Statistical Model Checking for Stochastic Hybrid Systems

Alexandre David, Marius Mikučionis, Kim Guldstrand Larsen, Danny Bogsten Poulsen, Axel Legay, Sean Sedwards, Dehui Du

Statistical Model Checking for Stochastic Hybrid Systems *

Alexandre David Kim G. Larsen Marius Mikučionis Danny Bøgsted Poulsen
Department of Computer Science
Aalborg University, Denmark
{adavid,kgl,marius,dannybp}@cs.aau.dk

Axel Legay Sean Sedwards
INRIA Rennes
Bretagne Atlantique, France
axel.legay@inria.fr,sean.sedwards@gmail.com

Dehui Du
Shanghai Key Laboratory of Trustworthy Computing
East China Normal University,
Shanghai 20062, China
dehuidu@cs.aau.dk

This paper presents novel extensions and applications of the UPPAAL-SMC model checker. The extensions allow for statistical model checking of stochastic hybrid systems. We show how our race-based stochastic semantics extends to networks of hybrid systems, and indicate the integration technique applied for implementing this semantics in the UPPAAL-SMC simulation engine. We report on two applications of the resulting tool-set coming from systems biology and energy aware buildings.

1 Introduction

Statistical Model Checking (SMC) [29, 27, 32, 34, 25] is an approach that has recently been proposed as new validation technique for large-scale, complex systems. The core idea of SMC is to conduct some simulations of the system, monitor them, and then use statistical methods (including sequential hypothesis testing or Monte Carlo simulation) in order to decide with some degree of confidence whether the system satisfies the property or not. By nature, SMC is a compromise between testing and classical formal method techniques. Simulation-based methods are known to be far less memory and time intensive than exhaustive ones, and are some times the only option. SMC has been implemented in a series of tools that have defeated well-known tools such as PRISM on several case studies. Unlike more "academic" exhaustive (and intractable) techniques, SMC is spreading to various research areas such as systems biology [21, 24] and software engineering [36, 30], in particular for industrial applications [4, 6, 5].

There are several reasons for this success. Firstly, SMC is simple to understand, use and (in principle) to implement. Secondly, no additional modelling or specification effort is needed, provided that the mode-ling formalism used can be given a natural stochastic semantics serving as the basis for interpretation of the specification formalism and as a basis for generating simulation-runs. Thirdly, SMC allows to analyse properties [11, 4] that cannot be expressed in classical temporal logic, including properties for which classical model checking is undecidable.

In a series of recent works [16, 15], we have investigated the problem of Statistical Model Checking for networks of Priced Timed Automata (PTAs). PTAs are timed automata, whose clocks can evolve with different rates, while[1] being used with no restrictions in guards and invariants. In [15], we have proposed a natural stochastic semantics for such automata, which allows to perform statistical model

*Work partially supported by the VKR Centre of Excellence MT-LAB, the Sino-Danish Basic Research Center IDEA4CPS, and by the CREATIVE project ESTASE.

[1] in contrast to the usual restriction of priced timed automata [7, 1]

Ezio Bartocci and Luca Bortolussi (Eds.): HSB 2012
EPTCS 92, 2012, pp. 122–136, doi:10.4204/EPTCS.92.9

checking. Our work has later been implemented in UPPAAL-SMC, that is a stochastic and statistical model checking extension of UPPAAL. UPPAAL-SMC relies on a series of extensions of the statistical model checking approach generalised to handle real-time systems and estimate undecidable problems. UPPAAL-SMC comes together with a friendly user interface that allows a user to specify complex problems in an efficient manner as well as to get feedback in the form of probability distributions and compare probabilities to analyse performance aspects of systems. The UPPAAL-SMC model checking has been applied to a wide range of examples from networking and Nash equilibrium[9] through systems biology [14], real-time scheduling [13], and energy aware systems [12].

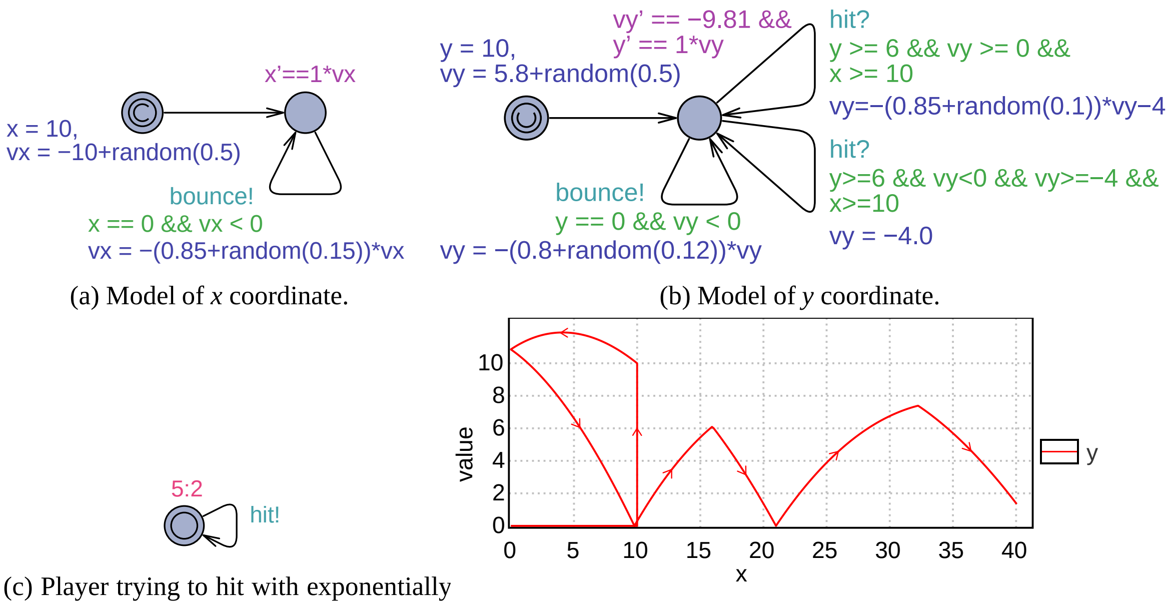
For PTAs clocks evolve with fixed rates depending only on the discrete state of the model, i.e. locations and discrete variables. In this paper, we present and implement an extension of the modelling formalism of UPPAAL-SMC, where clock rates may depend not only on values of discrete variables but also on the value of other clocks, effectively amounting to ordinary differential equations (ODEs). As a first contribution of this paper, we present an extension of our race-based stochastic semantics to networks of stochastic hybrid automata (SHA). One major difficulty is the implementation of this extensions in our engine for generating random runs, in a manner which is correct with respect to the stochastic semantics. In fact UPPAAL-SMC does not solve the equations exactly but currently supports the Euler integration method. A fixed time step (defined by the user) is used by an internal integrator component added to the system. This integrator races with the other processes during its time step and all rates are considered constant and defined by the equations in the model. As a second contribution of the paper we describe this integration in UPPAAL-SMC. It is worth mentioning that while other SMC-based approaches exist for SHAs [36, 35, 2], none of them do consider ODE-based modelisation for clock rates.

To demonstrate the applicability of this new extension, we apply advanced SMC techniques to two challenging applications coming from systems biology and energy aware buildings. For the case of systems biology, we show how the combination of ODEs and SMC allows us to reason on biological oscillations – a problem that is beyond the scope of most existing formal verification techniques. We model a genetic circadian oscillator, which is used to distil the essence of several real circadian oscillators. For the case of energy aware buildings, we refer to a recently developed framework including components for layout of buildings, availability of heaters, climate and user behaviours allowing to evaluate different strategies for distributing heaters among rooms in terms of the resulting comfort and energy consumption. To indicate central parts of this framework and the clear advantages of modelling the evaluation of room temperatures with ODEs, we illustrate in this paper the framework with a small instance comprising two rooms with a single shared heater.

Structure of the Paper. The remainder of the paper is structured as follows. In the next Section 2 we preview the expressive power of the hybrid extensions of UPPAAL-SMC using an extension of the well-known bouncing ball. Sections 3 and 4 details the semantics of the extended formalism of networks of stochastic hybrid automata. Applications to Energy Aware Buildings and a Biological Oscillator are given in Section 5 and Section 6, respectively. Finally, Section 7 concludes the paper and suggests directions for future research.

2 Throwing, Bouncing and Hitting Ball

To give an early illustration of the expressive power of the extended modelling formalism of UPPAAL-SMC, we consider a variant of the well-known bouncing ball. In our version[2] the ball is initially thrown



(a) Model of x coordinate.

(b) Model of y coordinate.

(c) Player trying to hit with exponentially distributed delay.

(d) Trajectory as (x,y) plot.

Figure 1: Models and a trajectory of a thrown/bouncing ball hit by a player.

against a wall, bounces against it, and then continues its trajectory by falling and bouncing against the floor. In addition, a (inexperienced) player tries to hit the ball randomly according to an exponential distribution. The model is depicted in Fig. 1(a)–(c). The player is modelled as a simple automaton that broadcasts `hit!` with an exponential distribution of rate $5/2$. The `x` coordinate (Fig. 1a) is initialised to 10 with an uncertain derivative `vx` uniformly between $[-10,-9.5]$ (the ball is thrown against the wall), after which the ball moves toward the wall (placed at 0). Here, the automaton outputs `bounce!` on an *urgent* channel, which forces the transition to take place deterministically at `x`$= 0$. After a bounce with a random dampening factor of the velocity `vx` uniformly between $[0.85,1]$, the ball continues to move in the opposite direction. The `y` coordinate (Fig. 1b) is initialised to 10 with an uncertain derivative `vy`. The model shows the effect of gravitation with `vy'`$= -9.81$. The ball bounces with a random dampening factor on the floor (at 0) and when the ball is away from the wall (`x`≥ 10) then it can be hit by the player provided it is high enough (`y`≥ 6). Depending on the current direction of the ball, the ball may bounce or it is pushed. One possible trajectory of the ball is shown in Figure 1d. The plot is obtained by checking the query "`simulate 1 [x<=40]`$\{$`y`$\}$". The vertical line shows the ball moving to its initial position and should be ignored. The ball bounces as expected against the wall, the floor, and the hitting of the player. UPPAAL-SMC is able to simulate this hybrid system that has a second order ODE, a stochastic controller (the player), and a stochastic environment (random dampening factor).

In addition, we may perform statistical model-checking in order to estimate the probability that the ball is still bouncing above a height of 4 after 12 time units with the query:

```
Pr[<=20](<> time>=12 and y>=4)
```

which returns the confidence interval $[0.44,0.55]$ with 95% confidence after having generated 738 runs. We can also test for the hypothesis

[2] Our version is inspired by the recently announced solution by a 16 year old German school boy (Shouryya Ray), to a 350 year open problem by Newton concerned with predicting the trajectory of a ball thrown at a wall.

```
Pr[<=20](<> time>=12 and y>=4) >= 0.45,
```

which gives a more precise lower bound. The hypothesis holds with a region of indifference ± 0.01 and a level of significance of 5% after generating 970 runs.

3 Networks of Hybrid Automata

UPPAAL-SMC now supports the analysis of stochastic hybrid automata (SHA) that are timed automata whose clock rates can be changed to be constants or expressions depending on other clocks, effectively defining ODEs. This generalizes the model used in our previous work [16, 15] where only linear priced automata were handled. Our new release UPPAAL-SMC 4.1.10[3] supports fully hybrid automata with ODEs and a few built-in complex functions (such as `sin`, `cos`, `log`, `exp` and `sqrt`)!

Hybrid Automata Intuitively, a hybrid automaton $\mathscr{H}$ is a finite-state automaton extended with continuous variables that evolve according to dynamics characterizing each discrete state (called a *location*). Let X be a finite set of continuous variables. A *variable valuation* over X is a mapping $v : X \to \mathbb{R}$, where $\mathbb{R}$ is the set of reals. We write $\mathbb{R}^X$ for the set of valuations over X. Valuations over X evolve over time according to *delay functions* $F : \mathbb{R}_{\geq 0} \times \mathbb{R}^X \to \mathbb{R}^X$, 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) $\mathscr{H}$ is a tuple $\mathscr{H} = (L, \ell_0, X, E, F, I)$, where: (i) L is a finite set of locations, (ii) $\ell_0 \in L$ is an initial location, (iii) X is a finite set of continuous variables, (iv) $\Sigma = \Sigma_i \uplus \Sigma_o$ is a finite set of actions partitioned into inputs (Σ_i) and outputs (Σ_o), (v) E is a finite set of edges of the form $(\ell, g, a, \varphi, \ell')$, where ℓ and ℓ' are locations, g is a predicate on $\mathbb{R}^X$, action label $a \in \Sigma$ and φ is a binary relation on $\mathbb{R}^X$, (vi) for each location $\ell \in L$ $F(\ell)$ is a* delay function*, and (vii) I assigns an invariant predicate $I(\ell)$ to any location ℓ.*

The semantics of a HA $\mathscr{H}$ is a timed labeled transition system, whose states are pairs $(\ell, v) \in L \times \mathbb{R}^X$ with $v \models I(\ell)$, and whose transitions are either delay transitions $(\ell, v) \xrightarrow{d} (\ell, v')$ with $d \in \mathbb{R}_{\geq 0}$ and $v' = F(d, v)$, or discrete transitions $(\ell, v) \xrightarrow{a} (\ell', v')$ if there is an edge $(\ell, g, a, \varphi, \ell')$ such that $v \models g$ and $\varphi(v, v')$. We write $(\ell, v) \rightsquigarrow (\ell', v')$ if there is a finite sequence of delay and discrete transitions from (ℓ, v) to (ℓ', v').

In the above definition, we have deliberately left open the concrete syntax for the delay function F as well as the invariant I. In UPPAAL-SMC the delay update for a simple clock x – used in (priced) timed automata – is given by an implicit rate $x' = 1$ or an explicit rate $x' = e$ appearing in the invariant of ℓ, where e is an expression only depending on the discrete part of the current state. More generally, the effect of the delay function F may be specified by a set of ODEs needing to be solved. It is important to note that in specifying the delay function F and the invariant I, the full syntax of UPPAAL expressions – including user-defined functions – is at the disposal.

Example 1. *Reconsider the extended bouncing ball example from Section 2. Here the automaton for the $\mathtt{x}$ coordinate may be initialized to the state* $(\mathtt{x} = 10, \mathtt{vx} = -9.8)$*, after which the following transition sequence may occur:*

$$(\mathtt{x} = 10, \mathtt{vx} = -9.8) \xrightarrow{10 \div 9.8} (\mathtt{x} = 0, \mathtt{vx} = -9.8) \xrightarrow{\mathtt{bounce!}} (\mathtt{x} = 0, \mathtt{vx} = 9.31)$$

[3] `www.uppaal.org`.

where in the `bounce!`*-transition the dampening factor has non-deterministically been chosen from the interval* $[0.85, 1.00]$ *as* 0.95*. The automaton for the* `y` *coordinate may be initialized to the state* $(\mathtt{y} = 10, \mathtt{vy} = 5.9)$ *after which the delay function will effectively be given by delay transitions:*

$$(\mathtt{y} = 10, \mathtt{vy} = 5.9) \xrightarrow{d} (\mathtt{y} = -9.81/2d^2 + 5.9d + 10, \mathtt{vy} = -9.81d + 5.9)$$

The ball `bounce!`*'es on the floor, i.e.* $\mathtt{y} = 0$ *or when (approximately)* $d = 2.15$*, and will afterwards - having non-deterministically chosen the dampening factor* 0.9 *- continue from the state* $(\mathtt{y} = 0, \mathtt{vy} = 13.67)$.

Networks of Hybrid Automata Following the compositional specification theory for timed systems in [17], we shall assume that NHAs are *input-enabled* in the sense, that for all states (ℓ, ν) and input actions $\iota \in \Sigma_i$, for all HAs j, there is an edge $(\ell^j, g, \iota, \varphi, \ell^{j'})$ such that $\nu \models g$ and $\varphi(\nu, \nu')$ for some valuation ν'. Also, we shall assume that time always diverges, and that different automata synchronize on matching inputs and outputs as a standard broadcast synchronization [20].

Whenever $\mathscr{A}^j = (L^j, X^j, \Sigma^j, E^j, F^j, I^j)$ $(j = 1 \ldots n)$ are NHA, they are *composable* into a *closed network* iff their variable-sets are disjoint ($X^j \cap X^k = \emptyset$ when $j \neq k$), they have the same action set ($\Sigma = \Sigma^j = \Sigma^k$ for all j, k), and their output action-sets provide a partition of Σ ($\Sigma_o^j \cap \Sigma_o^k = \emptyset$ for $j \neq k$, and $\Sigma = \cup_j \Sigma_o^j$). For $a \in \Sigma$ we denote by $c(a)$ the unique j with $a \in \Sigma^j$. If $\nu \in \mathbb{R}^X$ with $X = \cup_j X^j$, we denote by $\nu \downarrow_{X^j}$ the projection of ν to X^j.

Definition 2. *Let* $\mathscr{A}^j = (L^j, X^j, \Sigma, E^j, F^j, I^j)$ *(with* $j = 1 \ldots n$*) be composable NHAs. Their* composition $(\mathscr{A}_1 | \ldots | \mathscr{A}_n)$ *is the HA* $\mathscr{A} = (L, X, \Sigma, E, F, I)$ *where (i)* $L = \times_j L^j$*, (ii)* $X = \cup_j X^j$*, (iii)* $F(\ell)(d, \nu)(x) = F^j(\ell^j)(d, \nu \downarrow_{X^j})(x)$ *when* $x \in X^j$*, (iv)* $I(\ell) = \cap_j I(\ell^j)$*, and (v)* $(\ell, \cap_j g_j, a, \cup_j \varphi_j, \ell') \in E$ *whenever* $(\ell_j, g_j, a, \varphi_j, \ell'_j) \in E^j$ *for* $j = 1 \ldots n$*.*

4 Stochastic Semantics for Networks of Hybrid Automata

Reconsidering again our extended version of the bouncing ball from Section 2, it is clear that there is a constant race between the ball `bounce!`ing on the floor and the player `hit!`ing the ball. Whereas the time of bouncing is deterministic – given by the ODE obtained from the (stochastic) effect of the previous `bounce!` or `hit!` – the time of hitting is stochastic according to an exponential distribution with rate $5/2$. However based on this, a measure on sets of runs of the systems is induced, according to which quantitative properties such as *"the probability that the ball with have a height greater than 4 after 12 time-units"* become well-defined.

Our early works [15] – though aimed at stochastic semantics of *priced timed automata* – is sufficiently general that it also provides the basis for a natural stochastic semantics for networks of HAs, where components associate probability distributions to both the time-delays spent in a given state as well as to the transition between states.

Let $\mathscr{A}^j = (L^j, X^j, \Sigma, E^j, F^j, I^j)$ $(j = 1 \ldots n)$ be a collection of composable HAs. Under the assumption of input-enabledness, disjointedness of clock sets and output actions, states of the composite NHA $\mathscr{A} = (\mathscr{A}_1 | \ldots | \mathscr{A}_n)$ may be seen as tuples $\mathbf{s} = (s_1, \ldots, s_n)$ where s_j is a state of $\mathscr{A}^j$, i.e. of the form (ℓ, ν) where $\ell \in L^j$ and $\nu \in \mathbb{R}^{X^j}$. Our probabilistic semantics is based on the principle of independence between components. Repeatedly each component decides on its own – based on a given delay density function and output probability function – how much to delay before outputting and what output to broadcast at that moment. Obviously, in such a race between components the outcome will be determined by the component that has chosen to output after the minimum delay: the output is broadcast and all other components may consequently change state.

Stochastic Semantics of HA Components The stochastic semantics of HAs refine the non-deterministic choices that may exist with respect to delay, output and next state. We consider the component $\mathscr{A}^j$ and let St^j denote the corresponding set of states. For each state $s = (\ell, v)$ of $\mathscr{A}^j$, we shall assume that there exist probability distributions for delays, output as well as next-state:

- the *delay density function*, μ_s over delays in $\mathbb{R}_{\geq 0}$, provides stochastic information for when the component will perform an output, thus $\int \mu_s(t)dt = 1$;
- the *output probability function* γ_s assigns probabilities for resolving what output $o \in \Sigma_o^j$ to generate, i.e. $\sum_o \gamma_s(o) = 1$;
- the *next-state density function* η_s^a provide stochastic information on the next state $s' = (\ell', v') \in \mathbb{R}^X$ given an action a, i.e. $\int_{s'} \eta_s^a(s') = 1$.

For outputs happening deterministically at an exact time point d (or deterministic next states s'), μ_s (η_s^a) becomes a Dirac delta function δ_d ($\delta_{s'}$)[4].

In UPPAAL-SMC uniform distributions are applied for states where delay is bounded, and exponential distributions (with location-specified rates) are applied for the cases, where a component can remain indefinitely in a location. Also, UPPAAL-SMC provides syntax for assigning discrete probabilities to different outputs as well as specifying stochastic distributions on next-states (using the function `random[b]` denoting a uniform distribution on $[0, b]$).

Stochastic Semantics of Networks of HA For the stochastic semantics of closed networks of HA consider $\mathscr{A} = (\mathscr{A}_1 | \dots | \mathscr{A}_n)$ with a state space $\mathsf{St} = \mathsf{St}_1 \times \cdots \times \mathsf{St}_n$. For $\mathbf{s} = (s_1, \dots, s_n) \in \mathsf{St}$ and $a_1 a_2 \dots a_k \in \Sigma^*$ we denote by $\pi(\mathbf{s}, a_1 a_2 \dots a_k)$ the set of all maximal runs from $\mathbf{s}$ with a prefix $t_1 a_1 t_2 a_2 \dots t_k a_k$ for some $t_1, \dots, t_n \in \mathbb{R}_{\geq 0}$, that is runs where the i'th action a_i has been outputted by the component $A_{c(a_i)}$. Providing the basic elements of a Sigma-algebra, we now inductively define the following measure for such sets of runs:

$$\mathbb{P}_{\mathscr{A}}\big(\pi(\mathbf{s}, a_1 \dots a_n)\big) = \\ \int_{t \geq 0} \mu_{s_c}(t) \cdot \big(\prod_{j \neq c} \int_{\tau > t} \mu_{s_j}(\tau) d\tau\big) \cdot \gamma_{s_c{}^t}(a_1) \cdot \int_{\mathbf{s}'} \big(\prod_j \eta^{a_1}_{s^t_j}(s'_j) \cdot \mathbb{P}_{\mathscr{A}}\big(\pi(\mathbf{s}', a_2 \dots a_n)\big) d\mathbf{s}'\big)\, dt$$

This definition requires a few words of explanation: at the outermost level we integrate over all possible initial delays t. For a given delay t, the outputting component $c = c(a_1)$ will choose to make the broadcast at time t with the stated density. Independently, the other components will choose to a delay amount, which – in order for c to be the winner – must be larger than t; hence the product of the probabilities that they each make such a choice. Having decided for making the broadcast at time t, the probability of actually outputting a_1 is included. Finally, integrating over all global states $\mathbf{s}'$ that may result from all components having delayed t time-units and changed state stochastically with respect to the broadcasted action a_1, the probability of runs according to the remaining actions $a_2 \dots a_n$ is taken into account.

[4] which should formally be treated as the limit of a sequence of delay density functions with decreasing, non-zero support around d.

Generation of Random Runs The central component of the SMC engine of UPPAAL-SMC is the efficient generation of random runs according to the stochastic semantics proposed in the preceding Section. UPPAAL-SMC has to integrate the ODEs given in the model while still respecting its stochastic semantics. The tool does not solve the equations exactly and currently supports the Euler integration method. A fixed time step δt (defined by the user) is used by an internal integrator component added to the system. This integrator races with the other processes during its time step and all rates are considered constant and defined by the equations in the model. At the end of every step or if another process wins the race, all the rates are re-evaluated and the resulting values of the clocks are computed according to $x_{new} = x_{old} + \delta t * x'_{old}$. We plan to implement more robust methods, such as Runge-Kutta's, that give different evaluations of x_{new}. In this case, x'_{old} is then derived from the method and not directly from the equation in the model and kept constant during δt. The stochastic semantics and the races between the components are still respected.

The integrator component always picks a delay equal to the given time step δt. The other components pick delays according to their distributions. The winning component delays to the point in time it wanted and tries to take a transition at that point. If it appears that no guard is satisfied, no transition is taken and all components race again. In practice, every component will use the current rates of the clocks to evaluate a lower bound on the delay from which anything can happen to skip delaying for nothing.

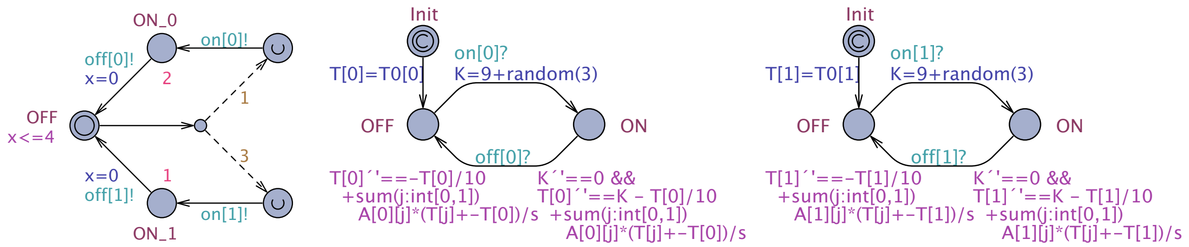
Statistical Model Checking We use SMC [28, 32, 34, 29] to estimate and test on the probability that a random run of a network of SHAs will satisfy a given property. Given a model $\mathscr{H}$ and a trace property φ (e.g. expressed in LTL [31] or MTL [26]), SMC refers to a series of simulation-based techniques that can be used to answer two questions: (1) *Qualitative:* is the probability that a random run of $\mathscr{H}$ will satisfy φ greater or equal to a certain threshold θ (or greater or equal to the probability to satisfy another property φ')? and (2) *Quantitative:* what is the probability that a random run of $\mathscr{H}$ will satisfy φ? In both cases, the answer will be correct up to a user-specified level of confidence, providing a upper bound on the probability that the conclusion made by the algorithm will be wrong. For the quantitative approach, which we will use intensively in this paper, the method computes a confidence interval that is an interval of probabilities that contains the true probability to satisfy the property. Here the confidence level provides the probability that the computed confidence interval indeed contains the unknown probability.

Our UPPAAL-SMC tool-set implements a wide range of SMC algorithms for networks of SHAs not only for reachability and safety properties, but also for general weighted MTL properties [10, 8]. In addition, the tool offers several features to visualize and reason on the results.

5 Energy Aware Buildings

UPPAAL-SMC has recently [12] been applied to an evaluation framework for energy aware buildings, where the control of heating is optimized with respect to environmental and user profiles. To indicate central parts of this framework and the clear benefit of modeling with ODEs, we illustrate the framework on a simplified instance and recall results from our case study [12] based on a benchmark for hybrid systems verification [18].

A Simple 2-Room Example To illustrate the various aspects of the (extended) modeling formalism supported by UPPAAL-SMC, we consider the case of two independent rooms that can be heated by a single heater shared by the two rooms, i.e., at most one room can be heated at a time. Fig.2(a) shows



(a) stochastic heater. (b) room 0. (c) room 1.

Figure 2: A simple two room example with an autonomous heater.

the automaton for the heater. It turns itself on with a uniform distribution over time in-between $[0,4]$ time units. With probability $1/4$ room 0 is chosen and with probability $3/4$ room 1. The heater stays on for some time given by an exponential distribution (rate 2 for room 0, rate 1 for the room 1). In summary, one may say that the controller is more eager to initiate the heating of room 1 than room 0, as well as less eager to stop heating room 1. The rooms are similar and are modeled by the same template instantiated twice as shown in Fig. 2(b-c). The room is initialized to its initial temperature and then depending on whether the heater is turned on or not, the evolution of the temperature is given by $T_i' = -T_i/10 + \sum_{j=0,1} A_{i,j}(T_j - T_i)$ or $T_i' = K - T_i/10 + \sum_{j=0,1} A_{i,j}(T_j - T_i)$ where $i, j = 0, 1$ are room identifiers. The sum expression corresponds to an energy flow between rooms and matrix A encodes the energy transfer coefficient between adjacent rooms. Furthermore, when the heater is turned on, its heating is not exact and is picked with a uniform distribution of $K \in [9,12]$, realized by the update `K=9+random(3)`.

This example illustrates the support for NSHA in UPPAAL-SMC with extended arithmetics on clocks and generalized clock rates.

Extended Input Language UPPAAL-SMC takes as input NSHA as described above. Additionally, there is support for other features of the UPPAAL model checker's input language such as integer variables, data structures and user-defined functions, which greatly ease modeling. UPPAAL-SMC allows the user to specify arbitrary rates for the clocks, which includes a mix of integer and clock expressions on any location. In addition, the automata support branching edges where weights can be added to give a distribution on discrete transitions. It is important to note that rates and weights may be general expressions that depend on the states and not just simple constants.

Checking Queries The fundamental principle in UPPAAL-SMC is to generate runs and evaluate some expression on the states along the obtained run. Runs are always *bounded*, either by time, by a number of steps, or more generally by cost (when using a clock explicitly). The engine has a built-in heuristic detection of Zeno behaviours to abort the generation of such runs. Examples of the syntax for the different types of bounds are `[<=100]` for 100 time units since the beginning of the run, `[#<=50]` for 50 discrete transitions taken from the initial state, and `[x<=200]` until the clock `x` reaches 200[5].

UPPAAL-SMC supports simulations with monitoring custom expressions, probability evaluation, hypothesis testing, and probability comparison. We can simulate and plot the temperatures with the query

```
simulate 1 [<=600]{T[0],T[1]}
```

[5] It is up to the modeler to ensure that the clock eventually reaches the bound.

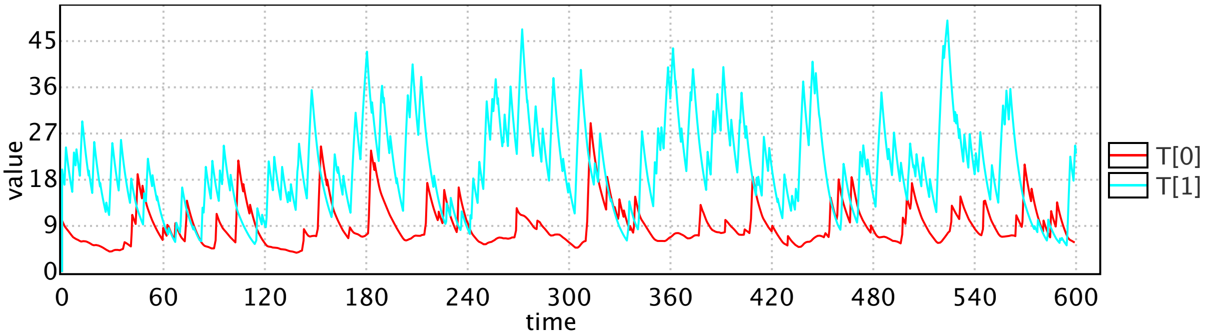


Figure 3: Evolution of the temperatures of the two rooms.

The query request the checker to provide one simulate run over 600 time units and plot the temperatures of `Room(0)` and `Room(1)`. The heater in this example is purely stochastic and is not intended to enforce any particular property. Yet, the simulation obtained from this query in Fig. 3 shows that the heater is able to maintain the temperatures within (mostly) distinct intervals.

We can evaluate on a shorter time scale the probability for the temperature of `Room(0)` to stay below 30 and the temperature of `Room(1)` to stay above 5 with the queries

```
Pr[<=100]([] Room(0).Init || T[0] <= 20)
Pr[<=100]([] Room(1).Init || T[1] >= 7)
```

The results are respectively in $[0.45, 0.55]$ and $[0.65, 0.75]$. The precision and confidence of these so-called confidence intervals are user-defined and influence the number of runs needed to compute the probability. In this example, for having the precision to be ± 0.05 with a confidence of 95%, we need 738 runs. In fact if we are only interested in knowing if the second probability is above a threshold it may be more efficient to test the hypothesis

```
Pr[<=100]([] Room(1).Init || T[1] >= 7) >= 0.69
```

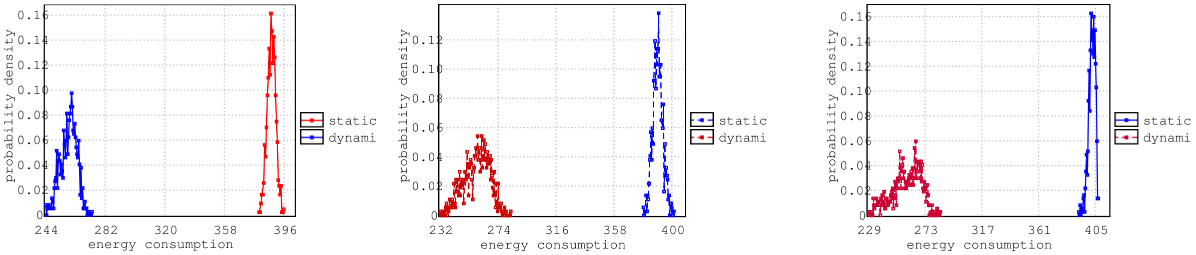
which is accepted in our case with 902 runs for a level of significance of 95%. To obtain an answer at comparable level of precision with probability evaluation, we would need to use a precision of ± 0.005, which would require 73778 runs instead.

The tool can also compare probabilities without needing to compute them individually. We can test the hypothesis that the heater is better at keeping the temperature of `Room(1)` above 8 than keeping the temperature of `Room(0)` below 20:

```
Pr[<=100]([] Room(1).Init || T[1] >= 7) >=
Pr[<=100]([] Room(0).Init || T[0] <= 20)
```

which is accepted in this case with 95% level of significance with just 258 runs.

Results In [12] we have estimated the comfort time (duration while being in comfortable temperature range) and energy consumption for various weather conditions, user profiles and central controller strategies. Fig. 4 shows six energy consumption estimates in different configurations (a building with 5 rooms and 3 heaters). The energy comparison shows that the dynamic user profile can save more than 33% of energy regardless of the chosen central controller strategy.



(a) Strategy 1.

(b) Strategy 2.

(c) Strategy 3.

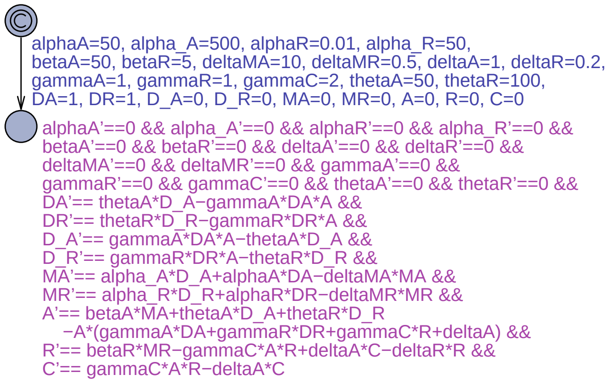
Figure 4: Energy consumption estimates for static and dynamic user profiles.

6 Biological Oscillator

One of the key oscillatory behaviours in biology is the circadian rhythm that allows an organism to take advantage of periods of day and night to optimise when to maximise activity and recovery. We show how the genetic circadian oscillator of [3, 33] can be modelled and analysed using UPPAAL-SMC. This synthetic model distils the essence of several real circadian oscillators and demonstrates how a reliable system can be constructed in the face of inherent stochasticity. Figure 5a shows a system of differential equations from [33]. The equations are typeset in UPPAAL as invariant expression on a location shown in Fig. 5b, where each quantity $D_A, D_R, D'_A, M_A, M_R, A, R, C$ are modelled as continuous clock variables DA, DR, D_A, MA etc with rates defined by a corresponding differential equation. The preceding expressions about alphaA, alpha_A, alphaR, alpha_R, beta_A and so on are modelling the constants $\alpha_A, \alpha'_A, \alpha_R, \alpha'_R, \beta_A$ and so on. The assignments on the first transition initialise all the variables with initial conditions. UPPAAL-SMC is then used to simulate the model and provide plots of how the variable values evolve over time, which are displayed in Figure 7a.

$$\begin{aligned}
dD_A/dt &= \theta_A D'_A - \gamma_A D_A A \\
dD_R/dt &= \theta_R D'_R - \gamma_R D_R A \\
dD'_A/dt &= \gamma_A D_A A - \theta_A D'_A \\
dD'_R/dt &= \gamma_R D_R A - \theta_R D'_R \\
dM_A/dt &= \alpha'_A D'_A + \alpha_A D_A - \delta_{M_A} M_A \\
dM_R/dt &= \alpha'_R D'_R + \alpha_R D_R - \delta_{M_R} M_R \\
dA/dt &= \beta_A M_A + \theta_A D'_A + \theta_R D'_R \\
&\quad - A(\gamma_A D_A + \gamma_R D_R + \gamma_C R + \delta_A) \\
dR/dt &= \beta_R M_R - \gamma_C AR + \delta_A C - \delta_R R \\
dC/dt &= \gamma_C AR - \delta_A C
\end{aligned}$$

(a) Ordinary differential equations.



```
alphaA=50, alpha_A=500, alphaR=0.01, alpha_R=50,
beta A=50, betaR=5, deltaMA=10, deltaMR=0.5, deltaA=1, deltaR=0.2,
gammaA=1, gammaR=1, gammaC=2, thetaA=50, thetaR=100,
DA=1, DR=1, D_A=0, D_R=0, MA=0, MR=0, A=0, R=0, C=0
```

```
alphaA'==0 && alpha_A'==0 && alphaR'==0 && alpha_R'==0 &&
betaA'==0 && betaR'==0 && deltaA'==0 && deltaR'==0 &&
deltaMA'==0 && deltaMR'==0 && gammaA'==0 &&
gammaR'==0 && gammaC'==0 && thetaA'==0 && thetaR'==0 &&
DA'== thetaA*D_A−gammaA*DA*A &&
DR'== thetaR*D_R−gammaR*DR*A &&
D_A'== gammaA*DA*A−thetaA*D_A &&
D_R'== gammaR*DR*A−thetaR*D_R &&
MA'== alpha_A*D_A+alphaA*DA−deltaMA*MA &&
MR'== alpha_R*D_R+alphaR*DR−deltaMR*MR &&
A'== betaA*MA+thetaA*D_A+thetaR*D_R
   −A*(gammaA*DA+gammaR*DR+gammaC*R+deltaA) &&
R'== betaR*MR−gammaC*A*R+deltaA*C−deltaR*R &&
C'== gammaC*A*R−deltaA*C
```

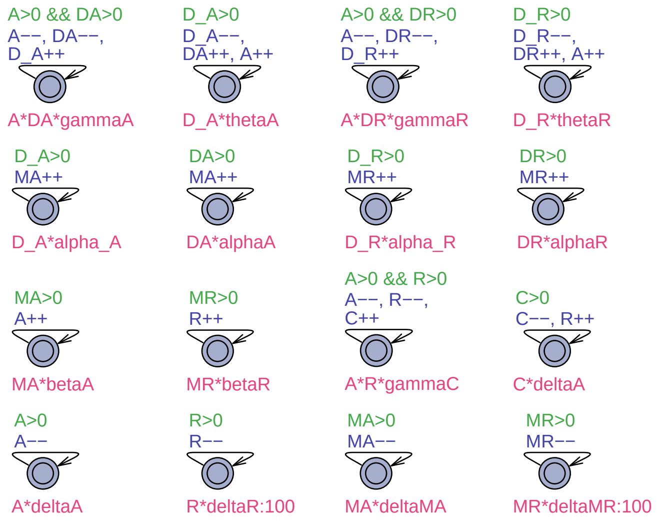
(b) UPPAAL automaton representation.

Figure 5: Dynamics of genetic oscillator.

The ODE system can be interpreted as a system behavior near the thermodynamic limit (infinite population sizes while maintaining the same concentrations). Alternatively this oscilator can be modeled as a system of stochastic chemical reactions, where each molecule is counted as discrete entity moving according to Brownian motion laws. Using a standard translation between deterministic and stochastic semantics of chemically reacting systems (e.g., Gillespie's algorithm [19]) the coefficients in ODE can be interpreted as reaction rates. The reactions are enumerated in Fig. 6a. Each reaction is then modeled

$$\begin{aligned}
\mathrm{A}+\mathrm{D}_A &\xrightarrow{\gamma_A} \mathrm{D}'_A & \mathrm{M}_A &\xrightarrow{\beta_A} \mathrm{M}_A+\mathrm{A}\\
\mathrm{D}'_A &\xrightarrow{\theta_A} \mathrm{D}_A+\mathrm{A} & \mathrm{M}_R &\xrightarrow{\beta_R} \mathrm{M}_R+\mathrm{R}\\
\mathrm{A}+\mathrm{D}_R &\xrightarrow{\alpha_R} \mathrm{D}'_R & \mathrm{A}+\mathrm{R} &\xrightarrow{\gamma_C} \mathrm{C}\\
\mathrm{D}'_R &\xrightarrow{\theta_R} \mathrm{D}_R+\mathrm{A} & \mathrm{C} &\xrightarrow{\delta_A} \mathrm{R}\\
\mathrm{D}'_A &\xrightarrow{\alpha'_A} \mathrm{M}_A+\mathrm{D}'_A & \mathrm{A} &\xrightarrow{\delta_A} \emptyset\\
\mathrm{D}_A &\xrightarrow{\alpha_A} \mathrm{M}_A+\mathrm{D}_A & \mathrm{R} &\xrightarrow{\delta_R} \emptyset\\
\mathrm{D}'_R &\xrightarrow{\alpha'_R} \mathrm{M}_R+\mathrm{D}'_R & \mathrm{M}_A &\xrightarrow{\delta_{M_A}} \emptyset\\
\mathrm{D}_R &\xrightarrow{\alpha_R} \mathrm{M}_R+\mathrm{D}_R & \mathrm{M}_R &\xrightarrow{\delta_{M_R}} \emptyset
\end{aligned}$$

(a) Reactions.



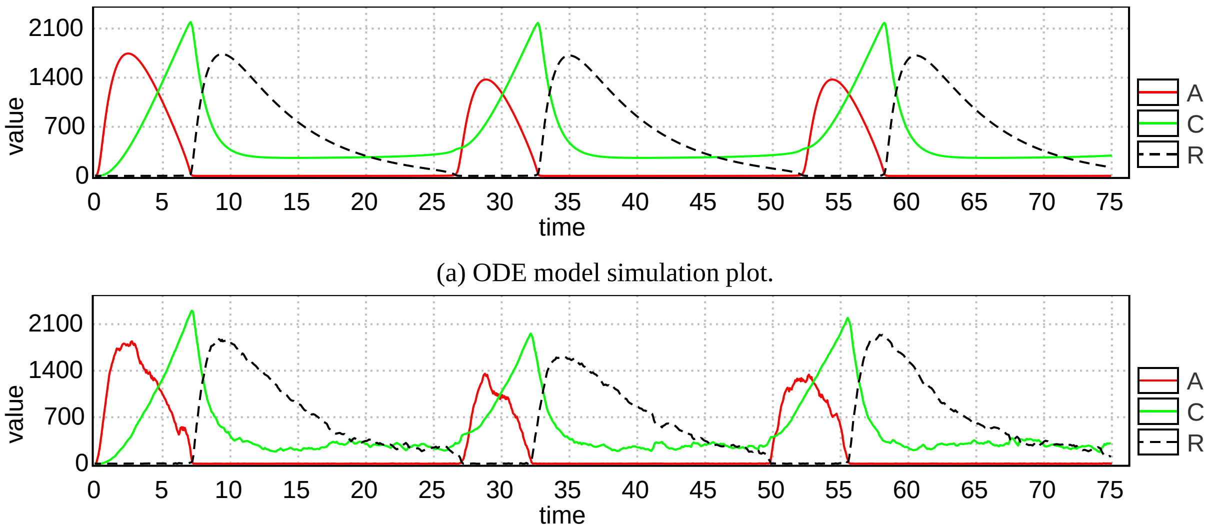
(b) UPPAAL model representation.

Figure 6: Stochastic model of genetic oscilator.

as a separate SHA process shown in Fig. 6b, which can be viewed as an encoding of continuous-time Markov chain process. For example, the first reaction means taking one molecule of each A and D_A substance and producing D'_A with a rate of γ_A, which can be interpreted as a transition requiring positive amount of `A` and `DA` (modeled as integer variables), consuming one molecule each and producing one `D_A` and the reaction rate is `gammaA` and proportional to available quantities of `A` and `DA`. The resulting trajectories of quantities `A`, `C` and `R` are displayed in Fig. 7b, where the patterns seem to resemble the one from ODE model (Fig. 7a), but the inherent stochasticity result in shaky lines (even saw teeth) and unpredictably fluctuating amplitudes and their periods. Interestingly, our formalism is flexible enough to accommodate a hybrid model combining stochastic aspects as in Fig. 7b *and* continuous aspects as in Fig. 7a.

The amplitude of each protein quantity can be measured by the queries `E[<=75; 2000](max:` q`)`, where 75 is the time limit for simulation, 2000 is the number of simulations and q is either `A`, `C`, or `R`. The upper plots of Fig. 8 show the probability density for a range of possible values of amplitude with a vertical line for average value.

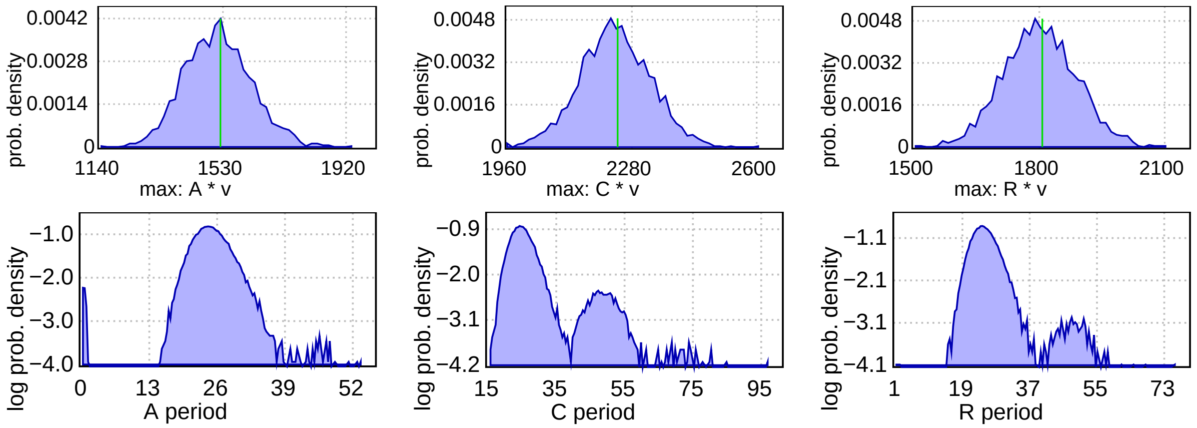
UPPAAL-SMC can also estimate a distance between peaks by using techniques developed for MITL (Metric Interval Temporal Logic – a more expressive property language than a subset of CTL supported by UPPAAL). The idea of the approach is to translate MITL formula into a monitoring automata which start an auxiliary clock `x` with a first peak and stop with a second peak [10]. To detect peaks of `A` when its amount rises above 1100 and drops below 1000 within 5 time units, we use the formula (in the tool syntax): `true U [<=1000] (A>1100 & true U[<=5] A<=1000)`. Then the distance between peaks can be estimated by measuring maximal value of clock `x`. The result is shown as logarithm of probability density plots in a second row of Fig. 8. The plots show that in most cases the measured distance between peaks is about 24.2 hours (slightly more than one day-night cycle). Then there are some smaller bumps with several magnitudes lower probability which can be explained by either a) false positive peak as MITL monitor is confused by a sudden stochastic saw tooth in signal `A`, or b) missing a peak or two, or even three (in `C`) if the peak is not high enough to be registered, hence the next one is registered instead.



(a) ODE model simulation plot.

(b) Stochastic model simulation plot.

Figure 7: UPPAAL-SMC simulations: `simulate 1 [<=75] { A, C, R }`.



(a) `A`. (b) `C`. (c) `R`.

Figure 8: Estimated probability density distributions for amplitude and period.

7 Conclusions

This paper presents an extensions of the UPPAAL-SMC model checker supporting statistical model checking for stochastic hybrid automata, where dynamic of systems can be specified with ODEs. This is a major advance in comparison to existing SMC checkers that can only handle simple derivatives. Currently, the tool applies the Euler integration method. In the future, we plan to implement more robust methods, such as Runge-Kutta's. Another contribution will be to support rare-events as it is the case in [35, 23, 22].

References

[1] Rajeev Alur, Salvatore La Torre & George Pappas (2001): *Optimal Paths in Weighted Timed Automata*. In Maria Di Benedetto & Alberto Sangiovanni-Vincentelli, editors: *Hybrid Systems: Computation and Control*, *Lecture Notes in Computer Science* 2034, Springer Berlin / Heidelberg, pp. 49–62, doi:10.1007/3-540-45351-2_8.

[2] Paolo Ballarini, Hilal Djafri, Marie Duflot, Serge Haddad & Nihal Pekergin (2011): *COSMOS: A Statistical Model Checker for the Hybrid Automata Stochastic Logic*. In: *QEST*, IEEE Computer Society, pp. 143–144. Available at `http://doi.ieeecomputersociety.org/10.1109/QEST.2011.24`.

[3] Naama Barkai & Stanislas Leibler (2000): *Biological rhythms: Circadian clocks limited by noise*. *Nature* 403, pp. 267–268, doi:10.1038/35002258.

[4] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Caillaud, Benoît Delahaye & Axel Legay (2010): *Statistical Abstraction and Model-Checking of Large Heterogeneous Systems*. In: *FMOODS/FORTE*, *Lecture Notes in Computer Science* 6117, Springer, pp. 32–46, doi:10.1007/978-3-642-13464-7_4.

[5] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye & Axel Legay (2012): *Statistical abstraction and model-checking of large heterogeneous systems*. *STTT* 14(1), pp. 53–72, doi:10.1007/s10009-011-0201-2.

[6] Ananda Basu, Saddek Bensalem, Marius Bozga, Benoît Delahaye, Axel Legay & Emmanuel Sifakis (2010): *Verification of an AFDX Infrastructure Using Simulations and Probabilities*. In: *RV*, *Lecture Notes in Computer Science* 6418, Springer, pp. 330–344, doi:10.1007/978-3-642-16612-9_25.

[7] Gerd Behrmann, Ansgar Fehnker, Thomas Hune, Kim Guldstrand Larsen, Paul Pettersson, Judi Romijn & Frits W. Vaandrager (2001): *Minimum-Cost Reachability for Priced Timed Automata*. In: *HSCC*, pp. 147–161, doi:10.1007/3-540-45351-2_15.

[8] P. Bulychev, A. David, K.G. Larsen, A. Legay, G. Li & D.B. Poulsen: *Rewrite-Based Statistical Model Checking of WMTL*. To appear at RV 2012.

[9] Peter E. Bulychev, Alexandre David, Kim G. Larsen, Axel Legay & Marius Mikucionis (2012): *Computing Nash Equilibrium in Wireless Ad Hoc Networks: A Simulation-Based Approach*. In: *IWIGP*, *EPTCS* 78, pp. 1–14, doi:10.4204/EPTCS.78.1.

[10] Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Axel Legay, Guangyuan Li, Danny Bøgsted Poulsen & Amélie Stainer (2012): *Monitor-Based Statistical Model Checking for Weighted Metric Temporal Logic*. In Nikolaj Bjørner & Andrei Voronkov, editors: *LPAR*, *Lecture Notes in Computer Science* 7180, Springer, pp. 168–182, doi:10.1007/978-3-642-28717-6_15.

[11] Edmund Clarke, Alexandre Donzé & Axel Legay (2010): *On simulation-based probabilistic model checking of mixed-analog circuits*. *Formal Methods in System Design* 36, pp. 97–113, doi:10.1007/s10703-009-0076-y.

[12] Alexandre David, DeHui Du, Kim G. Larsen, Marius Mikučionis & Arne Skou (2012): *An Evaluation Framework for Energy Aware Buildings using Statistical Model Checking* Submitted.

[13] Alexandre David, Kim. G. Larsen, Axel Legay & Marius Mikučionis (2012): *Schedulability of Herschel-Planck Revisited Using Statistical Model Checking*. In Bernhard Steffen & Tiziana Margaria, editors: *5th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, Heraclion, Crete. To appear.

[14] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen & Sean Sedwards (2012): *Runtime Verification of Biological Systems*. In Bernhard Steffen & Tiziana Margaria, editors: *5th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, Heraclion, Crete. To appear.

[15] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Jonas Van Vliet & Zheng Wang (2011): *Statistical Model Checking for Networks of Priced Timed Automata*. In: *FORMATS*, LNCS, Springer, pp. 80–96, doi:10.1007/978-3-642-24310-3_7.

[16] Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis & Zheng Wang (2011): *Time for statistical model checking of real-time systems*. In: *Proceedings of the 23rd international conference on Computer aided verification*, LNCS, Springer-Verlag, Berlin, Heidelberg, pp. 349–355. Available at `http://dl.acm.org/citation.cfm?id=2032305.2032332`.

[17] Alexandre David, Kim G. Larsen, Axel Legay, Ulrik Nyman & Andrzej Wasowski (2010): *Timed I/O automata: a complete specification theory for real-time systems*. In: *HSCC*, ACM ACM, pp. 91–100, doi:10.1145/1755952.1755967.

[18] Ansgar Fehnker & Franjo Ivancic (2004): *Benchmarks for Hybrid Systems Verification*. In Rajeev Alur & George J. Pappas, editors: *HSCC*, *Lecture Notes in Computer Science* 2993, Springer, pp. 326–341, doi:10.1007/978-3-540-24743-2_22.

[19] Daniel T. Gillespie (1977): *Exact Stochastic Simulation of Coupled Chemical Reactions*. *Journal of Physical Chemistry* 81, pp. 2340–2361, doi:10.1021/j100540a008.

[20] Rodolfo Gómez (2009): *A Compositional Translation of Timed Automata with Deadlines to UPPAAL Timed Automata*. In Joël Ouaknine & Frits Vaandrager, editors: *Formal Modeling and Analysis of Timed Systems*, *Lecture Notes in Computer Science* 5813, Springer Berlin / Heidelberg, pp. 179–194, doi:10.1007/978-3-642-04368-0_15.

[21] Haijun Gong, Paolo Zuliani, Anvesh Komuravelli, James R. Faeder & Edmund M. Clarke (2010): *Computational Modeling and Verification of Signaling Pathways in Cancer*. In: *ANB*, *Lecture Notes in Computer Science* 6479, Springer, pp. 117–135, doi:10.1007/978-3-642-28067-2_7.

[22] Cyrille Jegourel, Axel Legay & Sean Sedwards (2012): *Cross-entropy optimisation of importance sampling parameters for statistical model checking*. In Madhusudan Parthasarathy & Sanjit A. Seshia, editors: *CAV*, LNCS, Springer, Berkeley, California, USA. To appear.

[23] Cyrille Jegourel, Axel Legay & Sean Sedwards (2012): *A Platform for High Performance Statistical Model Checking – PLASMA*. In Cormac Flanagan & Barbara König, editors: *TACAS*, LNCS, Springer, Tallinn, Estonia, doi:10.1007/978-3-642-28756-5_37.

[24] Sumit Kumar Jha, Edmund M. Clarke, Christopher James Langmead, Axel Legay, André Platzer & Paolo Zuliani (2009): *A Bayesian Approach to Model Checking Biological Systems*. In: *CMSB*, *LNCS* 5688, Springer, pp. 218–234, doi:10.1007/978-3-642-03845-7_15.

[25] Joost-Pieter Katoen, Ivan S. Zapreev, Ernst Moritz Hahn, Holger Hermanns & David N. Jansen (2011): *The ins and outs of the probabilistic model checker MRMC*. *Perform. Eval.* 68(2), pp. 90–104, doi:10.1016/j.peva.2010.04.001.

[26] Ron Koymans (1990): *Specifying Real-Time Properties with Metric Temporal Logic*. *Real-Time Systems* 2(4), pp. 255–299, doi:10.1007/BF01995674.

[27] S. Laplante, R. Lassaigne, F. Magniez, S. Peyronnet & M. de Rougemont (2007): *Probabilistic abstraction for model checking: An approach based on property testing*. *ACM TCS* 8(4), doi:10.1145/1276920.1276922.

[28] Kim Guldstrand Larsen & Arne Skou (1991): *Bisimulation through Probabilistic Testing*. *Inf. Comput.* 94(1), pp. 1–28, doi:10.1016/0890-5401(91)90030-6.

[29] Axel Legay, Benoît Delahaye & Saddek Bensalem (2010): *Statistical Model Checking: An Overview*. In: *RV*, *Lecture Notes in Computer Science* 6418, Springer, pp. 122–135, doi:10.1007/978-3-642-16612-9_11.

[30] João Martins, André Platzer & João Leite (2011): *Statistical Model Checking for Distributed Probabilistic-Control Hybrid Automata with Smart Grid Applications*. In: *ICFEM*, *Lecture Notes in Computer Science* 6991, Springer, pp. 131–146, doi:10.1007/978-3-642-24559-6_11.

[31] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: *FOCS*, IEEE Computer Society, pp. 46–57. Available at `http://doi.ieeecomputersociety.org/10.1109/SFCS.1977.32`.

[32] Koushik Sen, Mahesh Viswanathan & Gul Agha (2004): *Statistical Model Checking of Black-Box Probabilistic Systems*. In: *CAV*, LNCS 3114, Springer, pp. 202–215, doi:10.1007/11513988_26.

[33] José M. G. Vilar, Hao Yuan Kueh, Naama Barkai & Stanislas Leibler (2002): *Mechanisms of noise-resistance in genetic oscillators*. *Proceedings of the National Academy of Sciences* 99(9), pp. 5988–5992, doi:10.1073/pnas.092133899. Available at `http://www.pnas.org/content/99/9/5988.abstract`.

[34] Håkan L. S. Younes & Reid G. Simmons (2002): *Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling*. In: *CAV*, LNCS 2404, Springer, pp. 223–235, doi:10.1007/3-540-45657-0_17.

[35] Paolo Zuliani, Christel Baier & Edmund M. Clarke (2012): *Rare-event verification for stochastic hybrid systems*. In: *HSCC*, ACM, pp. 217–226, doi:10.1145/2185632.2185665.

[36] Paolo Zuliani, André Platzer & Edmund M. Clarke (2010): *Bayesian statistical model checking with application to Simulink/Stateflow verification*. In: *HSCC*, ACM, pp. 243–252, doi:10.1145/1755952.1755987.