffic control). This information is processed either at the component, in the network or at a remote location (e.g., the base station), or in any combination of these.

Characteristic for nowadays complex systems is that they have to meet a multitude of quantitative constraints, e.g., timing constraints, power consumption, memory usage, communication bandwidth, QoS, and often under uncertainty of the behavior of the environment. There is thus the need for new mathematical foundation and supporting tools allowing to handle the combination of quantitative aspects concerning, for example, time, stochastic behavior, hybrid behavior including energy consumption. The main difficulties being that the state space of nowadays systems is too complex to be analyzed with classical validation technique. Another problem being that the interleaving of information eventually leads to undecidability.

In a series of recent works, the formal methods community has studied *Statistical Model Checking techniques* (SMC) [11,17,20,15] as a way to reason on quantitative (potentially undecidable) complex problems. SMC can be seen as a trade-off between testing and formal verification. The core idea of the approach is to conduct some simulations of the system and then use results from the statistic area in order to estimate the probability to satisfy a given property. Of course, in contrast with an exhaustive approach, a simulation-based solution does not guarantee a correct result. However, it is possible to bound the probability of making an error. Simulation-based methods are known to be far less memory and time intensive than exhaustive ones, and are sometimes the only option. SMC

gets widely accepted in various research areas such as systems biology [5,14] or software engineering, in particular for industrial applications. There are several reasons for this success. First, it is very simple to implement, understand and use. Second, it does not require extra modeling or specification effort, but simply an operational model of the system, that can be simulated and checked against state-based properties. Third, it allows to verify properties [2] that cannot be expressed in classical temporal logics. SMC algorithms have been implemented in a series of tools such as Ymer [20], Prism [16], UPPAAL [8], Plasma Lab [4].

Albeit SMC has already been widely adopted by the research community and by the industry. In this paper we propose two new contributions that have the potential to leverage the applicability of SMC to a broader class of systems.

We first adapt CUSUM [18], an algorithm that can be used to detect changes in signal monitoring. We will show that CUSUM can be used to detect when the probability to satisfy a given property drops below some value. This algorithm offers new possibilities to detect, e.g., emergent behaviors in dynamic systems. Our main contributions will be to extend temporal logic with a change-based operator, discuss the calibration of the algorithm and its integration in a global monitoring procedure, and illustrate the applicability on a concrete case study.

While simulation-based approaches have the potential to be directly integrated in design tools used by industry, most of them still rely on their own input languages. Such an approach is often perceived negatively by the engineers. Indeed, formal verification should behave as a helper and not as yet another language they need to learn. We recently developed Plasma Lab an API-based SMC checker. In this paper, we show that by using the API, Plasma Lab can directly be integrated as a Simulink library, hence offering the first in house tool for the verification of stochastic Simulink models – this tool completes the panoply of validation toolsets already distributed with Simulink (see `http://www.mathworks.nl/verification-validation/`). Another advantage of our approach is that any advance on SMC that we will implement within Plasma Lab in the future will directly be available to the Simulink users.

Finally, our third contribution is to show the potential of our approach on a realistic case study.

2 Systems and Problems

Consider a set of states S and a set of states variables SV. Assume that each state variable $x \in SV$ is assigned to domain $\mathbb{D}_x$, and define the valuation function V, such that $V(s, x) \in \mathbb{D}_x$ is the value of x in state s. Consider also a time domain $T \subseteq \mathbb{R}$. We propose the following definition to capture the behavior of a large class of stochastic systems.

Definition 1 (Stochastic Process). *A* stochastic process *is a family of random variables $\mathcal{X} = \{X_t \mid t \in T\}$, each random variable X_t having range S.*

An *execution* for a stochastic process $\mathcal{X}$ is any sequence of observations $\{x_t \in S \mid t \in T\}$ of the random variables $X_t \in \mathcal{X}$. It can be represented as a sequence $\pi = (s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n)$, such that $s_i \in S$ and $t_i \in T$, with

time stamps monotonically increasing, e.g. $t_i < t_{i+1}$. Let $0 \leq i \leq n$, we denote $\pi^i = (s_i, t_i), \ldots, (s_n, t_n)$ the suffix of π starting at position i. Let $\overline{s} \in S$, we denote $Path(\overline{s})$ to be the set of executions of $\mathcal{X}$ that starts in state $(\overline{s}, 0)$ (also called initial state) and $Path^n(\overline{s})$ the set of executions of length n.

In [20], Youness showed that the executions set of a stochastic process is a measurable space, which defines a probability measure μ over $Path(\overline{s})$. The precise definition of μ depends on the specific probability structure of the stochastic process being studied. We now define the general structure for *stochastic discrete event systems*.

Definition 2 (Stochastic Discrete Event System (SDES)). *A* stochastic discrete event system *is a tuple $\mathcal{S}ys = \langle S, I, T, \mu, SV, V \rangle$, where S is a set of states, $I \subseteq S$ is the set of initial states, T is a set of time stamps, μ a probability measure over the executions of Sys, SV is a set of state variables and V is the valuation function.*

Properties over traces of $\mathcal{S}ys$ are defined via the so-called Bounded Linear Temporal Logic (BLTL) [3]. BLTL restricts Linear Temporal Logic by bounding the scope of the temporal operators. Syntactically, we have

$$\begin{aligned} \varphi, \varphi' := \ & \mathsf{true} \mid \mathsf{false} \mid x \sim v \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi' \mid \varphi \Rightarrow \varphi' \mid \neg\varphi \mid (\varphi) \\ & \mid X_{\leq t}\, \varphi \mid F_{\leq t}\, \varphi \mid G_{\leq t}\, \varphi \mid \varphi\, U_{\leq t}\, \varphi' \end{aligned}$$

where φ, φ' are BLTL formulas, $x \in SV$, $v \in D_x$ and $t \in \mathbb{Q}^+$ and $\sim \in \{<, \leq, =, \geq, >\}$. As usual, we define $F_{\leq t}\varphi \equiv \mathsf{true}\ U_{\leq t}\varphi$ and $G_{\leq t}\varphi \equiv \neg F_{\leq t}\neg\varphi$. The semantics of a BLTL is defined with respect to executions π of a SDES using the following rules:

- $\pi \models X_{\leq t}\, \varphi \equiv \exists i,\ i = max\{j \mid t_0 \leq t_j \leq t_0 + t\}$ and $\pi^i \models \varphi$
- $\pi \models \varphi_1\, U_{\leq t}\, \varphi_2 \equiv \exists i,\ t_0 \leq t_i \leq t_0 + t$ and $\pi^i \models \varphi_2$ and $\forall j,\ 0 \leq j < i,\ \pi^j \models \varphi_1$
- $\pi \models \varphi_1 \wedge \varphi_2 \equiv \pi \models \varphi_1$ and $\pi \models \varphi_2$
- $\pi \models \neg\varphi \equiv \pi \not\models \varphi$ and $\pi \models x \sim v \equiv V(s_0, x) \sim v$
- $\pi \models \mathsf{true}$ and $\pi \not\models \mathsf{false}$

In the rest of the paper, we consider three problems that are 1. the quantitative problem for BLTL, 2. the optimization problem for parametric values and 3. the detection of changes.

2.1 Quantitative and Optimization Problems

Given a SDES $\mathcal{S}ys$ and a BLTL property φ, we use $Pr[\mathcal{S}ys \models \varphi]$ to denote the probability for an execution of $\mathcal{S}ys$ to satisfy φ. The *quantitative problem* consists in computing this probability.

We will also study the *optimization problem*, that is the one of finding an initial state that maximizes/minimizes the value of a given observation. Consider a set $\mathcal{O}$ of observations over $\mathcal{S}ys$. Each observation $o \in \mathcal{O}$ is a function o :

$Path^n(\overline{s}) \rightarrow \mathbb{D}_o$ that associates to each run of length n and starting at $\overline{s}$ a value in a domain $\mathbb{D}_o$. We denote $(\tilde{o})_n$ the average value of $o(\pi)$ over all the executions $\pi \in Path^n(\overline{s})$. The *optimization problem* for $\mathcal{S}ys$ is to determine an initial state $\overline{s} \in I$ that minimizes or maximizes the value $(\tilde{o})_n$, for all $o \in \mathcal{O}$.

As an example, an observation can simply be the maximal value of a given parameter along an execution. The average observation then becomes the sum of those observations divided by the number of runs. In this context, the optimization could be to find the initial state that minimizes the value of the parameters.

2.2 Change Detection Problem

In this section, we consider the problem of detecting whether the probability to satisfy a given BLTL property φ changes at execution time. More precisely, given an execution $\pi = (s_0, t_0), (s_1, t_1), \ldots, (s_n, t_n)$ that satisfies φ with a probability $p_0 < k$ (resp. $p_0 > k$) at (s_0, t_0), we are looking for a suffix π^i that satisfies φ with a probability $p \geq k$ (resp. $p \leq k$), with $k \in]0, 1[$. As an example, assume that under some traffic conditions, the firemen can extinguish a fire before three hours with a probability smaller than 0.7, it is expected that this probability decreases when the traffic increases. The challenge is to detect the time t when this change happens.

Formally, we consider a sequence of Bernoulli variables X_i such that $X_i = 1$ iff $\pi^i \models \varphi$. An execution π satisfies a change $p \geq k$ where $p = Pr[\varphi]$ at time t, iff $Pr[X_i = 1] < k$ for $t_i < t$ and $Pr[X_i = 1] \geq k$ for $t_i \geq t$. We note $\tau!$ the fact that an execution is subjected to a change at time t. Using those notations, one can define Boolean propositions over changes and their respective time. One can also combine changes propositions with BLTL formulas, providing that those propositions are not in the scope of temporal operators. We now introduce extended BLTL change-based relations, an extension of BLTL.

Definition 3. *An extended BLTL change relation is defined as:*

$$
\begin{aligned}
\mathit{change} &:= p \star k \ \texttt{where}\ p = Pr[\varphi] \\
\mathit{prop} &:= \texttt{let}\ \tau = \mathit{change}\ \texttt{and}\ \tau' = \mathit{change}\ \texttt{and}\ \ldots\ \texttt{in}\ \delta \\
\delta, \delta' &:= \quad \tau \diamond \tau' + t \mid \tau \diamond t \mid \tau! \mid \varphi \in \mathit{BLTL} \\
&\qquad \mid \delta \vee \delta' \mid \delta \wedge \delta' \mid \neg\delta \mid \delta \Rightarrow \delta' \mid (\delta)
\end{aligned}
$$

with p is a probability identifier, $k \in]0, 1[$, $t \in \mathbb{Q}^+$, $\diamond \in \{<, \leq, \geq, >\}$, $\star \in \{\leq, \geq\}$ and φ is a BLTL formula.

This extension allows us, e.g., to express conditions such as "if a change occurs at time t, then the system shall reach a state x in less than 10 units of time".

3 The Algorithms

In this section, we detail our algorithmic solutions to the three problems described in Section 2. Our solutions rely on a simulation-based approach that falls into the family of the so-called *Statistical Model Checking* (SMC) approach.

3.1 Quantitative Verification and Optimization

We first focus on the problem of computing the probability for a SDES $\mathcal{S}ys$ to satisfy a given BLTL property. For doing so, we rely on Statistical Model Checking (SMC).

SMC estimates the probability that a system satisfies a property using a number of statistically independent simulation traces of an executable model. The idea being to monitor the property on each simulation, and then use an algorithm from the statistic area to compute an estimate of the probability. Those algorithms include Monte Carlo, or importance sampling/splitting [12]. Monitoring algorithms for BLTL can be found in [10]. In our work, the quantitative problem will mainly be solved in the context of calibrating a change algorithm as well as to validate quantitative properties of the model.

Optimization We now show that a simulation approach can also be used to perform an optimization of the model by varying the model parameters and evaluating the observable quantities to optimize. We consider a stochastic state transition system $\mathcal{S}ys$, with a set of initial states I, and a set of observations $\mathcal{O}$.

For each initial state $\overline{s} \in I$ we perform N random simulations of the system $\mathcal{S}ys(\overline{s})$ and we compute the average value of the observed quantities over the simulations. Therefore, for each observation $o \in \mathcal{O}$ we compute an estimation $\frac{1}{N}\sum_{i=1}^{N} o(\pi_i)$ of the average value $(\tilde{o})_n$ after runs of length n.

To solve the optimization problem, we must determine the configurations in I that optimize (minimize or maximize) these quantities. When the problem is defined with several observable quantities, we are faced with a multi-objective problem, and the best configurations are then selected by computing the Pareto frontier of the set of observations.

Observe that the simulation approach allows us to distribute the computation if the architecture permits it.

3.2 Change Detection with CUSUM

Assuming a SDES $\mathcal{S}ys$, we consider the change $p \geq k$ `where` $p = Pr[\varphi]$ with φ a BLTL property and $k \in]0,1[$. Let $X_1, \ldots X_N$ be a finite set of samples collected during an execution π of $\mathcal{S}ys$. We note p_i the probability value at the sample X_i. We assume the probability initially satisfies $p_0 < k$. The problem is stated as:

- $H_0 : \forall\ 1 \leq i \leq N, p_i < k$ *i.e.* no change occurs
- $H_1 : \exists\ 1 \leq i \leq N$ such that the change occurs at time t: $\forall 1 \leq j \leq N$, we have $t_j < t \Rightarrow p_i < k$ and $t_j \geq t \Rightarrow p_i \geq k$.

In this paper, we will decide between those hypothesis by using the CUSUM algorithm [1]. Like the Sequential Probability Ratio Test (SPRT) [19,15], the CUSUM comparison is based on a likelihood-ratio test: it consists in computing the cumulative sum of the logarithm of the likelihood-ratio S_i over the sample sequence $X_1, \ldots X_i$ and detecting the change decision as soon as S_i satisfies the stopping rule.

$$S_i = \sum_{j=1}^{i} s_j \qquad\qquad s_j = \begin{cases} \ln \frac{k}{p_j}, & \text{if } X_j = 1 \\ \ln \frac{1-k}{1-p_j}, & \text{otherwise} \end{cases}$$

The typical behavior of the log-likelihood ratio S_i is a global decreasing before the change, and an increasing shape after the change. Then the stopping rule purpose is to detect when the positive drift is sufficiently relevant to detect the change. It consists in saving $m_i = \min_{1 \leq j \leq i} S_j$, the minimal value of CUSUM, and compare it with the current value. If the distance is sufficiently great, the stopping decision is taken, i.e., an alarm is raised at time $t_a = \min\{t_i : S_i - m_i \geq \lambda\}$, where λ is a sensitivity threshold.

In case there is no detection, we set $t_a = +\infty$. Note that we presented CUSUM monitoring for the case $p \geq k$, but it could be set up for $p \leq k$ by defining the stopping rule for the maximum value of CUSUM instead.

CUSUM Calibration: it is important to note that the likelihood-ratio test assumes that the considered samples must be independent. This assumption may be difficult to ensure over a single execution of a system, but several heuristic solutions to guarantee independence exist. One of them consists in finding a state frequently visited during the execution of the system. Collecting exactly one sample each time such a state is visited, ensures independence between samples. Another solution being to introduce delays between the samples.

The CUSUM sensitivity depends on the choice of the threshold λ. A smaller value increases the sensitivity, *i.e.* the false alarms rate. A false alarm is a change detection at a time when no relevant event actually occurs in the system. Conversely, big values may delay the detection of the changes. The false alarms rate of CUSUM is defined as $E[t_a]$, the expectation of the time before the CUSUM raises an alarm while the system is still running before the change occurs. Ideally, this value must be the biggest as possible $E[t_a] \to +\infty$. The detection delay is defined as the expectation time between the actual change of time t and the alarm time t_a of the CUSUM: $E[t_a - t \mid t < t_a]$. Ideally, this value has to be small as possible. In Section 5, we will propose an heuristic that uses the quantitative model checking problem in order to calibrate the algorithm.

The empirical way to choose the stopping rule: Theoretically, the properties of the CUSUM are based on the computation of the Average Run Length function (ARL) [1]. In a very few cases, this function may be computed or approximated using some approximating techniques (Wald or Siegmund) but most of the time, it is too complex to be used and to deduce λ. The alternative [18] consists in simulating $\mathcal{S}ys_0$, that is a version of the system for which the change does not occur, i.e., the probability will not reach the value k. The idea, which relies on a Monte-Carlo approach, consists in generating the CUSUM values for a huge set of different executions of $\mathcal{S}ys_0$. The objective being to characterize the duration of false alarms that may be triggered on those situation. In Section 5 we apply such a characterization for change-based BLTL.

Monitoring for Change Relation Satisfiability Let us consider the change relation γ based on $\tau_1, \ldots, \tau_n$ changes. Using the syntax introduced in Section 2.2, it is expressed as `let` τ_1 `and ... and` τ_n `in` γ, where γ contains Boolean operations over changes and BLTL formulas. We use the following monitoring procedure for each atom:

1. For each change τ_i, we set a CUSUM monitor that splits the monitoring into one sub-monitor for each random variable. Note that heuristics allows to reuse information between monitoring actions.
2. The proposition $\tau_i!$ holds iff $t_i \neq +\infty$. The proposition $\tau_i \diamond t$ holds iff $t_i \diamond t$. Similarly, the proposition $\tau_i \diamond \tau_j + t$ holds only if $t_i \diamond t_j + t$ but it is undefined if $t_i = t_j = +\infty$.
3. BLTL formulas can be monitored with classical techniques.

In practice, the tool generates monitors on demand for the given atoms and combine their answers in a Boolean manner.

4 Simulink and Plasma Lab: an Integration

The results presented in Section 3 have been implemented in the Plasma Lab SMC toolbox available at `https://project.inria.fr/plasma-lab/`. In this section, we first recap the main features of the tool, and then show how architecture of the implementation can be exploited in order to integrate Plasma Lab within Simulink, hence providing an in shell new verification theory for this widely used language.

4.1 On Plasma Lab

Plasma Lab is a compact, efficient and flexible platform for statistical model checking of stochastic models. The tool offers a series of SMC algorithms which includes rare events simulation, distributed SMC, non-determinism, or 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.

Figure 1 presents Plasma Lab architecture. More specifically, the relations between model simulators, property checkers, and SMC algorithms components. Simulators features include starting a new trace and simulating a model step by step. Checkers decide a property on a trace by accessing to state values. It also control the simulation, with a *state on demand* approach. A SMC algorithm component, such as the CUSUM algorithm, is a runnable object. The algorithm process is carried out by the run method. Progress and results are then notified through the Controller API. Samples are obtained by calls to the check function

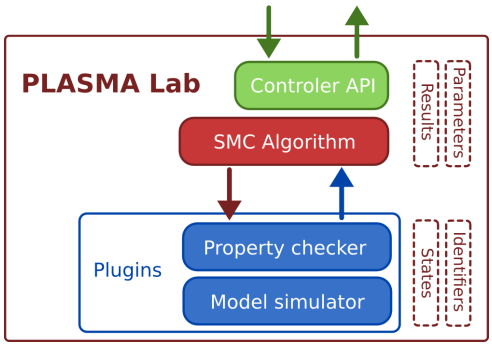

Fig. 1: Plasma Lab architecture

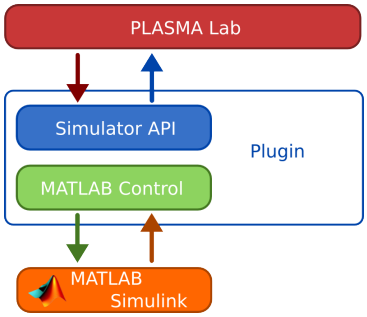

Fig. 2: Interface Plasma Lab- Simulink

of a checker component. Depending on the property language, a checker can return either Boolean or numerical values.

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, checker or algorithm component is pretty straightforward as they share a similar plugin architecture. Thus, it requires only a few classes and methods 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.

4.2 On Integrating Plasma Lab within Simulink

We now show how to integrate Plasma Lab within Simulink, hence lifting the power of our simulations approaches directly within the tool. We will focus on those Simulink models with stochastic information. The experienced reader shall observe that our approach is different from the one of Zuliani et al. [21] that consists in programming one SMC algorithm within the Matlab toolbox of Simulink. Indeed, the flexibility of our tool will allows us to incrementally add new algorithms to the toolbox without new programming efforts. Moreover, the user will directly use Plasma Lab within the Simulink interface, without tierce party. The approach is also different from the one in [6] that consists in translating parts of Simulink models into the Uppaal language (which makes it difficult to analyse counter examples). The reader shall observe that Plasma Lab for Simulink offers the first integrated verification tool for Simulink models with stochastic information.

Simulink is a block diagram environment for multidomain simulation and Model-Based Design approach. It supports the design and simulation at the system level, automatic code generation, and the testing and verification of embedded systems continuously . Simulink provides a graphical editor, a customizable

set of block libraries and solvers for modeling and simulation of dynamic systems. It is integrated within MATLAB. The Simulink models we considered have special extensions to randomly behave like failures. By default the Simulink library provides some random generators that are not compatible with statistical model checking: they always generate the same random sequence of values at each execution. To overcome this limitation we use some C-function blocks call that generate independent sequences of random draws.

Our objective was to integrate Plasma Lab as a new Simulink library. For doing so, we developed a new simulator plugin whose architecture is showed in Figure 2. One of the key point of our integration has been to exploit MATLAB Control, that is a library[1] allowing to interact with MATLAB from Java. This library uses a proxy object connected to a MATLAB session. MATLAB invokes, *e.g.* functions `eval`, `feval` ... as well as variables access, that are transmitted and executed on the MATLAB session through the proxy. This allowed us to implement the features of a model component, controlling a Simulink simulation, in MATLAB language. 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 the executions of a SDES, *i.e.* sequences of states and time stamps based on the set of state variables SV. This set must be defined by declaring in 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.

5 An Illustrative Case study

We applied all the three analyses presented on the Simulink model of a temperature controller in a pig shed. This model is inspired by similar studies [13,9,7]. The system under control is a pig shed equipped with a fan and a heater to regulate the air temperature. Air temperature in the shed is subjected to random variations due to the variation of external temperature and the variation of the number of pigs that produce heat. The objective of the controller is to counter these variations such that the temperature remains within a given comfort zone. To do so, the controller can activate the heater to increase the temperature, and the fan to bring external air and therefore cool the shed. Then the temperature T of the shed is given by the following differential equation:

$$T' = T_{ext} * Q - T * Q + W_{heater} + W_{pigs}$$

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

where T_{ext} is the external temperature, $Q = Q_{min} + Q_{fan}$ is the air flow created by a minimal flow Q_{min}, and an additional flow Q_{fan} when the fan is activated, W_{heater} is the heat produced by the heater, when activated, and W_{pigs} is the heat produced by the pigs. This equation is modeled by the Simulink subsystem of Fig. 3. The number of pigs is computed every 500 time units according to

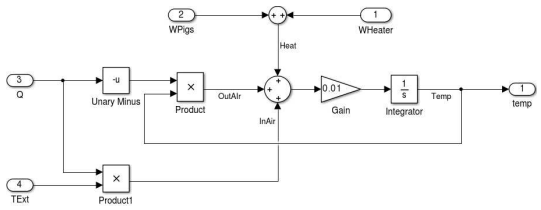

Fig. 3: Simulink model of the differential equation controlling the temperature

a Poisson law with $\lambda = 13$. The external temperature is modulated by two sinusoidal between $-1\,^{\circ}\mathrm{C}$ and $+25\,^{\circ}\mathrm{C}$, with periods 628 t.u and 62.8 t.u.

The controller that we study applies a *bang-bang* strategy that is specified by four temperature thresholds, that is (1) when the temperature goes above TFanOn, the fan is turned on, (2) when the temperature returns below TFanOff, the fan is turned off, (3) when the temperature goes below THeaterOn, the heater is turned on, (4) when the temperature returns above THeaterOff, the heater is turned off. This controller is implemented by Stateflow automata given in Fig. 4.

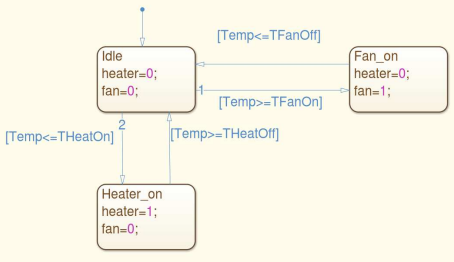

Fig. 4: Temperature controller

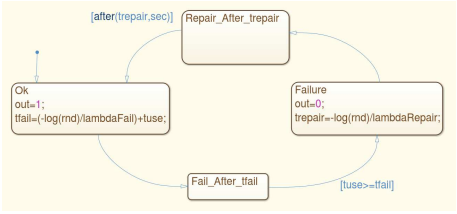

Fig. 5: Failure generator

The fan and the heater are subjected to random failures when they are in use. Exponential distributions control the occurrence time of a failure. After a failure a reparation process allows to restart the fan or the heater, but it also takes a random time, exponentially distributed. These failures are modeled by two Stateflow automata, as shown in Fig. 5. In this automaton, rnd is a random number between 0 and 1, and tuse is the duration of use of the fan or heater.

The timings tfail and trepair corresponds respectively to the time of next failure, and the repair time, each chosen according to an exponential distribution with parameter lambdaFail and lambdaRepair, respectively. Additionally, the failure rate increases with usage due to wear and tear. This continue until a replacement is performed, which reset the rate.

5.1 Quantitative Verification and Optimization

The controller goal is to maintain the temperature within a comfort zone specified by a minimum and a maximum temperature (resp. $\mathsf{T}_{\mathsf{min}} = 15\,^{\circ}\mathrm{C}$ and $\mathsf{T}_{\mathsf{max}} = 25\,^{\circ}\mathrm{C}$). We first consider the following values for the controller thresholds: $\mathsf{TFanOn} = 22\,^{\circ}\mathrm{C}$, $\mathsf{TFanOff} = 20\,^{\circ}\mathrm{C}$, $\mathsf{THeaterOn} = 18\,^{\circ}\mathrm{C}$ and $\mathsf{THeaterOff} = 20\,^{\circ}\mathrm{C}$.

We apply statistical model-checking to evaluate the efficiency of the controller both in the presence and absence of failures. The first BLTL property that we monitor checks that the system is not in discomfort for an excessive period of time. This is expressed by the following property:

$$\Phi_1 = G_{\leq t_1} F_{\leq t_2} \neg \mathsf{Discomfort}$$

where t_1 is the simulation time, t_2 is the accepted discomfort time, and Discomfort is a predicate that is true when the temperature of the system is outside the comfort zone. A dual of Φ_1 is to check for long periods without discomfort. This is possible with:

$$\Phi_2 = F_{\leq t_1} G_{\leq t_2} \mathsf{Discomfort}$$

Finally, a third BLTL property checks that each period of discomfort is followed by a period without discomfort:

$$\Phi_3 = G_{\leq t_1} \Big(G_{\leq t_2} \mathsf{Discomfort} \Rightarrow F_{\leq t_3} (G_{\leq t_4} \neg \mathsf{Discomfort}) \Big)$$

Here t_1 and t_2 are as previously, while $t_3 \geq t_2$ is the expected time at which the system returns to normal situation, and t_4 is the duration of the period without discomfort.

We use Plasma Lab to estimate the probability to satisfy these properties for different values of the timing constraints, on both models with and without failures. Each property is evaluated over a period of time $t_1 = 12000$ *t.u.* with precision $\epsilon = 0.01$ and confidence $\delta = 0.01$. Φ_1 and Φ_2 are evaluated for several values of t_2. Note that for $t_2 = 0$, Φ_1 resumes to checking $G_{\leq t_1} \neg \mathsf{Discomfort}$. Φ_3 is evaluated with $t_2 = 25$ *t.u.* and several values of t_3 and t_4.

The results for properties Φ_1 and Φ_2 are presented in Fig. 6. While the probabilities of satisfying Φ_1 show a significant difference between the models with and without failures, the results for Φ_2 are almost identical. This means that discomfort is as frequent in the two models, but it tends to last longer in the presence of failures. The results for Φ_3 are presented in Fig. 7. It shows again that the model without failures recovers quicker from a discomfort period.

Instead of estimating a probability using SMC techniques, we can compute the average value of two quantities in the model, namely the *discomfort time*,

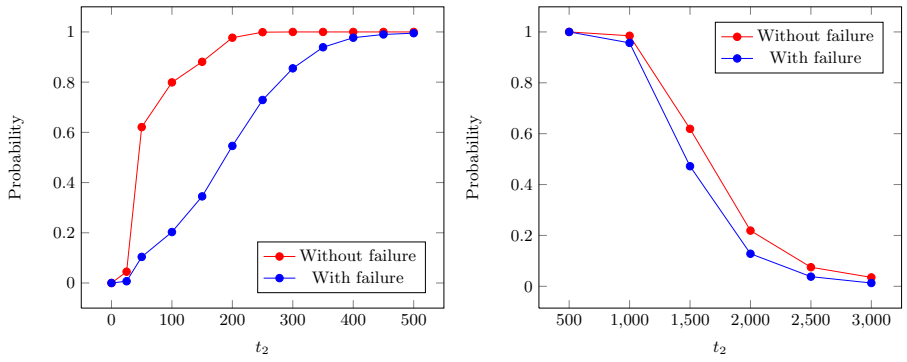

Fig. 6: Probability estimation with SMC of satisfying Φ_1 (left) and Φ_2 (right)

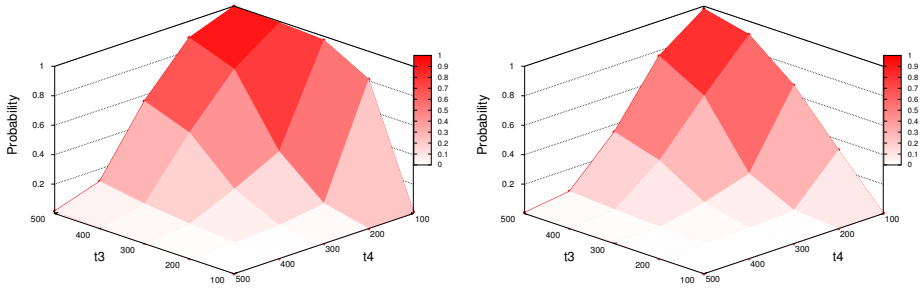

Fig. 7: Probability estimation with SMC of satisfying Φ_3 without failures (left) and with failures (right)

that is the cumulative time when the model is in a discomfort state, and the energy *cost*, computed with the duration of use of the heater and the fan. We aim at minimizing these two values by choosing adequate values of the model parameters.

Using Plasma Lab we can automatically instantiate the model with a range of values for the four temperature thresholds. We specify the ranges $[15, 20]$ for $\mathsf{THeaterOn}$ and $\mathsf{THeaterOff}$, and $[20, 25]$ for TFanOn and $\mathsf{TFanOff}$, with an increment of 1. We additionally specify the following constraints to select a subset of the possible values of the parameters: $\mathsf{TFanOff} < \mathsf{TFanOn}$, $\mathsf{THeaterOn} < \mathsf{THeaterOff}$, and $\mathsf{THeaterOn} < \mathsf{TFanOn}$.

Using these constraints Plasma Lab generates a set of 225 possible configurations, for each variant of the models, with and without failures. Each configuration is automatically analyzed with 100 simulations. We then plots the average values of the cost and the discomfort in Fig. 8. These graphs helps to select the

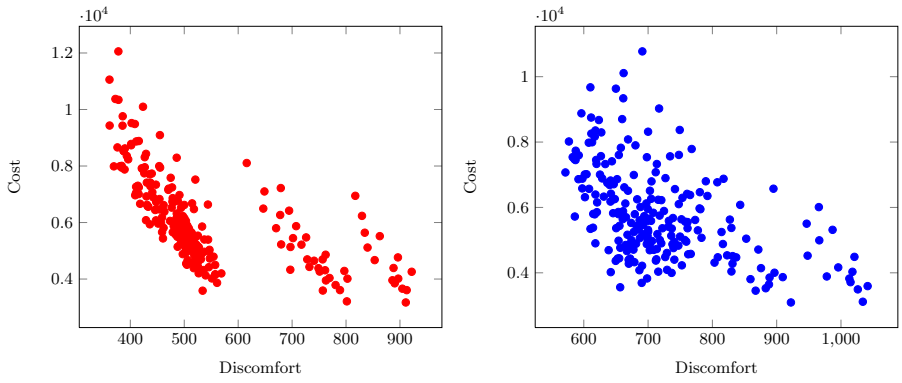

Fig. 8: Optimization of the thresholds parameters without failures (left) and with failures (right)

best values of the parameters by looking at the points that lies on the Pareto frontier of the data.

5.2 Change Detection

In our pig shed, the equipment may sometimes fail (heater or fan may break). In such situation, the shed may be too frequently in the discomfort zone, which may lead to the death of several pigs.

As we have seen, the probability of being in the discomfort zone is nominally very low. However, to avoid problems, one should be able to rise a flag as soon as the probability to be in the discomfort zones crosses a given threshold. Our objective is to detect that when such a change happens, there is a maintenance procedure that moves the shed out of the discomfort zone. In our example, this maintenance feature is modeled as a procedure that is regularly applied to the pig shed. Initially, the time between each maintenance is set to a very large value (500000 $t.u.$). The final objective is to set this time value in order to have an acceptable maintenance delay when the die risk is to heavy for the pigs (emergent behavior). This will be done by detecting changes.

We modeled the property using the change property language we proposed and we used CUSUM algorithm to check it. We first define τ to be the following change: " the probability to be in the discomfort zone more than $t_1 = 100$ $t.u.$ is greater than 0.35". We are now ready to propose a property that expresses that when the change occurs, then the maintenance must be done in less than $t_2 = 1000$ $t.u.$ Formally,

$$\phi_4 = \left| \begin{array}{l} \texttt{let} \quad \tau = \; p \geq 0.35 \quad \texttt{where} \\ \quad p \; \texttt{is} \; Pr\big[G_{\leq t_1}\mathsf{Discomfort}\big] \\ \texttt{in} \quad \tau! \Rightarrow F_{\leq \tau + t_2}\mathsf{Reparation} \end{array} \right.$$

In order to perform the analysis, the CUSUM algorithm needs a calibration step. We first require an estimate of p_0, the initial probability of being in the discomfort zone before the change occurs and the stopping sensitivity λ. To estimate p_0, we disable failures of the temperature regulation system (fans + heaters) in the shed model. The estimate of p_0 is obtained by a Monte Carlo simulation based on the monitoring of $G_{\leq t_1}\,\mathsf{Discomfort}$. The property was checked every 200 $t.u.$ over execution traces of length 21000 $t.u.$ After 630 sec. of analysis, Plasma Lab returns $p_0 \in [0, 0.05]$ with a confidence of 0.9.

Next step is to set the stopping rule, which is again done with a Monte Carlo approach. The objective is to observe several samples (value of the ratio) for several CUSUM. When there is no failure, the curve of samples should decrease. Indeed, it should only increases when failures happen, i.e., when the change happen. In practice, even without failure, the curve may locally increase for a short amount of time, which is due to the uncertainty introduced in the model. The objective is to characterize those local drifts to avoid false alarm.

To do so, we ran 100 executions of the CUSUM and observed 2000 samples per CUSUM (which corresponds to 201000 $t.u.$). From those experiments, we observed that the mean time (in CUSUM samples) between positive drift is 127.88 $t.u.$ and the mean duration of positive drift is 1.2 samples. The frequency of positive sample is thus $1.2/(127.88 + 1.2)$, which is in the interval $[0, 0.05]$ as predicted by Monte Carlo algorithm. In order to observe a real alarm one need to push this quotient to 0.35, which is the probability one wants to observe. This amounts to vary the duration of a positive sample, i.e., to replace 1.2 by a higher value in the above quotient. Doing so, we concluded that the probability will become greater than 0.35 when the positive drift is longer that 52 samples. From the definition of Φ_4, we compute that the drift is $\ln \frac{0.35}{0.05}$ for each positive sample. We finally set the stopping rule to $\lambda = 52 * \ln \frac{0.35}{0.05} \approx 101$.

We then launched the CUSUM on the model with failures over an execution of 210000 $t.u.$ for the property $G_{\leq t_1}\,\mathsf{Discomfort}$ checked every 200 $t.u.$. We applied Plasma Lab and observed that the stopping rule was satisfied after the sample 901 that corresponds to the simulation time 103473.34 $t.u.$ We reproduced the experiment several times (20): we observed the change occurred at 102543.23 $t.u.$ in average and in earlier $\approx$ 101000 $t.u.$ We conclude that to satisfy Property ϕ_4 the maintenance operation must be scheduled at 100000 $t.u.$

6 Conclusion

We have presented extensions of SMC, implemented within the Plasma Lab toolset. We have then shown the flexibility of Plasma by integrating it as a Simulink library. This integration constitutes one of the first proof of concept that SMC can indeed be integrated as feature library in a designed tool used in industry without compiling to an intermediary language. Finally, we have shown the utility of this integration on a large-size case study. Other examples are available at `https://project.inria.fr/plasma-lab/`. Future work include an integration of Plasma with the FMI standard in order to verify complex

heterogeneous systems. Another future work is to extend the power of distributed computing to Plasma-Simulink. The latter is technically challenging as it would require to duplicate compiled code to avoid license duplication and costs.

References

1. Basseville, M., Nikiforov, I.V.: Detection of Abrupt Changes: Theory and Application. Prentice-Hall, Inc. (1993)
2. Basu, A., Bensalem, S., Bozga, M., Delahaye, B., Legay, A.: Statistical abstraction and model-checking of large heterogeneous systems. STTT 14(1) (2012)
3. Biere, A., Heljanko, K., Junttila, T.A., Latvala, T., Schuppan, V.: Linear encodings of bounded LTL model checking. Logical Methods in Computer Science 2(5) (2006)
4. Boyer, B., Corre, K., Legay, A., Sedwards, S.: Plasma-lab: A flexible, distributable statistical model checking library. In: QEST. pp. 160–164 (2013)
5. Clarke, E.M., Faeder, J.R., Langmead, C.J., Harris, L.A., Jha, S.K., Legay, A.: Statistical model checking in biolab: Applications to the automated analysis of t-cell receptor signaling pathway. In: CMSB. pp. 231–250 (2008)
6. David, A., Du, D., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Smc for stochastic hybrid systems. In: HSB. pp. 122–136 (2012)
7. David, A., Du, D., Larsen, K.G., Legay, A., Mikucionis, M.: Optimizing control strategy using smc. In: NFM. pp. 352–367 (2013)
8. David, A., Larsen, K.G., Legay, A., Mikucionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In: CAV. pp. 349–355 (2011)
9. Grabiec, B., Traonouez, L., Jard, C., Lime, D., Roux, O.H.: Diagnosis using unfoldings of parametric time petri nets. In: FORMATS. pp. 137–151 (2010)
10. Havelund, K., Rosu, G.: Synthesizing monitors for safety properties. In: TACAS. pp. 342–356 (2002)
11. Hérault, T., Lassaigne, R., Magniette, F., Peyronnet, S.: Approximate probabilistic model checking. In: VMCAI, pp. 73–84 (2004)
12. Jégourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: CAV. pp. 576–591 (2013)
13. Jessen, J.J., Rasmussen, J.I., Larsen, K.G., David, A.: Guided controller synthesis for climate controller using uppaal tiga. In: FORMATS. pp. 227–240 (2007)
14. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to mc biological systems. In: CMSB. pp. 218–234 (2009)
15. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: An overview. In: Runtime Verification. pp. 122–135 (2010)
16. Parker D., Norman G., K.M.: The probabilistic model checker prism (jan 2012), `http://www.prismmodelchecker.org`
17. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: CAV. pp. 202–215 (2004)
18. Verdier, G., Hilgert, N., Vila, J.: Adaptive threshold computation for CUSUM-type procedures in change detection and isolation problems. Computational Statistics & Data Analysis 52(9) (2008)
19. Wald, A.: Sequential tests of statistical hypotheses. The Annals of Mathematical Statistics (1945)
20. Younes, H.L.S.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis, Carnegie Mellon (2005)
21. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to Stateflow/Simulink verification. FMSD 43(2), 338–367 (2013)