Usage-and Risk-Aware Falsification Testing for Cyber-Physical Systems

Kiviriga, Andrej; Larsen, Kim Guldstrand; Nickovic, Dejan; Nyman, Ulrik

Published in:
Formal Modeling and Analysis of Timed Systems

DOI (link to publication from Publisher):
10.1007/978-3-031-42626-1_9

Creative Commons License
Unspecified

Publication date:
2023

Document Version
Accepted author manuscript, peer reviewed version

Link to publication from Aalborg University

Citation for published version (APA):
Kiviriga, A., Larsen, K. G., Nickovic, D., & Nyman, U. (2023). Usage-and Risk-Aware Falsification Testing for Cyber-Physical Systems. In L. Petrucci, & J. Sproston (Eds.), Formal Modeling and Analysis of Timed Systems: 21st International Conference, FORMATS 2023, Antwerp, Belgium, September 19–21, 2023, Proceedings (pp. 141-157). Springer. https://doi.org/10.1007/978-3-031-42626-1_9

Usage-aware Falsification Testing for CPS

Andrej Kiviriga,[1] Kim Guldstrand Larsen,[1] Dejan Nickovic,[2] Ulrik Nyman[1]

[1] Aalborg University, Denmark
{kiviriga,kgl,ulrik}@cs.aau.dk
[2] AIT Austrian Institute of Technology, Austria
Dejan.Nickovic@ait.ac.at

Abstract. Verification of cyber-physical systems (CPS) is a challenging task. A considerable effort has been invested to develop pragmatic methods, such as falsification testing, which facilitate generation of inputs that lead to the violation of the CPS requirements. The resulting counterexamples are used to locate and explain faults and debug the system. However, CPS rarely operate in fully unconstrained environments and not all counterexamples have the same value – a fault resulting from a common usage of the system has more impact than a fault that is triggered by an esoteric input sequence. This aspect is neglected by the existing falsification testing techniques. We propose a new falsification testing methodology that is aware of the system's expected usage. Given a user profile model in the form of a stochastic hybrid automaton, an executable black-box implementation of the CPS and its formalized requirements, we provide a test generation method that (1) uses efficient randomized methods to generate multiple violating traces, and (2) estimates the probability of each counterexample, thus providing their ranking to the engineer.

Keywords: Cyber-physical systems · counterexample · black-box testing · randomized testing · falsification-based testing.

1 Introduction

Correctness is a crucial requirement during the design of safety-critical cyber-physical systems (CPS) such as smart homes, autonomous driving and intelligent medical devices. The interplay between computational (digital controllers, embedded software) and physical components (sensors and actuators), the increasing use of machine learning-based data-driven modules and the sophisticated interactions with unpredictable environments make the problem of correct and safe CPS design hard and challenging. Despite tremendous recent progress, formal verification does not scale yet to the size of realistic CPS applications. As a consequence, today's state-of-the-practice relies mainly on the more pragmatic simulation-based testing approaches.

The CPS community has invested in the recent past significant effort to improve the testing activities. Falsification-based testing (FBT) [15] is a popular method that renders the test generation process more systematic. FBT uses formal specifications with quantitative semantics to guide the system-under-test (SUT) to the violation of its requirements, whenever possible. It follows that FBT provides effective means to systematically detect a fault in the system. The resulting witness of the requirement violation is used to locate and explain the fault and hence to facilitate the system debugging task.

The classical FBT has a limitation – the test generation method focuses on finding one counterexample among possibly many of them. However, CPS often operate in partially constrained environments with certain assumptions on their usage in which not all counterexamples have the same value. For example, a fault triggered by a common usage of the system has much more impact then another fault resulting from some rare and esoteric input sequence. This is an important aspect for prioritizing debugging tasks under time and budget constraints and that is completely neglected by the existing FBT techniques.

We introduce in this paper a new methodology for *usage-aware* FBT to remedy the above situation. In our approach, we assume that: (1) a user profile model that describes the system's intended usage is given in the form of a stochastic hybrid automaton, (2) the SUT is provided in the form of an executable black-box implementation, and (3) the requirements are formalized using temporal logic.

We first use a randomized accelerator procedure to generate test inputs from the user profile model. We then feed the input vector to the SUT and execute it. We use the temporal logic monitor to detect potential violation of requirements. Whenever we find a counterexample, we use statistical model checking (SMC) [21, 18, 7], and more specifically importance splitting (IS) [11], to estimate the likelihood of the counterexample. The estimated probability of counterexamples enables us to rank them according to their likelihood, thus facilitating prioritization of the debugging tasks.

We instantiated our usage-aware FBT methodology (described in Section 2) with concrete methods and tools and implemented it in a prototype framework. We adopted UPPAAL SMC [5] to model stochastic hybrid automata (Section 3), MATLAB Simulink [6] to implement black-box SUTs and signal temporal logic (STL) [14] to formalize CPS requirements (Section 3). We developed a modified variant of the randomized reachability analysis (RRA) [12] procedure as our randomized accelerator for efficiently finding counterexamples (Section 4) and adapted a version of the IS algorithm to estimate counterexample probabilities (Section 5), integrating both methods to the UPPAAL SMC engine. We used MATLAB Simulink's simulation environment to execute generated input sequences and the RTAMT [16] runtime verification library to monitor the resulting simulation traces against STL requirements. We used a thermal model of a house as our case study (Section 6) to evaluate our approach (Section 7).

Related Work

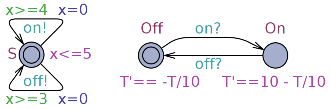
Falsification-based testing Falsification-based testing (FBT) [15] is a test generation method that uses formal specifications equipped with quantitative semantics to guide the search for behaviors that violate the formalized requirements. In that work, the authors propose to use deterministic assumptions for restricting the test search space. The test search space can be additionally restricted using symbolic reachability methods [4]. The classical FBT approaches also stop the generation of tests after finding the first violation of a requirement. The adaptive FBT method [3] remedies this situation by introducing the notion of specification coverage and providing means to generate multiple qualitatively different counterexamples. None of these works allow one to compare violation witnesses according to their likelihood to happen. To contrast, we introduce probabilistic user profile models of the SUT to enable ranking counterexamples.

Probabilistic Model Checking Counterexamples play an important role in probabilistic model checking and have received considerable attention in the last two decades, see [1] for a survey on methods for generating probabilistic counterexamples. We mention the early work from Han and Katoen [10], who originally propose a method for finding the strongest evidence, i.e. the most likely counterexample violating an until-specification as a hop-constrained shortest path problem. The tool DiPro [2] allows generating probabilistic counterexamples discrete time Markov chains, continuous time Markov chains and Markov decision processes. In the work on probabilistic model checking, the model of the SUT is available as a white-box, which allows precise computation of counterexample probabilities but limits the scalability of the approach to the systems of small size and complexity. In our approach, we consider black-box SUTs of arbitrary size and complexity, and use simulation-based methods to detect counterexamples and estimate their probabilities.

Statistical and Randomized Testing Statistical model checking (SMC) is a Monte Carlo simulation method used to estimate the probability of violating formal requirements. Reliable estimation of rare events remains difficult and is typically addressed by the *importance splitting* (IS) [17, 13]. IS divides the goal with small probability into a sequence of intermediate goals that are easier to reach. An alternative way to address the problem of rare-event simulation is to use *randomized reachability analysis* (RRA) [12]. RRA discards the stochastic semantics of the model to increase the chance of exercising a rare event. While RRA can efficiently find a counterexample, it cannot be used alone to estimate its probability. On the other hand, SMC and IS can reason about the probability that a given SUT violates a property, but are less appropriate to estimate the probability of a single counterexample. In our work, we use the synergies between SMC, IS and RRA to achieve efficient falsification while enabling the likelihood estimation of counterexamples.

2 Methodology

In this section, we describe our user-aware falsification-based testing methodology. The input to our approach are three artefacts: the *user profile* model, the black-box implementation of the *system-under-test (SUT)*, and a *formalized requirement*. The output of our approach is a list of input sequences (test cases) that lead to the falsification of the requirement ranked according to their estimated probability of happening.



$$\varphi = \Box_{[0,20]}(T \leq 75)$$

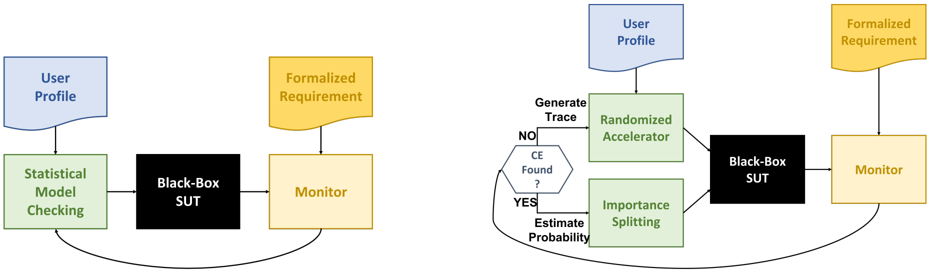
Fig. 1: Running example. User profile model (top left), the heater controller (top right), and the formalized requirement (bottom).

The user profile is a stochastic hybrid automaton that models the expected use of the SUT. It allows for rich and complex dynamics as well as stochastic behavior. Straightforward simulations of the user profile can be performed to generate inputs for the SUT. Generated simulations follow the underlying stochastic semantics of the model, which allows them to mimic the behavior of the real user and supports reasoning about the probability of generating a particular input sequence.

The SUT is a reactive dynamic system, which consumes an input sequence to generate another sequence of observable output quantities. We assume an executable black-box implementation of the SUT, whose behavior can be only observed at its input/output interface.

The formalized requirement is given in the form of a temporal logic specification. It defines the expected temporal and timing relations between input and output quantities and is used as an oracle to discriminate behaviors that satisfy the requirement from those that violate it.

To illustrate the different steps of the methodology, we use a simplified heat controller, depicted in Figure 1. The simplified heat controller follows a deterministic implementation (Figure 1 top right) has a single continuous state variable, the temperature T and consists of two discrete modes, `Off` and `On`. In the `Off` mode, the temperature decreases according to the differential equation $T' = -\frac{T}{10}$. Conversely, the temperature increases in the `On` mode according to the differential equation $T' = 10 - \frac{T}{10}$. The temperature range is limited to the interval between 0 and 100 degrees (not shown in the figure). The change between the two heater's modes is triggered by actions `on` and `off` provided by an external environment. We note that while the implementation is given in the form of a hybrid automaton for illustration purposes, we assume that it is seen as a black box to the tester, i.e. the tester can only provide the actions `off` and `on` and observe the temperature T. The stochastic user profile (Figure 1 top left) models the expected generation of the `off` and `on` actions. The clock x measures the time between two consecutive actions. After an action happens, x is reset to 0 and a time delay between 0 and 5 is sampled according to the uniform distribution. If the time delay is in the interval $[0, 3)$ no action is enabled and an additional time delay must be taken. If it is in the interval $[3, 4)$, the action `off` is taken with probability 1. If the time delay is in the interval $[4, 5)$, the actions `off` and `on` are triggered, each with probability 0.5. It follows that the action `off` is likely to happen three times more often then the action `on`. Finally, the formalized requirement φ (Figure 1 bottom) states that within 20 time units, the temperature T must continuously remain within 75 degrees.



(a) Baseline solution

(b) Usage-aware falsification

Fig. 2: Methodology workflow for falsification of black-box CPS.

Baseline Solution We first propose a straightforward baseline solution to estimate the probability of an error in the system is by using SMC [19, 20]. The core idea of SMC is to generate simulations of a stochastic model and then statistically analyze them to estimate the probability of the system to violate requirements with some degree of confidence. Figure 2 (a) shows the workflow of the SMC application to our case – the input sequences are generated by the user profile and the outputs from SUT are analyzed by SMC to conclude on probability estimate of an error.

Discovery of the most stubborn bugs in the CPS often requires generation of "exotic" input sequences. In non-trivial stochastic models, generating an "exotic" input can often be considered an extremely rare event, i.e. its probability being in range $[0; 10^{-100}]$. Therefore, we anticipate methods like SMC, which exercises the most likely behavior of the system, to be impractical due to the time required to generate enough simulations to achieve a reasonable statistical confidence.

In most real applications there is either a limit of resources for system verification or a requirement on the system being fail-proof up to a certain degree, as further bugs are more costly to fix than to replace a system. In both cases, it is crucial to focus resources on eliminating the most probable bugs first. While SMC estimates the overall existence of the bug, it cannot reason about a probability of any individual concrete input sequence. Hence, SMC does not help to conclude if a particular counterexample is of any concern in practice.

Efficient Solution We recall that our proposed methodology not only (1) allows us to identify violations efficiently but also (2) estimates the probability for each discovered counterexample. It utilizes the Randomized Accelerator (RA) which is a modified version of the randomized reachability analysis algorithm, initially proposed by [12] as an efficient error detection method for timed and stopwatch automata models. Among the modifications, we extend the algorithm to support the rich dynamics of hybrid automaton models to allow simulation of the user profile. In contrast to SMC, RA discards the underlying stochastic semantics to favor exploration of otherwise unlikely to reach parts of the model. As a consequence, RA cannot reason about the probability of its generated simulations, but excels at finding “exotic” traces fast. In their study, [12] have shown their randomized reachability analysis to be up to three orders of magnitude faster than SMC at discovering bugs.

The workflow for a single iteration of our methodology is shown in Figure 2 (b). The process starts by applying RA on the user profile to generate input sequences for SUT. The latter is simulated with the given input and its output is monitored w.r.t. to the property of interest. The process is repeated until the counterexample input sequence that violates the property is discovered. Moreover, RA exploits the information from previous runs to favor “more promising” parts of the user profile that are deemed to have affected the monitored property towards being violated.

The counterexample contains the information about the execution of the user profile – transitions taken in the hybrid automaton and the outputted dynamics. To reason about the probability of counterexamples we use the Importance Splitting (IS) from [13]. IS allows one to estimate probability of rare events which SMC cannot do reliably or quickly. We use IS to estimate the probability of following the qualitative trace from the user profile, i.e. focusing on executing transitions of the user profile model in the sequence commanded by the trace. Additionally, a run of IS generates a number of traces with different timing behaviors w.r.t to the stochastic semantics. Since the timing behavior might have a crucial impact on the property satisfaction, we check all the traces generated by IS against SUT to estimate the ratio of traces violating the property.

The counterexample and its estimated probability becomes the result of a single iteration of our proposed methodology. Each counterexample is ranked according to its estimated probability to occur in practice. The search can then continue in a similar fashion to discover more bugs.

3 Stochastic Hybrid Systems

In this section, we recall the definition of hybrid automata with stochastic semantics [8] that we use to model the user profiles. Let X be a finite set of continuous variables. A variable valuation over X is a mapping $v : X \to \mathbb{R}$. We write $\mathbb{R}^X$ for the set of valuations over X. Valuations over X evolve over time according to a delay function $F : \mathbb{R}_{\geq 0} \times \mathbb{R}^X \to \mathbb{R}^X$, where for a delay d and valuation v, $F(d, v)$ provides the new valuation after a delay of d. As is the case for delays in timed automata, delay functions are assumed to be time additive in the sense that $F(d_1, F(d_2, v)) = F(d_1 + d_2, v)$. To allow for communication between different hybrid automata we assume a set of actions Σ, which is partitioned into disjoint sets of input and output actions, i.e. $\Sigma = \Sigma_i \uplus \Sigma_o$.

Definition 1. *A Hybrid Automaton (HA) $\mathcal{H}$ is a tuple $\mathcal{H} = (L, l_0, X, \Sigma, E, F, I)$, where (1) L is a finite set of locations, (2) l_0 is an initial location s.t. $l_0 \in L$, (3) X is a finite set of continuous variables, (4) Σ is a finite set of actions partitioned into inputs (Σ_i) and outputs (Σ_o) s.t. $\Sigma = \Sigma_i \uplus \Sigma_o$, (5) E is a finite set of edges of the form (l, g, σ, r, l') where $l, l' \in L$, g is a predicate on R^X which* *acts as a guard that must be satisfied, $a \in \Sigma$ is an action label and u is a binary relation on $\mathbb{R}^X$ which acts as an update, (6) $F(l)$ is a delay function for each location $l \in L$, and (7) I assigns invariant predicates $I(l)$ to any location $l \in L$.*

The semantics of a HA $\mathcal{H}$ is a timed labeled transition system, whose states are pairs $(l, v) \in L \times \mathbb{R}^X$ with $v \models I(l)$, and whose transitions are either delay transitions $(l, v) \xrightarrow{d} (l, v')$ with $d \in \mathbb{R}_{\geq 0}$ and $v' = F(d, v)$, or discrete transitions $(l, v) \xrightarrow{a} (l', v')$ if there is an edge (l, g, a, u, l') such that $v \models g$ and $u(v, v')$. The effect of the delay function F may be specified by a set of ODEs that need to be solved and govern the evolution of the continuous variables in time.

We denote $\omega = s_0 d_1 a_1 s_1 d_2 a_2 \ldots$ to be a *timed word* where for all $i, s_i \in S$, $a_i \in \Sigma$, $s_i \xrightarrow{d_{i+1}} \xrightarrow{a_{i+1}} s_{i+1}$ and $d_i \in \mathbb{R}_{\geq 0}$. If ω is a finite timed word, we write $|\omega| = n$ to denote the length. We write $\omega[i]$ to denote a prefix run of ω up to i such that $w[i] = s d_1 a_1 s_1 \ldots d_i a_i s_i$. Last, we denote by $\Omega(\mathcal{H})$ the entire set of timed words over $\mathcal{H}$.

Figure 3 (a) shows the resulting evolution of the continuous temperature variable from the SUT from Figure 1 induced by the example input sequence (timed word) $3.7 \cdot$ off $\cdot\, 4.1 \cdot$ on $\cdot\, 3.1 \cdot$ off $\cdot\, 4.5 \cdot$ off $\cdot\, 4 \cdot$ off. The property φ is satisfied as the temperature stays under 75 degrees for the 20 time units of the simulation.

Hybrid Automata may be given a stochastic semantics by refining the non-deterministic choices of transitions and delays by probabilistic and stochastic distributions. For each state $s = (l, v)$ of HA $\mathcal{H}$ there exists:

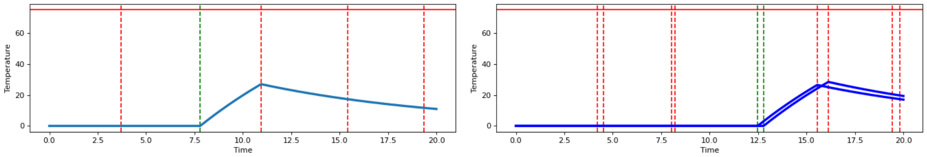
- the *delay density function* μ_s gives a probability distribution over delays in $\mathbb{R}_{\geq 0}$ that can be taken by a component, such that $\int \mu_s(t)dt = 1$,
- the *output probability function* γ_s gives a probability of taking an output $o \in \Sigma_o^j$ such that $\sum_o \gamma_s(0) = 1$, and
- the *next-state density function* η_s^a gives a probability distribution on the next state $s' = (l', v') \in \mathbb{R}^X$ given an action a such that $\int_{s'} \eta_s^a(s') = 1$.

Consider $\mathcal{H}$ to be a stochastic HA. For $s \in S$ and $a_1 a_2 \ldots a_k \in \Sigma^*$ we denote a *timed cylinder* $\pi(s, a_1 a_2 \ldots a_k)$ to be the set of all timed words from s with a prefix $t_1 a_1 t_2 a_2 \ldots t_k a_k$ for some $t_1, \ldots, t_n \in \mathbb{R}_{\geq 0}$. An infinite timed word $\omega = s_0 d_1 a_1^\omega s_1 d_2 a_2^\omega \ldots d_k a_k^\omega s_k \ldots$ belongs to the timed cylinder, written as $\omega \in \pi(s, a_1, a_2, \ldots, a_k)$, if $a_i^\omega = a_i$ for all i up to k and $s_0 = s$. Figure 3 (b) shows two timed words belonging to the same timed cylinder $\pi(s,$ off, off, on, off, off$)$, where $s = ($S, x=0$)$ is the initial state.

Providing the basic elements of a Sigma-algebra we now recall from [8] the inductively defined measure for such timed cylinders:

$$\mathbb{P}_{\mathcal{H}}(\pi(s, a_1 a_2 \ldots a_k)) = \int_{t \geq 0} \mu_s(t) \cdot \gamma_{s^t}(a_1) \cdot \int_{s'} \left(\eta_{s^t}^{a_1}(s') \cdot \mathbb{P}_{\mathcal{H}}(\pi(s', a_2 \ldots a_k)) ds' \right) dt \qquad (1)$$

The probability of following a timed cylinder π is computed by integrating over the initial delays t in the outermost level. Next, we take the probability of



(a) Timed word example

(b) Two timed words in the same cylinder

Fig. 3: A timed word with `on` and `off` actions represented by green and red vertical dashes lines, respectively. The temperature dynamics of SUT (blue line(s)) and φ threshold of 75 degrees (red line) is shown.

outputting a_i. The last part integrates over all successors s' and takes a product of probabilities for stochastic state changed after taking the delay t and output a_1, and the probability of following the remainder of the timed cylinder.

A general system can represented as a network of HA. Under the assumption of input-enabledness, an arbitrary number of HA can be composed into a network where the individual components communicate with each other and all together act as a single system. A race-based stochastic semantics determines which of the components in the network gets to perform an output such that the winning component is the one with the smallest chosen delay. Here we skip the definition of networks of HA and their stochastic semantics, and refer the interested reader to [8] for in-depth details. We now proceed with formalized requirements for HA.

Let $\mathcal{F}(\omega_{\downarrow}) = y$ be a function representing a black-box nonlinear hybrid system that gives a real-valued output y on the given projection of a timed word $\omega \in \Omega(\mathcal{H})$ of a stochastic HA $\mathcal{H}$, denoted as $\omega_{\downarrow}$. which represents a simulation output of the model $\mathcal{H}$ w.r.t. to the stochastic semantics.

Our formal property φ is expressed in a Signal Temporal Logic (STL) language [14]. A recap of STL is provided in Appendix A. STL supports quantitative semantics [9] with the help of function $\rho(\varphi, y, t)$ which gives *robustness*, i.e. degree of satisfaction of the formula φ for the input y at time t. The formula is satisfied if the robustness is positive and vice versa. Given a *robustness* function ρ, when $\rho(\varphi, y)$ is positive it indicates that y satisfies φ, written as $y \models \varphi$.

Let **Error** $\in S$ be a set of error states of HA $\mathcal{H}$. An infinite timed run $\omega = s_0 \xrightarrow{d_1}\xrightarrow{a_1} s_1 \xrightarrow{d_2}\xrightarrow{a_2} \ldots$ is an *error run* if $\exists i. \exists \tau \leq d_i. \, s^{\tau}_{i-1} \in \textbf{Error}$, where $\tau \in \mathbb{R}_{\geq 0}$ and s^{τ}_{i-1} is the state such that $s_{i-1} \xrightarrow{\tau} s^{\tau}_{i-1}$. We say that ω has an error at i'th transition and show it as $\mathcal{F}(\omega[i]_{\downarrow}) \not\models \varphi$. In this case, clearly any ω' such that $\omega' = \omega[i]\omega_n$ is also an error run. We now proceed to explain the next step – generation of violating traces.

4 Falsification Testing with Randomized Accelerator

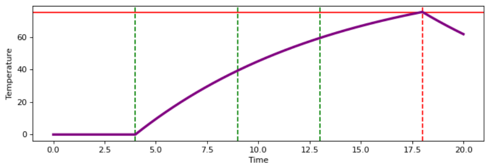


Fig. 4: A violating timed word (counterexample).

In this section, we describe our FBT approach for discovery of traces that violate requirements. In order to efficiently find counterexamples, we use the modified version of the randomized reachability analysis (RRA) initially proposed by [12] as a lightweight, quick and efficient error detection technique for timed systems. The core idea of RRA is to discard the underlying stochastic semantics of the model in an attempt to exercise an "exotic" behavior faster than with Uppaal SMC. The method is based on exploring the model by means of repeated *random walks* that operate on concrete states and delays, and avoid expensive computations of symbolic, zone-based abstractions. More formal details about RRA, including the algorithm, are given in Appendix B.

The result of RRA simulation is a finite timed word $w \in \Sigma$ of HA $\mathcal{H}$ where the last state satisfies some simulation termination condition (e.g. time bound). For our running example, Figure 4 shows a RRA generated timed word that violates the requirement φ. In this study we use RRA as our RA (randomized accelerator from Section 2) as a mean of accelerating discovery of rare events in the user profile model. In the remainder of this section, we discuss improvements over the vanilla version of the RRA algorithm.

The RA from our methodology is applied for a number of iterations until a violating trace (counterexample) ω is discovered, i.e. until the robustness of the formula becomes negative $\rho(\varphi, \omega_{\downarrow}) < 0$. Clearly, in practice not all of the RA generated traces will violate the property, but some of them are likely to be close. Since evaluation of the generated input against SUT is expensive, we are interested in minimizing the number of RA iterations. Thus, we guide our RA towards the areas of the state space of HA which are deemed to be "promising" according to the previous runs. More specifically, for non-violating runs we analyze the robustness of the output and search for robustness that lays outside its standard deviation and towards a violation. The corresponding actions of HA are then identified and prioritized over other actions in the following RA iteration. To avoid actions that lead to robustness local minima, the "promising" actions are used only for a single iteration of our methodology and are discarded if the property is not violated. Afterwards, an unguided run of RA is carried out and new promising transitions are recorded for the next RA iteration in the similar fashion. Hence, only every even non-violating run of RA is guided.

Short counterexamples are not only more probable to occur in practice but also easier to debug. For those reasons, we employ an adaptive simulation duration (ASD) that may change with each discovered counterexample. First, a user profile is simulated for an initial duration. When a counterexample is found, we detect the time of first violation. In the next iteration the simulation duration is limited to that time. Intuitively, this is similar to the search for the shortest trace. However, at some point we may end up reducing the simulation time to a point where no counterexamples exist. Hence, we increase the current simulation time by 10% after a certain number of iterations is spent without finding a violation. We define that number of iterations to be the average number of iterations among so far discovered counterexamples plus a constant to ensure "enough" effort is spent before increasing the simulation duration.

5 Estimating Counterexample Probability

The probability of a timed cylinder is given by Equation 1. Unfortunately, it is a theoretical construct that we cannot compute efficiently in practice. Alternatively, we can use simulation-based methods, such as SMC, to estimate the probability of following a timed cylinder. However, for rare events, SMC cannot estimate such probability reliably as simply too many simulations would be required to achieve a reasonable confidence. As a solution to this problem we use a rare event simulating technique known as *importance splitting* (IS).

The idea of IS is to split the final reachability goal δ_f into a number of intermediate sub-goals $\delta_1, \delta_2, \dots, \delta_n$ that eventually lead to the main goal, i.e. $\delta_n = \delta_f$. The sub-goals are called *levels* that each get closer to the goal which naturally can be ensured by a *score function.* The score of each subsequent goal is required to be larger than that of a previous one. In our case a score function is binary function that helps to ensure that the timed cylinder is followed, i.e. the action transitions are taken strictly in a sequence defined by π.

The probability estimate of reaching the level i from level $i-1$ is then $\frac{\texttt{Successors}_i}{m}$, where $\texttt{Successors}_i$ are successors of level i obtained from level $i-1$ by following action a_i from the timed cylinder, and m is a fixed number of simulations performed at each level. Repeating this process for all levels n allows to estimate a probability of a timed cylinder of a HA $\mathcal{H}$ as follows:

$$\mathbb{P}_{\mathcal{H}}(\pi(s, a_1 a_2 \dots a_n)) \simeq \prod_{i=1}^{n} \frac{|\texttt{Successors}_i|}{m} \tag{2}$$

With the help of IS, we can now estimate the probability of the timed cylinder $\mathbb{P}(\pi(s, a_1, a_2, \dots, a_n))$ which was generated by our RA. To ease the notation we sometimes write (π_n) instead of $\pi(s, a_1, a_2, \dots, a_n)$. A run of IS also produces a number of concrete simulations that all follow the timed cylinder and are generated according to the stochastic semantics of the underlying HA. During a run of IS, each successor has its predecessor recorded together with the delay and broadcast action that lead to that successor. The concrete simulations are obtained by back-tracing from the successors that made it to the very last level and back to the starting state. We define a finite set of timed words (traces) w generated by IS from a timed cylinder π_n as $\Gamma_{\pi_n} \subseteq \pi_n$ such that $|\Gamma_{\pi_n}| \leq m$, where m is the effort allocated per level of IS fixed effort scheme.

Even though all traces in Γ_{π_n} follow the same timed cylinder, the differences in the timing behavior (chosen delays) may influence the dynamics of the user profile to a large degree. Violation (or satisfaction) of the property monitored for SUT - as well as the level at which a violation appear - is therefore not guaranteed to be identical for all traces $\varGamma_{\pi_n}$ reported by IS.

We first estimate the probability of violating the STL property with a timed cylinder π_n, i.e. $\mathbb{P}(\omega \in \pi_n \wedge \mathcal{F}(\omega_\downarrow) \not\models \varphi)$. In the following let m be a number of IS simulation per level and let $\texttt{Successors}_i$ be the successors of each level i in IS. Considering the IS sampled traces $\varGamma_{\pi_n}$ and their (varying) violation of the monitored property, we may estimate this probability as follows:

$$\begin{aligned}\mathbb{P}(\omega \in \pi_n \wedge \mathcal{F}(\omega_\downarrow) \not\models \varphi) = \mathbb{P}(\pi_n) \cdot \mathbb{P}(\omega \in \pi_n \wedge \mathcal{F}(\omega_\downarrow) \not\models \varphi \mid \pi_n) \simeq \\ \prod_{i=1}^{n} \left(\frac{|\texttt{Successors}_i|}{m} \right) \cdot \left(\sum_{\omega \in \varGamma_{\pi_n}} \mathcal{F}(\omega_\downarrow) \not\models \varphi \right) \cdot \frac{1}{|\varGamma_{\pi_n}|}\end{aligned} \tag{3}$$

However, the above equation does not take into account that for a given trace ω, the property can be violated earlier than at the very last step n, i.e. $\mathcal{F}(\omega'_\downarrow) \not\models \varphi$, where $\omega' \subset \omega$ and $|\omega'| < |\omega|$. Furthermore, performing IS on the timed cylinder of length n may lose information about timed sub-cylinders of shorter length k, $|\pi_k| < |\pi_n|$. We will write $\mathcal{F}_k(\omega_\downarrow) \not\models \varphi$ if k is the minimal index such that $\mathcal{F}(\omega[k]_\downarrow) \not\models \varphi$ and $\mathcal{F}(\omega[k-1]_\downarrow) \models \varphi$. Also let $\varTheta$ be the function that given a finite set of traces $\varGamma$, a property φ and an index i returns only prefix words $\omega[i]$ such that the first violation occurs at step i, formally defined as:

$$\varTheta(\varGamma, \varphi, i) = \{\omega[i] \mid \omega \in \varGamma \wedge \mathcal{F}_i(\omega_\downarrow) \not\models \varphi \wedge \mathcal{F}_{i-1}(\omega_\downarrow) \models \varphi)\} \tag{4}$$

Now taking the level of occurrence of property violation into account leads to a higher error probability that may be estimated as follows:

$$\begin{aligned}\mathbb{P}(\omega \in \pi_k \wedge \mathcal{F}_k(\omega_\downarrow) \not\models \varphi \wedge k \leq n) = \sum_{k=1}^{n} \Big(\mathbb{P}(\pi_k) \cdot \mathbb{P}(\omega \in \pi_k \wedge \mathcal{F}_k(\omega_\downarrow) \not\models \varphi \mid \pi_k) \Big) \simeq \\ \sum_{k=1}^{n} \left(\left(\prod_{i=1}^{k} \frac{|\texttt{Successors}_i|}{m} \right) \cdot \frac{|\varTheta(\varGamma_{\pi_k}, \varphi, k)|}{|\texttt{Successors}_k|} \right)\end{aligned} \tag{5}$$

The key idea of this approach is to separate a timed cylinder of length n into sub-cylinders of length k such that $|\pi_k| \leq |\pi_n|$. Summing up the probabilities of sub-cylinders π_k and violation occurring at the very last step k gives an upper bound of the probability estimate for the timed cylinder to violate the property. However, this formula requires not only to compute the set of traces $\varGamma_{\pi_k}$ for each k, but to also check each of the traces $\omega \in \varGamma_{\pi_k}$ against the Simulink model. With latter of the steps being the most computationally demanding in our proposed methodology, we believe that equation 5 will be too expensive in practice. Instead, we use another lower bound probability estimate:

$$\sum_{k=1}^{n} \left(\left(\prod_{i=1}^{k} \frac{|\texttt{Successors}_i|}{m} \right) \cdot \frac{|\varTheta(\varGamma_{\pi_n}, k, \varphi)|}{|\texttt{Successors}_k|} \right) \tag{6}$$

It requires computing only the traces Γ_{π_n} for the main cylinder. This approach gives a smaller, lower bound probability estimate, but it is much cheaper to compute in practice.

For our running example, we consider 100 traces generated by IS (Figure 5) for a timed cylinder with `on`, `on`, `on`, `off`, `on` action sequence. Violation of 75 degrees threshold is observed in 9 (out of 100), all at step 4. The probability estimate of Equation 6 is then $\left(\frac{23}{100}\cdot\frac{27}{100}\cdot\frac{25}{100}\cdot\frac{77}{100}\right)\cdot\frac{9}{100}\approx 0.001$, where the parenthesis give IS probability estimate of the cylinder with 4 steps.

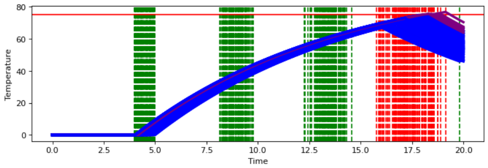


Fig. 5: One hundred traces generated by IS. Blue line traces satisfy the property and purple line ones violate.

6 Case Study

We use a Thermal Model of a House[3] as our black-box SUT. The model, shown in Figure 6 accounts for the heating system, thermal dynamics of the house, the outdoor environment and a number of associated properties such as house geometry, materials' thermal resistance, heater flow rate and hot air temperature. The heating of the house is controlled by the thermostat that turns on/off the heater once the temperature is below/above specified thresholds. The inside temperature is calculated by considering the heat flow from the heater and the heat losses to the environment through the insulation of the house.

To simulate the outdoor environment for the thermal house system, we use the HA weather model which provides "realistic" complex dynamics of the potential temperature. The model is shown in Figure 7. The daily and yearly temperature fluctuations are modelled by sinusoidal waves with varying phase, amplitude and biases. In our experiments we simulate the weather model for a period of 1 year while the starting period of the simulation being January 1st; therefore, the waves are adjusted accordingly and depend on an elapsing, observing clock x that we use to model the time in hours. In addition to daily and yearly temperature changes the model supports small and large "anomalous" temperature fluctuations that are enforced to occur at least every so often by

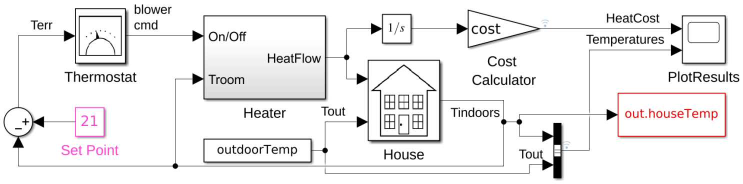


Fig. 6: Simulink Thermal Model of a House.

[3] https://se.mathworks.com/help/simulink/slref/thermal-model-of-a-house.html

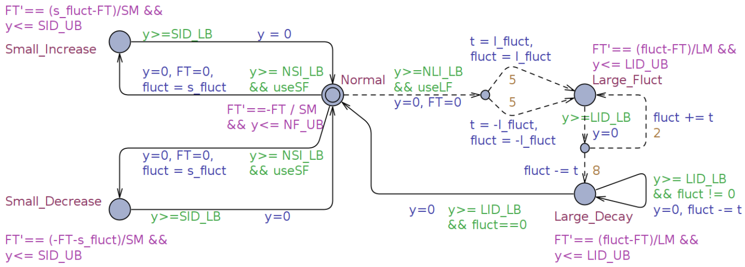


Fig. 7: Weather profile hybrid automaton model.

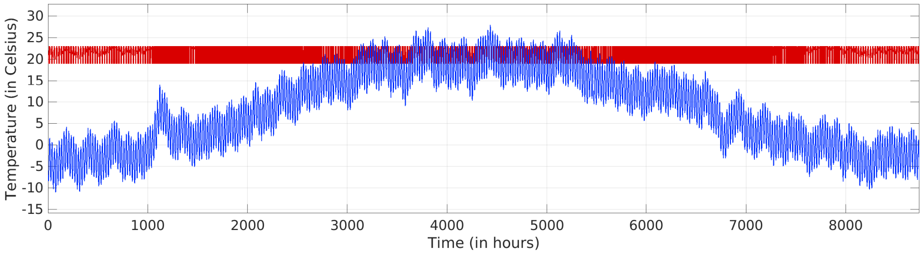


Fig. 8: Simulink Thermal House indoor (red) and outdoor (blue) temperatures simulated for 1 year (8736 hours).

`guards` and `invariants` on edges and locations, respectively. Each fluctuation results in a temperature change of the magnitude and dynamics governed by ODEs. A number of large fluctuations can happen sequentially, representing e.g. a heat wave or a sudden temperature drop. The type of a fluctuation, as well as its duration, which largely affects the dynamics, is decided during the simulation and in accordance with the stochastic semantics described in Section 3. The likelihood of an additional large fluctuation taking place right after a previous one is defined by discrete probabilities 8 and 2 on the outgoing transition from the `Large_Fluct` location. Even though no restrictions are made on the maximal number of large fluctuations happening sequentially, the probability of additional n sequential large fluctuations (after the initial one) is $(\frac{2}{8+2})^n = 0.2^n$.

Finally, as a property monitor we use python library RTAMT [16] which supports both offline and online monitoring of STL properties. The inside temperature of the house is monitored in an offline setting to measure the property robustness. We monitor the property $\Box(T \leq 16 \rightarrow \Diamond_{[0,24]}(T \geq 18))$, i.e. it is required that the temperature, if it drops below 16 degrees, always recovers to at least 18 degrees within 24 hours.

7 Experiments

We use SMC as a baseline and compare it to the performance of our falsification methodology. The two approaches are rather different as SMC cannot reason

Fig. 9: Number of simulations required to produce an approximation interval $[p - \epsilon, p + \epsilon]$ with confidence $1 - \alpha$ using Chernoff–Hoeffding inequality.

Confidence	Probability uncertainty ϵ			
α	0.05	5×10^{-3}	5×10^{-4}	5×10^{-5}
0.1	600	59,915	5,991,465	599,146,455
0.05	738	73,778	7,377,759	737,775,891
0.01	1,060	105,967	10,596,635	1,059,663,474

about probability of each individual violating trace. Rather, SMC estimates an overall property violation probability which lays within some approximation interval $p \pm \epsilon$ with a confidence $1 - \alpha$. The amount of simulations N required to produce an approximation interval given ϵ and α can be computed using Chernoff–Hoeffding inequality with $N \geq \frac{\log(2/\alpha)}{2\epsilon^2}$. To accurately estimate a very improbable error in the system, the probability uncertainty ϵ must be sufficiently small. As can be seen in Table 9, the growth of the required simulations is logarithmic and exponential in relation to α and ϵ, respectively. However, in practice this approach is too conservative. As an alternative, Uppaal SMC uses a sequential approach of Clopper-Pearson that computes the approximation interval with each iteration (for given α) and until the target ϵ is reached. Moreover, the further away a true probability is from $\frac{1}{2}$, the fewer simulations are needed. Empirical evidence[4] suggests that a true probability in range $[0, 10^{-5}]$ in practice requires roughly around 10% simulations (depending on α) of what Chernoff–Hoeffding inequality suggests. Nonetheless, for $\alpha = 0.01$ and $\epsilon = 5 \times 10^{-4}$ around as many as 10^6 simulations would be required.

In our experiments we only give an estimate of the time required by SMC to derive approximation intervals for a sufficiently small ϵ and α as each simulation is costly due to the execution of the Simulink model. To estimate the time of one iteration of the workflow from Figure 2a we ran 20,000 SMC simulations with 220 of them violating the property. With the total time of 17h 36m 20s, a single iteration in average takes 3.169 seconds. Figure 8 gives black-box SUT dynamics of one such iteration that includes outdoor ambient temperature (input) and a resulting indoor temperature (output) obtained after execution of SUT. As Uppaal SMC primarily exercises common behavior of the weather profile, we observe no requirement violation in the output from SUT.

In addition to IS that is used to estimate the probability to follow a timed cylinder, we implement an exact method for probability computation. This method supports only a subset of HA models where all clocks are reset in every transition (this is the case for our weather profile). It enables us to compare how far the estimated probability is away from the true one.

We perform an experiment to determine to which degree the variance in the probability estimates produced by IS is affected by the number of simulations m performed per level by IS. The results are reported in Appendix D and do not show any clear pattern indicating the amount of IS simulations per level to significantly affect the variance of the probability estimates. Since the execution

[4] https://docs.uppaal.org/language-reference/requirements-specification/ci_estimation/

Table 1: Falsification of the case study model following the workflow from Section 2. Given are 20 counterexamples ranked according to the “Complex” probability from Equation 6. Time is given in HH:MM:SS. DO is Discovery Order, RE is Risk Estimate, Tr. is transitions, and RAIt is RA iteration.

DO	Exact cylinder pr.	IS pr. mean	Trace ratio (TR)	IS pr. · TR (Equation 3)	Complex pr. (Equation 6)	Expected Severity	Risk Estimate	# Tr.	# RAIt	Execution time
2	1.7778e-03	2.4578e-03	49/84	1.4337e-03	1.4337e-03	0.568	8.1505e-04	4	13	00:02:14
4	1.9753e-04	2.6430e-04	15/19	2.0866e-04	2.0866e-04	0.761	1.5882e-04	6	50	00:01:21
5	3.5556e-04	1.2879e-03	100/100	1.2879e-03	5.7907e-04	0.253	1.4648e-04	9	17	00:00:59
17	1.9753e-04	1.8976e-04	14/19	1.3982e-04	1.3983e-04	0.709	9.9173e-05	6	13	00:00:41
3	8.8889e-05	4.0117e-05	18/18	4.0117e-05	7.5893e-05	0.918	6.9685e-05	5	12	00:00:37
19	3.5117e-04	1.6867e-04	42/78	9.0821e-05	9.0821e-05	0.513	4.6573e-05	8	74	00:03:09
8	3.1605e-05	7.7357e-05	70/70	7.7357e-05	1.6036e-04	0.275	4.4050e-05	8	19	00:00:46
6	7.9012e-05	1.0264e-04	49/86	5.8480e-05	5.8480e-05	0.543	3.1783e-05	7	84	00:03:27
9	1.7558e-05	2.3601e-05	24/24	2.3601e-05	3.0574e-05	0.948	2.8976e-05	9	27	00:01:09
7	8.7791e-05	7.9734e-05	11/22	3.9867e-05	3.9867e-05	0.475	1.8933e-05	8	35	00:01:16
11	3.1215e-05	1.5124e-05	100/100	1.5124e-05	8.9287e-06	0.345	3.0768e-06	13	11	00:01:06
16	6.3210e-06	2.4243e-06	77/77	2.4243e-06	2.0218e-05	0.125	2.5244e-06	9	191	00:03:43
10	1.5607e-06	7.8735e-07	24/24	7.8735e-07	4.0096e-06	0.623	2.4987e-06	12	11	00:00:35
15	6.3210e-06	3.9281e-06	100/100	3.9281e-06	1.9927e-05	0.096	1.9126e-06	13	18	00:00:32
20	6.2430e-06	4.6763e-06	100/100	4.6763e-06	8.0270e-06	0.150	1.2076e-06	13	130	00:02:53
13	3.5117e-06	5.0046e-06	40/82	2.4413e-06	2.4413e-06	0.455	1.1116e-06	10	9	00:02:09
12	7.9012e-07	1.0644e-06	20/20	1.0644e-06	1.0644e-06	0.945	1.0058e-06	10	10	00:00:41
18	1.9753e-05	2.1648e-06	8/16	1.0824e-06	1.0824e-06	0.477	5.1650e-07	7	21	00:00:48
14	1.7558e-07	4.4243e-07	12/25	2.1237e-07	2.1237e-07	0.444	9.4344e-08	12	2	00:00:39
1	1.5247e-55	1.4746e-56	100/100	1.4746e-56	5.4000e-06	0.010	5.2683e-08	151	1	00:00:28
Total:	3.2603e-03	4.7276e-03	-	3.4321e-03	2.8886e-03	-	1.4735e-03	-	748	00:29:23
Time estimate for SMC ($\alpha = 0.01$, $\epsilon = 5 \times 10^{-3}$) 10^4 iterations (based on Table 9).										**08:48:00**

time taken is roughly proportional to the number of IS simulations, in further experiments we fix the number of IS simulations per level to 100.

Finally, we evaluate our FBT methodology on the proposed case study and report the results in Table 1. Due to ASD, the quality of discovered errors tends to increase over time as the length (# transitions) of the counterexample decreases. The probability estimate of Equation 6 (E9) tends to be smaller for longer traces than that of Equation 3 (E6); however, for short traces E9 and E6 are equal as the error occurs at the very last transition of the trace. Discovery, evaluation and ranking of 20 bugs with our falsification methodology is an order of magnitude faster than an estimated performance of SMC to conclude on the overall likelihood of a bug in SUT.

8 Conclusion and Future Work

We introduced in this paper a new methodology for usage-aware FBT of CPS. It combines stochastic HA modeling, randomized reachability analysis, statistical model checking, importance splitting and runtime verification to efficiently generate input sequences that lead to the violation of the requirements, while estimating their probability of happening in the real usage of the system. We believe that the proposed methodology can significantly help the debugging effort by enabling to prioritize bugs with higher impact.

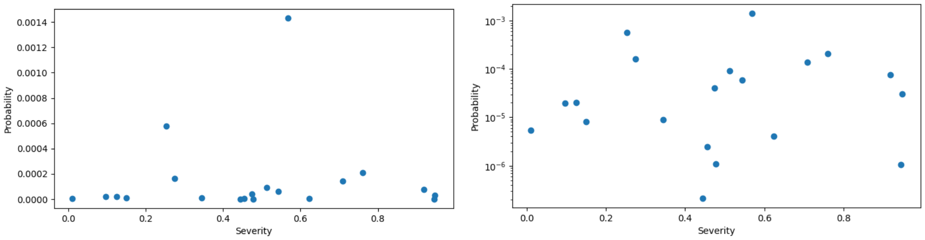


Fig. 10: Expected severity against probability plotted for each of the 20 counterexamples from Table 1. Plotted as linear scale (left) and logarithmic scale (right).

As future work, we plan to 1) develop more sophisticated guiding for our randomized accelerator, 2) introduce a state coverage metric, and 3) explore symbolic reachability techniques for HA.

References

1. Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J., Wimmer, R.: Counterexample generation for discrete-time markov models: An introductory survey. In: Formal Methods for Executable Software Models - 14th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2014, Bertinoro, Italy, June 16-20, 2014, Advanced Lectures. pp. 65–121 (2014)
2. Aljazzar, H., Leitner-Fischer, F., Leue, S., Simeonov, D.: Dipro - A tool for probabilistic counterexample generation. In: Model Checking Software - 18th International SPIN Workshop, Snowbird, UT, USA, July 14-15, 2011. Proceedings. pp. 183–187 (2011)
3. Bartocci, E., Bloem, R., Maderbacher, B., Manjunath, N., Nickovic, D.: Adaptive testing for specification coverage in CPS models. In: 7th IFAC Conference on Analysis and Design of Hybrid Systems, ADHS 2021, Brussels, Belgium, July 7-9, 2021. pp. 229–234 (2021)
4. Bogomolov, S., Frehse, G., Gurung, A., Li, D., Martius, G., Ray, R.: Falsification of hybrid systems using symbolic reachability and trajectory splicing. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019. pp. 1–10 (2019)
5. Bulychev, P.E., David, A., Larsen, K.G., Mikucionis, M., Poulsen, D.B., Legay, A., Wang, Z.: UPPAAL-SMC: statistical model checking for priced timed automata. In: Proceedings 10th Workshop on Quantitative Aspects of Programming Languages and Systems, QAPL 2012, Tallinn, Estonia, 31 March and 1 April 2012. pp. 1–16 (2012)
6. Chaturvedi, D.K.: Modeling and simulation of systems using MATLAB® and Simulink®. CRC press (2017)
7. Clarke, E.M., Zuliani, P.: Statistical model checking for cyber-physical systems. In: Automated Technology for Verification and Analysis, 9th International Sym-

posium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings. pp. 1–12 (2011)

8. David, A., Du, D., Larsen, K.G., Legay, A., Mikuč ionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for stochastic hybrid systems. Electronic Proceedings in Theoretical Computer Science **92**, 122–136 (aug 2012). https://doi.org/10.4204/eptcs.92.9
9. Donzé, A., Maler, O.: Robust satisfaction of temporal logic over real-valued signals. In: Chatterjee, K., Henzinger, T.A. (eds.) Formal Modeling and Analysis of Timed Systems. pp. 92–106. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
10. Han, T., Katoen, J.: Counterexamples in probabilistic model checking. In: Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 - April 1, 2007, Proceedings. pp. 72–86 (2007)
11. Jégourel, C., Legay, A., Sedwards, S.: Importance splitting for statistical model checking rare properties. In: Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings. pp. 576–591 (2013)
12. Kiviriga, A., Larsen, K.G., Nyman, U.: Randomized reachability analysis in uppaal: Fast error detection in timed systems. In: Lluch Lafuente, A., Mavridou, A. (eds.) Formal Methods for Industrial Critical Systems. pp. 149–166. Springer International Publishing, Cham (2021)
13. Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B.: Importance splitting in uppaal. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation. Adaptation and Learning - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part III. Lecture Notes in Computer Science, vol. 13703, pp. 433–447. Springer (2022). https://doi.org/10.1007/978-3-031-19759-8_26, https://doi.org/10.1007/978-3-031-19759-8_26
14. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems. pp. 152–166. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
15. Nghiem, T., Sankaranarayanan, S., Fainekos, G., Ivancic, F., Gupta, A., Pappas, G.J.: Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In: Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12-15, 2010. pp. 211–220 (2010)
16. Ničković, D., Yamaguchi, T.: Rtamt: Online robustness monitors from stl. In: Hung, D.V., Sokolsky, O. (eds.) Automated Technology for Verification and Analysis. pp. 564–571. Springer International Publishing, Cham (2020)
17. Rubino, G., Tuffin, B.: Rare event simulation using Monte Carlo methods. John Wiley & Sons (2009)
18. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings. pp. 202–215 (2004)
19. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. In: Alur, R., Peled, D.A. (eds.) Computer Aided Verification. pp. 202–215. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
20. Younes, H.L.S.: Verification and Planning for Stochastic Processes with Asynchronous Events. Ph.D. thesis (2004)

21. Younes, H.L.S., Simmons, R.G.: Probabilistic verification of discrete event systems using acceptance sampling. In: Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings. pp. 223–235 (2002)

Appendix A Signal Temporal Logic

An STL formula φ consists of atomic predicates together with Boolean and temporal operators. The temporal operators include *always* ($\Box$), *eventually* ($\Diamond$) and *until* ($\mathcal{U}$) and are restricted to intervals of form $[a, b]$, where $0 \leq a < b$ and $a, b \in \mathbb{R}_{\geq 0}$. Now let $\sim \in \{\leq, <, >, \geq =, \neq\}$ be the set of relational operators. The atomic predicates are defined over a scalar-valued function $f(y(t)) \sim c$ evaluated over an input y at time t and where $n \in \mathbb{N}$. The grammar of STL language is then given as:

$$\varphi := \mathrm{T} \mid f(y(t)) \sim c \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2 \tag{7}$$

where I is a non-empty interval defined over extended reals.

The temporal operators can be defined in terms of Boolean and logic operators such that $\Diamond_{[a,b]\varphi} \equiv \mathrm{T}\mathcal{U}_{[a,b]}\varphi$ and $\Box_{[a,b]}\varphi = \neg\Diamond_{[a,b]}\neg\varphi$. If the interval is omitted, it is assumed to be $[0, \infty)$. Furthermore, the quantitative semantics is defined for STL by [9] with the help of function $\rho(\varphi, y, t)$ which gives *robustness*, i.e. degree of satisfaction of the formula φ by (y, t). The formula is satisfied if the robustness is positive and the other way around. Given a *robustness* function ρ, when $\rho(\varphi, y)$ is positive it indicates that y satisfies φ, written as $y \models \varphi$. In this paper, we restrict our attention to the *bounded* fragment of STL.

Appendix B Randomized Reachability Analysis

Algorithm 1 shows an adapted version of RRA for simulation purposes. The random walks are issued until a state satisfying the property ψ_t is discovered. In the simulation setting, the property ψ_t is a simulation termination condition (e.g. time bound, step bound, etc.) and therefore the algorithm always terminates. In our case ψ_t is an integer constraint that requires the global (observing) clock to have elapsed until a specified value.

Once the simulation is completed, a finite timed word $\omega[n]$, that consists of the starting state and all delays and actions taken along the way, is returned (line 6).

In the random walks (line 11), different heuristics can be used for `SelectOutput` and `SelectDelay` functions. Among a number of heuristics presented by [12], we use the *random enabled transition* (RET) heuristic. RET chooses an *eventually enabled* transition uniformly at random, i.e. a transition that is either currently enabled or can become such after a delay.

More formally, consider $\mathcal{H}$ to be HA. Let $TB : S \times \Sigma \to \mathcal{P}(\mathbb{R}_{\geq 0})$ give the lower and the upper bound of the transition's availability range over the actions

Algorithm 1 Randomized Reachability Simulation

```
procedure RRA Simulate($s_0, \psi_t, maxSteps$)
    $steps \leftarrow 2^4$
    while true do
        $\omega \leftarrow$ RandomWalk($s_0$)
        if $\omega \models \psi_t$ then
            return $\omega = s_0, d_1 a_1 s_1, \ldots, d_n a_n s_n$
        end if
        $steps \leftarrow \mathbf{min}(steps \cdot 2, maxSteps)$
    end while
end procedure

procedure RandomWalk($s, steps$)
    $i \leftarrow 0$
    $\omega \leftarrow s$
    while $s \not\models \psi_t$ and $i < steps$ do
        $a \leftarrow$ SelectOutput($s$)
        $d \leftarrow$ SelectDelay($a$)
        $s \leftarrow s'$ such that $s \xrightarrow{d}\xrightarrow{a} s'$
        $\omega \leftarrow \omega d a s$
        $i \leftarrow i + 1$
    end while
    return $\omega$
end procedure
```

of a given HA. Simply put, TB gives both the smallest and the largest delay after which a certain action can be taken. Formally:

$$TB(s) = \{\mathop{\mathbf{arg\ min}}_{d \in \mathbb{R}_{\geq 0}} s \xrightarrow{d}\xrightarrow{a} s'\} \cup \{\mathop{\mathbf{arg\ max}}_{d \in \mathbb{R}_{\geq 0}} s \xrightarrow{d}\xrightarrow{a} s'\} \tag{8}$$

RRA simulation generates timed words $w \in \Sigma$ of HA $\mathcal{H}$ s.t.:

$$w = s_0 \xrightarrow{d_1}\xrightarrow{a_1} s_1 \xrightarrow{d_2}\xrightarrow{a_2} s_2 \xrightarrow{d_3}\xrightarrow{a_3} \dots \xrightarrow{d_n}\xrightarrow{a_n} s_n \tag{9}$$

where $s_n \models \psi_t$, and a_i is drawn uniformly at random from the set $\{a \mid s_{i-1} \xrightarrow{d}\xrightarrow{a} .d \in \mathbb{R}_{\geq 0}\}$ for all i, and d is also drawn uniformly at random from $TB(s_j, a_{j+1})$ for all j.

Appendix C Importance Splitting Algorithm

The procedure of our IS is shown in Algorithm 2 which is an adapted version of Algorithm 2 from [13] with the *fixed effort* scheme that we explain later. For each level n we perform a fixed number m of simulations. A state is chosen from the set of previous $\texttt{Successors}_{i-1}$(line 6) uniformly at random and a successor is generated randomly according to the stochastic semantics (line 7). The generated successor is recorded in the set of $\texttt{Successors}_i$, but only if it was obtained by following a corresponding action a_i from the timed cylinder.

Algorithm 2 Importance Splitting algorithm

```
 1: procedure IMPORTANCE SPLIT($\pi(s, a_1, a_2, \dots, a_n)$)
 2:     Successors$_0 = \{s\}$
 3:     for $i \leftarrow 1 \dots n$ do
 4:         Successors$_i \leftarrow \emptyset$
 5:         for $j \leftarrow 1 \dots m$ do
 6:             $s \in$ Successors$_{i-1}$                ▷ uniformly at random
 7:             Let $s \xrightarrow{d}\xrightarrow{a} s'$      ▷ w.r.t. to the stochastic semantics
 8:             if $a = a_i$ then
 9:                 Successors$_i \leftarrow$ Successors$_i \cup \{s'\}$
10:             end if
11:         end for
12:         $p_i \leftarrow \frac{|\texttt{Successors}_i|}{m}$
13:     end for
14:     return $\prod_{i=1}^{n} p_i$
15: end procedure
```

Appendix D Simulation sensitivity

We perform an experiment to determine to which degree the variance in the probability estimates produced by IS is affected by the number m of IS simulations performed per level by IS Algorithm 2. We vary the number of simulations per level and the total amount of levels, reporting the results in Table 2. As anticipated, in cases with a small number of 50 simulation per level, a considerable amount of IS attempts (up to 50%) have failed to follow an entire timed cylinder ($\frac{Fails}{Successes}$ column). That is due to the IS algorithm being unable ("unlucky") to get through one of the "difficult" levels with only 50 attempts per level. However, to our surprise we have not been able to see a clear pattern indicating the amount of IS simulations per level to significantly affect the variance of the probability estimates. To confirm this, we performed several additional experiments both with the same parameters and with different ones, but the observation remained the same. In the light of this and the fact that the execution time taken is roughly proportional to the number of IS simulations per level, in further experiments we fix the number of IS simulation per level to 100.

Table 2: IS fixed effort scheme sensitivity to the estimated probability variance w.r.t. number of simulation m per level. Each row represents 100 IS runs and a single exact probability calculation. Time is given in HH:MM:SS.

IS pr. stdev	IS pr. mean	Exact pr.	$\frac{\text{IS pr. mean}}{\text{Exact pr.}}$	Simulations per level	# levels (# trans.)	$\frac{\text{Fails}}{\text{Successes}}$	Total time
4.4554e-16	1.9357e-16	1.2628e-16	+53.3%	50	42	21/79	00:03:41
2.6050e-16	1.4082e-16	1.2628e-16	+11.5%	100	42	2/98	00:08:00
7.6806e-17	9.9405e-17	1.2628e-16	-21.3%	200	42	0/100	00:16:21
6.6849e-17	1.2627e-16	1.2628e-16	+0.0%	500	42	0/100	00:41:16
5.3405e-17	1.2639e-16	1.2628e-16	+0.1%	1000	42	0/100	01:23:37
2.1045e-26	8.3141e-27	4.1159e-27	102.0%	50	74	46/54	00:05:04
9.6553e-27	4.2122e-27	4.1159e-27	2.3%	100	74	4/96	00:15:17
3.3171e-27	3.2673e-27	4.1159e-27	-20.6%	200	74	0/100	00:31:18
3.4904e-27	4.1488e-27	4.1159e-27	0.8%	500	74	0/100	01:18:32
2.1917e-27	4.1945e-27	4.1159e-27	1.9%	1000	74	0/100	02:39:26
2.4205e-41	5.4142e-42	4.4194e-42	22.5%	50	118	25/75	00:09:59
4.8414e-42	2.7542e-42	4.4194e-42	-37.7%	100	118	1/99	00:24:42
3.7592e-42	2.9829e-42	4.4194e-42	-32.5%	200	118	0/100	00:49:17
4.0601e-42	4.1128e-42	4.4194e-42	-6.9%	500	118	0/100	02:07:44
2.9377e-42	5.0765e-42	4.4194e-42	14.9%	1000	118	0/100	04:16:32