Runtime Verification of Biological Systems

Alexandre David, Kim Guldstrand Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Sean Sedwards

Runtime Verification of Biological Systems *

Alexandre David[1], Kim G. Larsen[1], Axel Legay[2], Marius Mikučionis[1], Danny Bøgsted Poulsen[1], Sean Sedwards[2]

[1] Computer Science, Aalborg University, Denmark
[2] INRIA Rennes – Bretagne Atlantique, France

Abstract. Complex computational systems are ubiquitous and their study increasingly important. Given the ease with which it is possible to construct large systems with heterogeneous technology, there is strong motivation to provide automated means to verify their safety, efficiency and reliability. In another context, biological systems are supreme examples of complex systems for which there are no design specifications. In both cases it is usually difficult to reason at the level of the *description* of the systems and much more convenient to investigate properties of their executions.
To demonstrate runtime verification of complex systems we apply statistical model checking techniques to a model of robust biological oscillations taken from the literature. The model demonstrates some of the mechanisms used by biological systems to maintain reliable performance in the face of inherent stochasticity and is therefore instructive. To perform our investigation we use two recently developed SMC platforms: that incorporated in Uppaal and Plasma. Uppaal-smc offers a generic modeling language based on stochastic hybrid automata, while Plasma aims at domain specific support with the facility to accept biological models represented in chemical syntax.

Keywords: runtime verification, synthetic biology, statistical model checking, genetic oscillator, MITL, frequency domain analysis, Uppaal-smc, Plasma.

1 Introduction

It is conceivable to design systems in such a way that makes their analysis easier, but it is most usually the case that they are optimised for other constraints (efficiency, size, cost, etc.) and that they evolve over time, developing highly complex and unforeseen interactions and redundancies. These phenomena are epitomised by biological systems, which have absolutely no inherent need to be understandable or analysable. The discovery that the genetic recipe of life is written with just four characters (nucleotides Adenine, Cytosine, Guanine and Thymine) that are algorithmically transcribed and translated into the machinery

* Work partially supported by VKR Centre of Excellence – MT-LAB and by the CREATIVE project ESTASE.

of the cell (RNA and proteins) has led scientists to believe that biology also works in a computational way. The further realisation that biological molecules and interactions are discrete and stochastic then suggests the idea that biological systems may be analysed using the same tools used to verify, say, a complex aircraft control system.

Using formal methods to investigate natural systems can thus be seen as a way to challenge and refine the process of investigating man-made systems. It is very difficult to reason about systems of this type at the level of their descriptions, however. It is much more convenient to directly analyse their observed behaviour. In the context of computational systems we refer to this approach as runtime verification, while in the case of biological systems this generally takes the form of monitoring the simulation traces of executable computational models.

To demonstrate runtime verification of biological systems we apply advanced statistical model checking (SMC) techniques to a model of robust biological oscillations taken from the literature. SMC works by verifying multiple independent simulation traces of a probabilistic model against a property specified in linear temporal logic. The results are then used in an hypothesis test or to estimate the probability of the property. In adopting this approach, SMC avoids constructing the generally intractable explicit representation of the state space of the system. The price paid is that results are only known within confidence intervals, however these may be made arbitrarily tight by increasing the number of simulation runs. SMC can thus be seen as a specific instance of runtime verification.

The model we have chosen to investigate demonstrates some of the mechanisms used by biological systems to maintain reliable performance in the face of inherent stochasticity. These mechanisms are literally vital and have relevance beyond biology (e.g. amorphous computing). To perform our investigation we use two recently developed SMC platforms: that incorporated in Uppaal and Plasma. Uppaal-smc offers a generic modelling language based on stochastic hybrid automata, while Plasma aims at domain specific support and here accepts biological models represented in chemical syntax. Although our chosen model was conceived to be stochastic, its original description and analysis were in the continuous (ODE) domain. We therefore compare the behaviour of deterministic and stochastic models by performing a frequency domain analysis, taking advantage of Uppaal's recently implemented ability to work with ODE representations. We verify various interesting temporal properties of the model and compare Plasma's direct implementation of bounded LTL with Uppaal's monitor- and rewrite-based implementations of weighted MITL.

2 Beyond Runtime Verification with SMC

Runtime verification (RV) [10,21] refers to a series of techniques whose main objective is to instrument the specification of a system (code, ...) in order to disprove potentially complex properties at the execution level. The main problem

of the runtime verification approach is that it does not permit to assess the overall correctness of the entire system.

Statistical model checking (SMC) [4,26,23] extends runtime verification capabilities by exploiting statistical algorithms in order to get some evidence that a given system satisfies some property. Given a program B and a trace-based property[3] ϕ , Statistical model checking refers to a series of simulation-based techniques that can be used to answer two questions: (1) **Qualitative:** is the probability for B to 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 for B to satisfy ϕ?

We briefly review SMC approaches, referring the reader to [4,26] for more details. The main approaches [26,23] proposed to answer the qualitative question are based on *hypothesis testing*. Let p be the probability of $B \models \phi$, to determine whether $p \geq \theta$, we can test $H : p \geq \theta$ against $K : p < \theta$. A test-based solution does not guarantee a correct result but it is possible to bound the probability of making an error. The *strength* (α, β) of a test is determined by two parameters, α and β, such that the probability of accepting K (respectively, H) when H (respectively, K) holds is less or equal to α (respectively, β). Since it impossible to ensure a low probability for both types of errors simultaneously (see [26] for details), a solution is to use an *indifference region* $[p_1, p_0]$ (with θ in $[p_1, p_0]$) and to test $H_0 : p \geq p_0$ against $H_1 : p \leq p_1$. Several hypothesis testing algorithms exist in the literature. Younes[26] proposed a logarithmic based algorithm that given p_0, p_1, α and β implements the *Sequential Ratio Testing Procedure* (*SPRT*) (see [25] for details). When one has to test $\theta \geq 1$ or $\theta \geq 0$, it is however better to use *Single Sampling Plan* (*SSP*) (see [26,4,23] for details) that is another algorithm whose number of simulations is pre-computed in advance. In general, this number is higher than the one needed by *SPRT*, but is known to be optimal for the above mentioned values. More details about hypothesis testing algorithms and a comparison between *SSP* and *SPRT* can be found in [4].

In [11,17] Peyronnet et al. propose an estimation procedure (*PESTIMATION*) to compute the probability p for B to satisfy ϕ. Given a *precision* δ, Peyronnet's procedure computes a value for p' such that $|p' - p| \leq \delta$ with *confidence* $1 - \alpha$. The procedure is based on the *Chernoff-Hoeffding bound* [13].

3 Model and Properties

3.1 A genetic oscillator

It is well accepted that molecules are discrete entities and that molecular interactions are discrete. It is further accepted that molecules move randomly as a result of collisions with other molecules (thermal noise). From this it can be inferred that chemical reactions are the result of random interactions and can be modelled as stochastic processes. Biological organisms based on chemical reactions are thus supreme examples of complex stochastic systems. The means by

[3] i.e., a property whose semantics is trace-based.

which they overcome low level non-determinism and achieve apparent high level determinism is the subject of much ongoing research and informs such fields as amorphous computing [1].

Oscillation, arising from the execution loops in computer programs, is of great relevance to runtime verification of automated systems. Oscillation also plays a crucial role in biology - life being an essentially cyclic process. 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. In light of this, we have chosen the genetic circadian oscillator of [3,24] as the focus of our analysis. 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. In particular, the model has been shown in [12] to exhibit a kind of regularity referred to as *stochastic coherence*.

Though the authors of the model were interested in its stochastic properties, they nevertheless chose to represent it in the form of a system of ordinary differential equations (ODEs, reproduced in Figure 1). Each of the equations describes the infinitesimal rate of change of a particular molecular species; the functions being the sums of the rates of all reactions involving the species, weighted by the direction (positive or negative, corresponding to creation and consumption) and size of the corresponding change. ODEs are commonly used to represent the dynamics of chemically reacting systems and it is traditional to consider concentrations (numbers of molecules per unit volume). Trajectories of ODEs can closely approximate stochastic dynamics when the system operates near to *the thermodynamic limit* (infinite population sizes). This is rarely the case with biological models of cellular processes, which frequently consider molecular species in very low copy numbers. An obvious example is that within a cell there is often just a single copy of a particular gene (as in the case of the genetic oscillator we describe here). The ODE trajectory is often considered (informally) to be the 'average' of the stochastic traces, implying that the noise is somehow superimposed on top of a deterministic trajectory. In fact, the noise is *an inherent part* of the stochastic trajectory and the ODE describes the behaviour of the *limit* of the stochastic process as populations are taken to infinity while maintaining the same concentrations [7,9]. We demonstrate this using frequency domain analysis in Section 3.4.

Using a standard translation between deterministic and stochastic semantics of chemically reacting systems (see, e.g., [8]), it is possible to transform the ODEs given in Figure 1 into the chemical reaction syntax of Equations (1-16). These can then be visualised as a stochastic Petri net (Figure 2). The model comprises two genes (D_A and D_R) that are *transcribed* (Equations (5-8)) to produce two micro-RNA molecules (M_A and M_R, respectively) that are *translated* (Equations (9,10)) to produce two proteins (A and R, respectively). A acts as a promoter for its own gene (Equation (1)) and for that of R (Equation (3)) by reacting with D_A and D_R to produce their more efficient active forms D'_A and D'_R. A and R *dimerise* (Equation (11)) to form complex protein C that eventually degrades

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

Fig. 1: System of ordinary differential equations describing the genetic oscillator example.

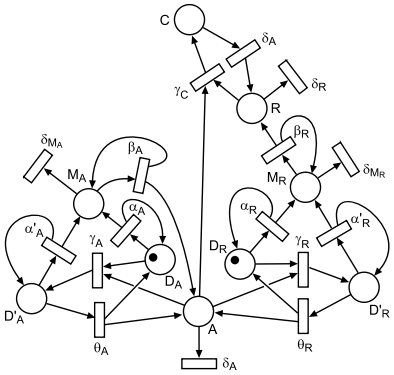


Fig. 2: Petri net representation of the initial state of the genetic oscillator example.

back to D. Oscillation arises from the fact that A is part of a positive feedback loop involved in its own production and promotes the production of R that, in turn, sequesters A (i.e., removes it) via the production of C (Equation (11)). This mechanism and other mechanisms of biological oscillation are discussed in more detail in a recent review of synthetic oscillators [19].

$$\mathrm{A} + \mathrm{D}_A \xrightarrow{\gamma_A} \mathrm{D}'_A \qquad (1)$$

$$\mathrm{D}'_A \xrightarrow{\theta_A} \mathrm{D}_A + \mathrm{A} \qquad (2)$$

$$\mathrm{A} + \mathrm{D}_R \xrightarrow{\alpha_R} \mathrm{D}'_R \qquad (3)$$

$$\mathrm{D}'_R \xrightarrow{\theta_R} \mathrm{D}_R + \mathrm{A} \qquad (4)$$

$$\mathrm{D}'_A \xrightarrow{\alpha'_A} \mathrm{M}_A + \mathrm{D}'_A \qquad (5)$$

$$\mathrm{D}_A \xrightarrow{\alpha_A} \mathrm{M}_A + \mathrm{D}_A \qquad (6)$$

$$\mathrm{D}'_R \xrightarrow{\alpha'_R} \mathrm{M}_R + \mathrm{D}'_R \qquad (7)$$

$$\mathrm{D}_R \xrightarrow{\alpha_R} \mathrm{M}_R + \mathrm{D}_R \qquad (8)$$

$$\mathrm{M}_A \xrightarrow{\beta_A} \mathrm{M}_A + \mathrm{A} \qquad (9)$$

$$\mathrm{M}_R \xrightarrow{\beta_R} \mathrm{M}_R + \mathrm{R} \qquad (10)$$

$$\mathrm{A} + \mathrm{R} \xrightarrow{\gamma_C} \mathrm{C} \qquad (11)$$

$$\mathrm{C} \xrightarrow{\delta_A} \mathrm{R} \qquad (12)$$

$$\mathrm{A} \xrightarrow{\delta_A} \emptyset \qquad (13)$$

$$\mathrm{R} \xrightarrow{\delta_R} \emptyset \qquad (14)$$

$$\mathrm{M}_A \xrightarrow{\delta_{M_A}} \emptyset \qquad (15)$$

$$\mathrm{M}_R \xrightarrow{\delta_{M_R}} \emptyset \qquad (16)$$

Each equation describes a possible productive interaction between types of molecules (molecular *species*). Monomolecular reactions of the form $\mathrm{A} \xrightarrow{k} \cdots$ have the semantics that a molecule of type A will spontaneously decay to some product(s) following a time delay drawn from an exponential distribution with mean k. Bimolecular reactions of the form $\mathrm{A} + \mathrm{B} \xrightarrow{k} \cdots$ have the semantics that if a molecule of type A encounters a molecule of type B they will react to become some product(s) following a time delay drawn from an exponential distribution with mean k. It is usually assumed that the system is 'well stirred' [8,7,9], such that the probability of a molecule occupying a particular position is uniform over the physical space of the system. This rarely represents reality in the case of biological cells, however it is a widely used mathematical expedient that

is common to both deterministic and stochastic treatments. The consequence is that molecules lose their individuality (populations are treated as multisets) and the rate of molecular interactions is simply the product of a rate constant (k in the examples) and a combinatorial factor arising from population sizes [8,7,9]. This is known as *mass action kinetics* [8,7,9]. Referring to molecular species A and B in the example reactions given above, for populations of instantaneous size A and B the overall rates of monomolecular and bimolecular reactions are given by kA and kAB, respectively.

3.2 Properties

The language we use to describe properties is based on the dense timed logic MITL, having the form

$$\phi = \phi \vee \phi \mid \phi \wedge \phi \mid \neg\phi \mid \top \mid \bot \mid \phi \mathrm{U}_{[a;b]} \phi \mid \phi \mathrm{R}_{[a;b]} \phi \mid \mathrm{X}\phi \mid \alpha$$

where $a, b \in \mathbb{N}$, $a < b$ and *alpha* is a proposition of the model. In the case of the genetic oscillator we consider in this paper, the propositions are numeric comparisons of the variables in question.

An expression as $\phi_1 \mathrm{U}[a;b]\phi_2$ means that ϕ_1 should be true until ϕ_2 is true and this should occur between a and b time units. The expression $\phi_1 \mathrm{R}[a;b]\phi_2$ means ϕ_2 should be true until either b time units has passed or both ϕ_1 and ϕ_2 is true and that occurs between a and b time units. $X\phi$ means that ϕ should be true in the next state of the system. The remaining operators have their standard interpretation from propositional calculus. The derived eventuality operator $\Diamond_{[a;b]}\phi$ is introduced as short for $\top \mathrm{U}_{[a;b]}\phi$ and the always operator $\Box_{[a;b]}\phi$ as short for $\bot \mathrm{R}_{[a;b]}\phi$.

3.3 Properties of the Oscillator

The model exhibit an oscillatoric behaviour in which a peak of one protein is followed by the increase of another protein. The increase of one protein also appears to be governed by highly regularity in the sense that one peak level is followed by another peak level in a specific amount of time.

In order to detect peaks we first define the shape of a peak. We say there is a peak if the protein level is above a threshold $thres_L$ and within l time units drops below another threshold $thres_R$.

Using $MITL_{\leq}$ we can express that we at the given time is in a peak of the N variable as

$$\phi_{peakN} \equiv N > thres_L \wedge \Diamond_{[0;l]} N < thres_R.$$

Expressing that there is a periodicity in the peaks of a single variable N within the first 1000 time units can be done using the formula:

$$\Box_{\leq 1000}(\phi_{peakN} \implies \Diamond_{\leq p}\phi_{peakN}),$$

where p is the maximum time between peaks. The same form of expression can of course also be used to express that a peak on the N variable should be followed by a peak on the M variable.

3.4 Frequency domain analysis

Frequency domain analysis provides a rigorous yet intuitive means to quantify the behaviour of stochastic systems from observations of their executions. This methodology is particularly relevant for oscillatory biological systems [14], but is not limited to these and is able to characterise the distance in behaviour between different models, different systems and different parts within the same system. It can also measure the difference between different simulation algorithms or semantics applied to the same system.

Our technique is to generate N simulation traces sampled at constant time intervals, δt, resulting in K sampled points per simulation. From each set of sampled points and for each state variable of interest we calculate a complex frequency spectrum using a 'fast Fourier transform' (FFT) algorithm. From these we generate N corresponding magnitude spectra and then calculate the pointwise average magnitude spectrum. The average magnitude spectrum often gives a visually compact notion of the complex stochastic behaviour of the system and can also be used to quantify a distance between behaviours using standard statistical metrics.

K and δt are chosen according to the temporal characteristics of the phenomenon of interest: $K\delta t$ is the maximum observed simulated time; $(K\delta t)^{-1}$ is the low frequency resolution (the spacing between spectral components) and $(2\delta t)^{-1}$ is the maximum observable frequency. It is generally desirable to increase K and reduce δt, but note that an optimal value of δt is usually significantly greater than the minimum time between successive update events, since these often do not apply to the same variable and the highest part of the spectrum is often uninformative. N is chosen according to the stochasticity of the system in relation to the desired discrimination of the metric; large N being desirable.

4 UPPAAL-SMC

The verification tool Uppaal [18] provide support for modeling and efficient analysis of real-time systems modeled as networks of timed automata [2]. To ease modeling, the tool comes equiped with a user-friendly GUI for defining and simulating models. Also, the modelling formalism extends basic timed automata with discrete variable over basic, structured and user-defined types that may be modified by user-defined functions written in a Uppaal specific C-like imperative language. The specification language of Uppaal is a fragment of TCTL supporting a range of safety, liveness and bounded liveness properties.

Uppaal-smc is a recent branch of Uppaal which support statistical model checking of *stochastic hybrid systems*, based on a natural stochastic semantics. Uppaal-smc extends the basic timed automata formalism of Uppaal by allowing rates of clocks to be defined by general expressions possibly including clocks, thus effectively defining ODEs. An overview of the architecture is given in Figure 3. The GUI of the tool allows the user to draw automata templates in the editor, instantiate and compose these into a system, simulate the system for

easy validation, verify queries, and visualize quantitative answers in the form of plots in the plot composer. The execution engine of UPPAAL-SMC implements the stochast semantics of interacting hybrid automata, and includes a proprietary virtual machine for the execution of imperative code of the model.

The specification formalism of UPPAAL-SMC is that of (weighted) MITL, with respect to which four different statistical model checking components are offered: hypothesis testing, probability estimation, probability comparison and simulation. Here the user may control the accuracy of the analysis by a number of statistical parameters (size of confidence interval, significance level, etc.). UPPAAL-SMC also provides distributed implementations of the hypothesis testing and probability estimation demonstrating linear speed-up [5].

The results generated by the analyses can be processed for visualization in various ways: Gannt charts for individual runs monitoring desired variables, plots of density functions and accumulated distribution functions with respect to given (W)MITL properties. Typically the simulation results in gigabytes of data which are filtered on-the-fly to plot only the relevant points.

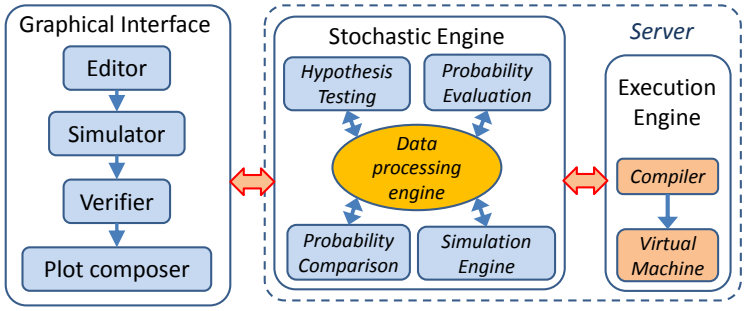


Fig. 3: Architecture of UPPAAL-SMC.

4.1 Modeling and Checking in UPPAAL-SMC

A Bouncing Ball Example. To illustrate the expressive power of the stochastic hybrid automata language supported by UPPAAL-SMC, we consider a simple, yet interesting variant of a bouncing ball. Figure 4(a) gives the principle of a ball bouncing on a floor and being hit by a piston. The hybrid model of the ball is given in Fig. 4(b) where three cases are visible: (i) it can be hit while going up (`v<=0`), (ii) hit while going down (`v<0`), or (iii) it bounces on the floor. The invariant on the location describes the trajectory of the ball in the form of two differential equations (`v'` and `p'`). The piston in Fig. 4(c) can hit the ball only if its position is high enough (`p>=6`). The ball will rebound with a random dampening coefficient both on the floor and the piston (given by the `random` function). The delays between hits of the piston are chosen stochastically according to an exponential distribution of rate 5/2. Semantically, the effect of ODE expressions is achieved by an implicit auxiliary process integrating the values based on a given fixed time step and thus directly competing with the rest of processes when the rest of invariant and guard expressions are evaluated.

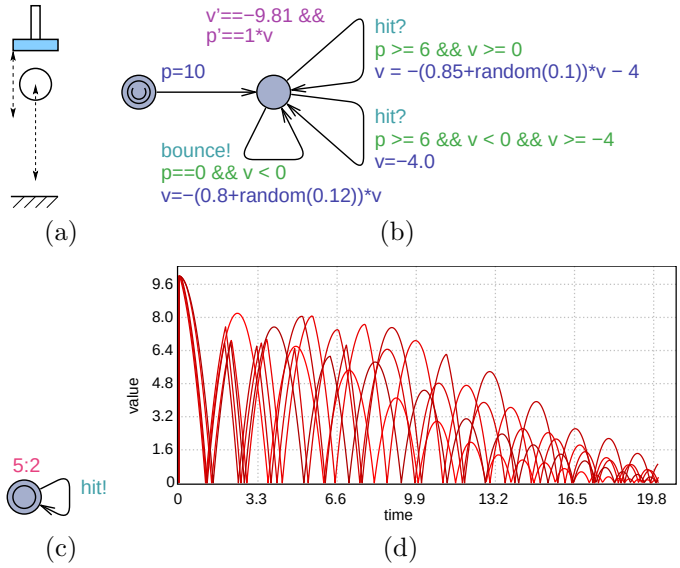


Fig. 4: The bouncing ball and the hitting piston (a), the automata for (b) the ball and (c) the piston, and (d) 5 trajectories of the ball in function of time.

Five different trajectories of the ball are obtained by the query `simulate 5 [<=20]{p}` and shown in Fig. 4(d). We may also ask the model-checker to estimate the probability that the ball is still bouncing above the height of 4 after 12 time units with the query:

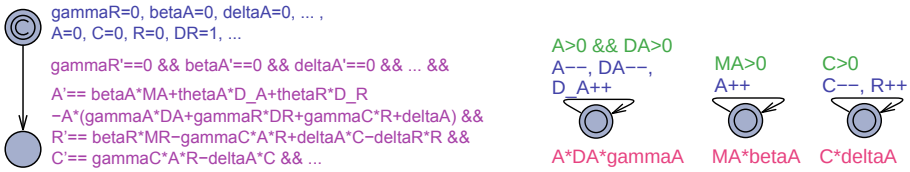
```
Pr[<=20](<>(time >= 12 && p >= 4))
```

Here `<>(time>=12 && p>=4)` is the UPPAAL-SMC syntax for the MITL property $\psi = \Diamond(\texttt{time} \leq 12 \wedge \texttt{p} \geq 4)$ and `Pr[<=20]`ψ denote the probability π that ψ will hold within 20 time-units for a random run. Given this query, the interval $[0.152, 0.163]$ is returned as an estimate for π with confidence 99.9% after generating 152020 runs. We can also test the hypothesis:

```
Pr[<=20](<> time >= 12 && p >= 4) >= 0.15
```

with a region of indifference of ± 0.005 and level of significance of 0.1% after generating 18543 runs.

For the analysis of more general MITL properties properties, UPPAAL-SMC generates monitoring automata to be put in parallel with the system. Statistical model checking requires that these monitors are determinstistic timed automata. Unfortunately, not all MITL properties may be monitored by determinstic timed automata. Thus UPPAAL-SMC offers a safe confidence interval based on two monitors corresponding to under- and over-approximations of the set of runs satisfying the particular formula [5]. Experimental results have shown that we obtain an exact monitor, and most recently this method has been replace by an exact rewrite technique.



(a) ODE model. (b) Stochastic model.

Fig. 5: Snippets of the UPPAAL models of the genetic oscilator.

Figure 5a shows a snippet from an ODE model of the genetic oscilator, where the coefficients (`gammaR, betaA, deltaA`) and variables (`A, C, R, DR`) are initialized with the first urgent transition and then the trajectories are computed based on the ODEs (the last three equations from Fig. 1). A snippet from stochastic genetic oscilator model is shown in Fig. 5b, where each reaction (from Eq. 1, 9 and 12) is modeled by a separate automaton. For example, the first automaton can be read as follows: reaction requires positive quantities of `A` and `DA` (guard conditions), one of each is consumed (`A--` and `DA--`), and one `D_A` is produced (`D_A++`) with an exponential rate `gammaA` times the available quantities of `A` and `DA`.

5 PLASMA

PLASMA is designed to be a high performance and flexible platform for statistical model checking, able to work with multiple modelling and property specification languages. Its basic architecture is shown in Figure 6 and comprises a user interface, a simulation management module, a virtual machine and modules that compile the model and property specifications. Models and properties are

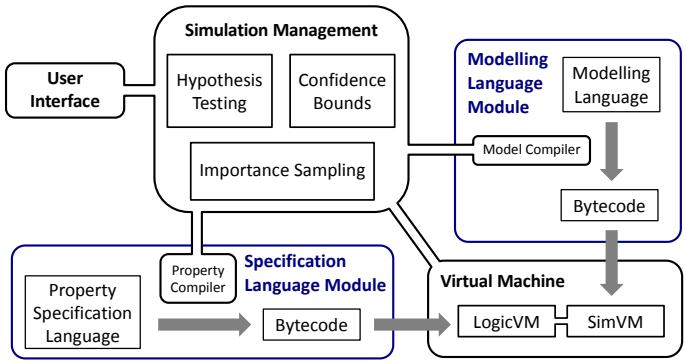


Fig. 6: The architecture of PLASMA

compiled into proprietary bytecode and executed on PLASMA's inbuilt virtual machine. Overall control of the verification process is maintained by the simulation management module according to the options specified by the user. The simulation management module contains various statistical model checking algorithms that implement confidence bounds, such as the Chernoff bound of [11]

and the sequential hypothesis test of [27], plus an *importance sampling* engine [16] that improves the performance when simulating rare properties. For simulating discrete and continuous time Markov models the virtual machine uses the 'method of arbitrary partial propensities' (MAPP [22,15]) that is an optimised version of the 'direct method' of [8]. The simulation management module executes the property bytecode that, in turn, executes the model bytecode until sufficient simulation steps are generated. In this way simulation traces contain the minimum number of states necessary to decide a property and the simulation management ensures that the minimum number of simulation traces are requested of the simulator.

5.1 Modeling and Checking in Plasma

Modelling languages are built on an underlying semantics of *guarded commands* [6] extended with stochastic rates to resolve non-determinism. These have the form $(guard, rate, action)$, where *guard* is a logical predicate over the state of the system which enables the command, *action* updates the state of the system and *rate* is a function over the state of the system that returns the stochastic rate at which an enabled command is executed. This semantics is equally applicable when the rate is actually a probability and time plays no part.

Plasma is designed to be language neutral, so for the present investigation Plasma adopts a simple chemical syntax modelling language that closely mirrors the style of Equations (1-16). The structure of the model file follows the form: constant initialisations, species initialisations, list of reactions. In the present context there is an implicit assumption of *mass action kinetics* [8,7,9] and *rate* specifies the mean of an exponential random variable that models the time between successive reaction events; non-determinism being resolved by races between realisations of the random variables of competing reactions. Reactions of the abstract form $\mathrm{A}+\mathrm{B} \xrightarrow{k} \mathrm{C}+\mathrm{D}$ have the concrete form `A + B k-> C + D` with guarded command semantics $(A > 0 \wedge B > 0, kAB, A = A - 1; B = B - 1; C = C + 1; D = D + 1)$.

Plasma verifies properties specified in bounded linear temporal logic of the kind described in Section 3.2. The logic accepts arbitrarily nested temporal formulae and Plasma achieves this using a buffer to store sequences of values of the variables of interest. When formulae are not nested, no buffer is required. Algorithms 1 and 2 illustrate the basic notions of checking non-nested temporal formulae, employing discrete time for clarity. Algorithm 3 is a naive implementation of a nested formula to illustrate the purpose of the buffer and how we improve efficiency.

Algorithms 1 and 2 generate and consider states in turn, returning a result as soon as ϕ is satisfied or not satisfied, respectively. These algorithms store nothing and generate the minimum number of states necessary. Algorithm 3 also only generates new states as required, but since the inner loop requires states further into the future than the outer loop, states are stored by the inner loop for subsequent use by the outer loop. As written, Algorithm 3 is naive because the

inner loop re-checks states that it has checked on previous iterations of the outer loop. PLASMA therefore records where the decision on the previous iteration was made and then needs only check the states after that. The case with continuous time is more complex because the length of the buffer is not known a priori (there may be an arbitrary number of steps to achieve a given time bound). PLASMA overcomes this by creating an initial buffer and then extends it as required.

Algorithm 1: $\Diamond_{\leq t}\phi$

```
for i = 0 to i = t do
    generate state_i;
    if state_i ⊨ φ then return ⊤
return ⊥
```

Algorithm 2: $\Box_{\leq t}\phi$

```
for i = 0 to i = t do
    generate state_i;
    if state_i ⊭ φ then return ⊥
return ⊤
```

Algorithm 3: $\Diamond_{\leq t_1}\Box_{\leq t_2}\phi$

```
create buffer of length t_2;
for i = 0 to i = t_1 do
    inner = ⊤;
    for j = i to j = i + t_2 do
        if state_j ∉ buffer then
            generate state_j;
            buffer_{j mod t_2} = state_j;
        if state_j ⊨ ¬φ then
            inner = ⊥;
            break
    if inner then return ⊤
return ⊥
```

5.2 Rare Events

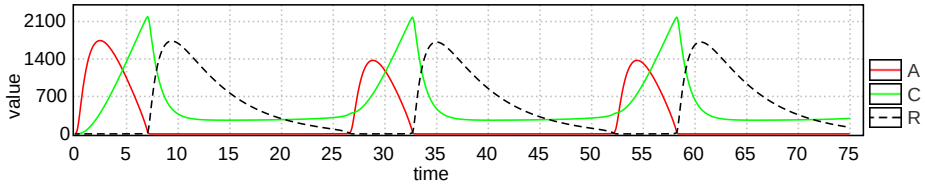
Rare events pose a challenge to simulation-based approaches, so PLASMA includes an *importance sampling* engine that makes it possible to estimate the probability of a rare property by simulating under a distribution that makes the property less rare. Given a property ϕ, with true probability γ under distribution P, the standard Monte Carlo estimator of γ is given by $\tilde{\gamma} = \frac{1}{N}\sum_{i=1}^{N} z(\omega_i)$, where ω_i is the trace of a simulation made under$P$ and $z(\omega) \in \{0, 1\}$ indicates whether $\omega \models \phi$. In general, N must be chosen significantly greater than $\frac{1}{\gamma}$ to accurately estimate γ, hence this is computationally expensive when γ is small. By contrast, the importance sampling estimator is given by $\tilde{\gamma} = \frac{1}{N}\sum_{i=1}^{N} z(\omega_i)\frac{P(\omega_i)}{Q(\omega_i)}$, where Q is ideally a distribution under which traces that satisfy ϕ are uniformly more likely and ω_i is now the trace of a simulation performed under Q. $\frac{P}{Q}$ is called the *likelihood ratio* and in a discrete event simulation can usually be calculated on the fly in constant time. Since Q is chosen to reproduce ϕ more frequently, N may be significantly less than $\frac{1}{\gamma}$. The effectiveness of importance sampling relies on finding a suitable Q.

An optimal importance sampling distribution is one under which traces that satisfy the rare property are uniformly more likely, to the exclusion of all traces that do not satisfy the property. It is possible to find such distributions by individually modifying all the transition probabilities in the system [20], however this is often intractable. PLASMA thus parametrises the distribution with a low dimensional vector of parameters applied to the rates of its guarded commands [16]. In the case of biological systems of the type considered here, this parametrisation corresponds to the rate constants of reactions.

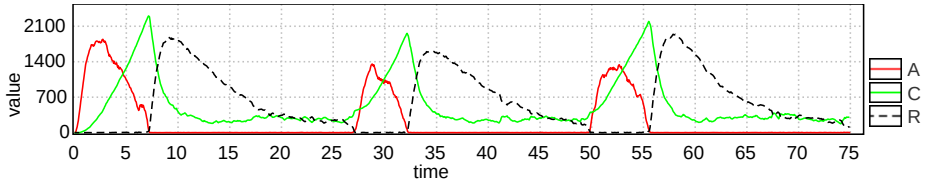
To demonstrate the application of importance sampling to biological systems we consider a simple chemical system comprising $A + B \xrightarrow{k_1=1} C$, $C \xrightarrow{k_2=1} D$, $D \xrightarrow{k_3=1} E$. With initial conditions $A = 1000, B = 1000, C = D = E = 0$, we then consider the property $\Pr[\Diamond\, D \geq 470]$ that has a probability of approximately 2×10^{-10}. By multiplying the rate constants k_1, k_2, k_3 by importance sampling parameters $\lambda_1 = 1.16, \lambda_2 = 1.15, \lambda_3 = 0.69$, respectively, PLASMA is able to estimate this probability using only 1000 simulation runs (of these approximately 600 satisfy the property). The parameters were generated using the *cross-entropy* algorithm described in [16].

6 Experiments

Our first level of validation is to inspect the simulation traces to verify that they are sensible. The result from UPPAAL-SMC is displayed in Figure 7 where the ODE model yields the same (deterministic) trajectories of a pattern which repeats every 26.2 hours. The stochastic model yields an *apparently similar* but "noisy" pattern where the amplitude and periodicity are also varying. This intuitive similarity is made more formal by the frequency analysis in Section 6.1. Notice that the signal C starts at zero which allows A to reach higher amplitude



(a) ODE model simulation plot.



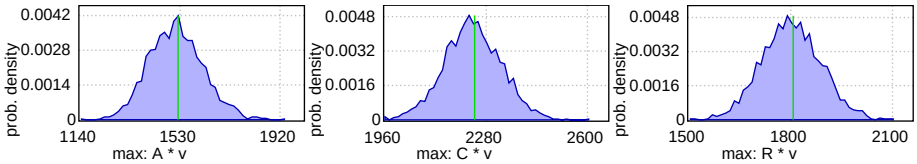
(b) Stochastic model simulation plot.

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

than it normally would, thus we ignore the first period in our measurements. The amplitude can be measured from the plot directly for the ODE model: A – 1375, C – 2183 and R – 1717.

The amplitude is not fixed for stochastic model, and thus the distribution of probable amplitude values is estimated instead. The start of the monitoring is constrained with a variable v gaining value 1 only after 15 time units, effectively ignoring the first peaks. The results of 2000 simulations of 75 time units are shown in Fig. 8. The average amplitudes are a bit larger than in ODE model.

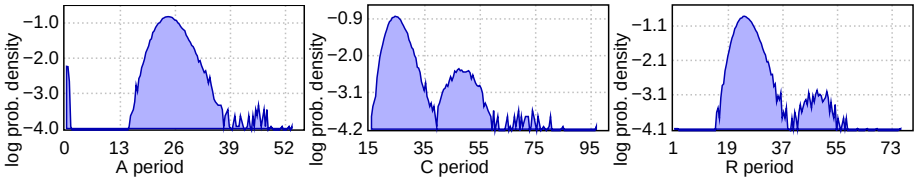
We can also estimate periods of various signals by measuring the time distance between peaks. For this purpose we add signal peak monitor-process gener-



(a) A, mean 1522. (b) C, mean 2243. (c) R, mean 1807.

Fig. 8: Estimated amplitude: `E[<=75; 2000](max: A*v,C*v,R*v)`.

ated by a MITL formula: `true U[<=1000] (A>1100 & true U[<=5] A<=1000)`, which is false unless the signal A rises above 1100 and then falls below 1000 within 5 time units. By coordinating two instances of such monitors and letting a stopwatch x run only between the two peaks we estimate the distribution of x as a period estimate. For signal C (R) we register a peak when the signal falls from 2000 to 1900 (1500 to 1400 resp.). The query then is a simple probability estimate whether the second peak is reached. Figure 9 shows estimated period with corresponding values at probability peaks. The most instances are situated around the peak value, but there are other tiny bumps on sides. The bump near zero corresponds to false positive recognition of a peak when the signal includes a saw tooth (common in stochastic simulations). The bump around 48 hours corresponds to a missed peak at 24, due to the local amplitude being too low. Interestingly there is another tiny bump at around 73 in Fig. 9b which correspond to two missed peaks in a row.



(a) A, peak at 24.22. (b) C, peak at 24.21. (c) R, peak at 24.23.

Fig. 9: Estimated periods: `Pr[x<=100](<> secondPeak.ACCEPT)`.

Similarly we can estimate the phase difference between the signals as a distance between peaks of different signals. Figure 10 shows a probability density distribution of a phase difference between A and R signals, which implies that a peak in A typically leads to a peak in R within 6.21, but there might be one missed peak.

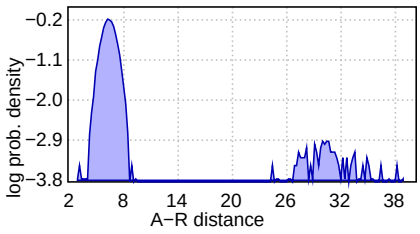


Fig. 10: Phase diff., peak at 6.21.

6.1 Frequency domain analysis

Figures 11 and 12 compare the average frequency spectra of two variables from our deterministic and stochastic simulations. We observe that the deterministic spectra (black) tend to have sharp, well defined, peaks with discernible harmonics at high frequencies. In contrast, the spectra of stochastic simulations (red)

tend to have softened peaks, few discernible high harmonics and contain an apparent continuum of frequencies. This is reflected in the reduced amplitude of these spectra relative to their deterministic counterparts: the amplitude of the original trace is effectively divided amongst many more frequencies. We note that the first three harmonics of the deterministic and stochastic spectra appear to coincide, confirming our expectation and intuition that the the two behaviours are 'similar'.

The apparent *thickness* of the red lines reflects the fact that the lines describe average spectra generated from stochastic data. Increasing N would make the lines less thick but would not change their overall form, that is derived from the frequency characteristics of the stochasticity.

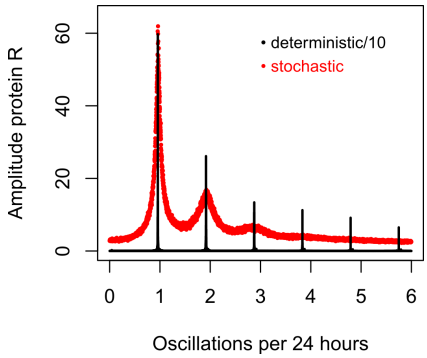


Fig. 11: Average frequency spectra of protein R. $\delta t = 2h, K = 12500$. $N = 100$ ($N = 1$) for the stochastic (deterministic) model.

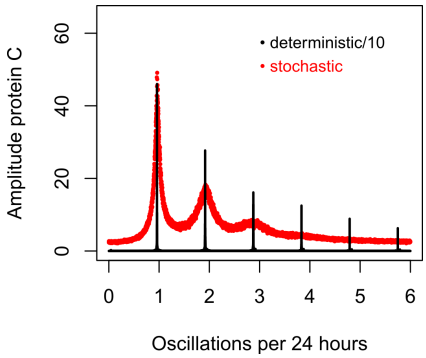


Fig. 12: Average frequency spectra of protein C. $\delta t = 2h, K = 12500$. $N = 100$ ($N = 1$) for the stochastic (deterministic) model.

7 Conclusion

We have introduced and applied various advanced statistical model checking techniques to an osillatory biological system and have demonstrated their relevance to runtime verification. We have used two state of the art tools: UPPAAL-SMC and PLASMA. UPPAAL is a mature general-purpose platform based on timed automata, while PLASMA is a relatively new tool that allows domain-specific languages to describe Markov chain models. Table 1 summarizes the aspects of both tools and shows that PLASMA is targeted for domain specific languages and the final result analysis (such as frequency analysis) is done using external tools, whereas UPPAAL provides generic features with integrated visualization of predetermined concepts. Although there are fundamental differences in the underlying semantics of the two tools, we have shown that they are united by the properties and problems of verification. In particular, frequency domain analysis and rare events are subjects of ongoing joint research.

Table 1: Tool comparison summary.

Aspect	PLASMA	UPPAAL
Models	Domain specific	Stochastic hybrid automata
Semantics	DTMC/CTMC	Stochastic hybrid transition systems
Properties	MITL, rare events	(Weighted) MITL
Results	Exported data	Generic visualizations

References

1. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, Jr., R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Commun. ACM*, 43(5):74–82, May 2000.
2. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
3. N. Barkai and S. Leibler. Biological rhythms: Circadian clocks limited by noise. *Nature*, 403:267–268, 2000.
4. S. Bensalem, B. Delahaye, and A. Legay. Statistical model checking: Present and future. In *RV*. Springer-Verlag, 2010.
5. P. E. Bulychev, A. David, K. G. Larsen, A. Legay, G. Li, D. B. Poulsen, and A. Stainer. Monitor-based statistical model checking for weighted metric temporal logic. In N. Bjørner and A. Voronkov, editors, *LPAR*, volume 7180 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2012.
6. E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Commun. ACM*, 18(8):453–457, Aug. 1975.
7. D. Gillespie. Stochastic simulation of chemical kinetics. *Annual review of physical chemistry*, 58:35–55, 2007.
8. D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977.
9. D. T. Gillespie. Deterministic limit of stochastic chemical kinetics. *The journal of physical chemistry. B*, 113:1640–1644, February 2009.
10. K. Havelund and G. Rosu. Synthesizing monitors for safety properties. In *TACAS*, LNCS, pages 342–356. Springer, 2002.
11. T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In B. Steffen and G. Levi, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 2937 of *LNCS*, pages 307–329. Springer, 2004.
12. R. C. Hilborn and J. D. Erwin. Stochastic coherence in an oscillatory gene circuit model. *Journal of Theoretical Biology*, 253(2):349 – 354, 2008.
13. W. Hoeffding. Probability inequalities. *Journal of the American Statistical Association*, 58:13–30, 1963.
14. A. Ihekwaba and S. Sedwards. Communicating oscillatory networks: frequency domain analysis. *BMC Systems Biology*, 5(1):203, 2011.
15. C. Jegourel, A. Legay, and S. Sedwards. A Platform for High Performance Statistical Model Checking – PLASMA. In C. Flanagan and B. König, editors, *TACAS*, LNCS, Tallinn, Estonia, March 2012. Springer. To appear.
16. C. Jegourel, A. Legay, and S. Sedwards. Cross-entropy optimisation of importance sampling parameters for statistical model checking. In M. Parthasarathy and S. A.

Seshia, editors, *CAV*, LNCS, Berkeley, California, USA, July 2012. Springer. To appear.
17. S. Laplante, R. Lassaigne, F. Magniez, S. Peyronnet, and M. de Rougemont. Probabilistic abstraction for model checking: An approach based on property testing. *ACM TCS*, 8(4), 2007.
18. K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *STTT*, 1(1-2):134–152, 1997.
19. O. Purcell, N. J. Savery, C. S. Grierson, and M. di Bernardo. A comparative analysis of synthetic genetic oscillators. *Journal of The Royal Society Interface*, 7(52):1503–1524, 2010.
20. A. Ridder. Importance sampling simulations of markovian reliability systems using cross-entropy. *Annals of Operations Research*, 134:119–136, 2005.
21. G. Rosu and S. Bensalem. Allen linear (interval) temporal logic - translation to ltl and monitor synthesis. In *CAV*, volume 4144 of *LNCS*, pages 263–277. Springer, 2006.
22. S. Sedwards. A Natural Computation Approach To Biology: Modelling Cellular Processes and Populations of Cells With Stochastic Models of P Systems. PhD thesis, University of Trento, 2009.
23. K. Sen, M. Viswanathan, and G. Agha. Statistical model checking of black-box probabilistic systems. In *CAV*, LNCS 3114, pages 202–215. Springer, 2004.
24. J. M. G. Vilar, H. Y. Kueh, N. Barkai, and S. Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proceedings of the National Academy of Sciences*, 99(9):5988–5992, 2002.
25. A. Wald. Sequential tests of statistical hypotheses. *Annals of Mathematical Statistics*, 16(2):117–186, 1945.
26. H. Younes. YMER: A Statistical Model Checker. In K. Etessami and S. Rajamani, editors, *CAV*, volume 3576 of *LNCS*, pages 171–179. Springer, 2005.
27. H. Younes and R. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *CAV*, volume 2404, pages 23–39. Springer, 2002.