Software Tools for Technology Transfer manuscript No.
(will be inserted by the editor)

Statistical Model Checking for Biological Systems*

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

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

The date of receipt and acceptance will be inserted by the editor

Abstract Statistical Model Checking (SMC) is a highly scalable simulation-based verification approach for testing and estimating the probability that a stochastic system satisfy a given linear temporal property. The technique has been applied to (discrete and continuous time) Markov chains, stochastic timed automata and most recently hybrid systems using the tool Uppaal SMC. In this paper we enable the application of SMC to complex biological systems, by combining Uppaal SMC with ANIMO, a plugin of the tool Cytoscape used by biologists, as well as with SimBiology®, a plugin of Matlab to simulate reactions. ANIMO and SimBiology® are two domain specific tools that have their own user interfaces and formalisms specifically tailored towards the biology domain. However – though providing means for simulation – both tools lack the powerful analytic capabilities offered by SMC, which in previous work have proved very useful for identifying interesting properties of biological systems. Our aim is to offer the best of the two worlds: optimal domain specific interfaces and formalisms suited to biology combined with powerful SMC analysis techniques for stochastic and hybrid systems. This goal is obtained by developing translators from the XGMML and SBML formats used by Cytoscape and SimBiology® to stochastic and hybrid automata, allowing Uppaal SMC to be used as an efficient backend analysis tool, that we demonstrate can handle real-world biological systems by pitting it against the BioModels database. We present detailed analysis on two particular case-studies involving the ANIMO and SimBiology® tools.

1 Introduction

It is conceivable to design systems to make their analysis easier, but usually they are optimised for other constraints (efficiency, size, cost, etc.) and they evolve over time, developing highly complex and unforeseen interactions and redundancies. These phenomena are epitomised by biological systems, which have 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 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 that biological systems can be analysed using the same tools used to verify for instance 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.

There already exists several formal tools dedicated to this purpose. As an example, the ANIMO toolset [22] can be used to model biological pathways. This tool handles phenomena described via the Cytoscape library [24]. The tool chain goes via a translation from the library to timed automata, which allows to exploit Uppaal for verifying properties of the system. Unfortunately, ANIMO is restricted so that it cannot capture the stochastic behaviours inherent to many biological phenomena. Moreover, its expressive power is restricted to timed automata while many behaviours have to be described via general Ordinary Differential Equations (ODE). Another

* Work is supported by the VKR Center of Excellence MT-LAB and by the Sino-Danish Basic Research Center IDEA4CPS, DNRF86-10.

interesting toolset is MATLAB with SimBiology®[1] frontend from Mathworks. While MATLAB is clearly powerful enough to handle complex phenomena, including non linear ones, it does not provide efficient verification techniques and powerful logics to describe eventually complex properties one may want to measure and check on the model.

In this paper, we propose a new tool-chain for the analysis of biological systems. Our approach heavily relies on Statistical Model Checking (SMC) [20, 23, 26], a powerful formal approach that has recently been proposed as a 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) 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 [27, 4, 18] that have defeated well-known numerical-based analysis tools on several non-academic case studies. Such tools have been applied to large-size industrial applications such as the verification of a complex aircraft control system [3], schedulability analysis of mixed criticality systems [13] or more recently for complex systems of systems within the integrated project DANSE[2] and performance evaluation of energy-aware buildings in the Sino-Danish Basic Research Center IDEA4CPS[3].

In fact, one shall see that biological phenomena can be represented by networks of Stochastic Hybrid Automata (SHAs). SHAs are timed automata whose clocks can evolve with different rates, which may depend not only on values of discrete variables but also on the value of other clocks, effectively amounting to ordinary differential equations (ODEs). In [10, 11], we showed that SHAs can be equipped with a stochastic semantic based on stochastic delays and repeated races between the components of composite model. Importantly, the stochastic semantics provide the foundation for well-defined probability measures for a range of linear temporal properties. In the present paper, we shall see that this model is general and can be used to capture a wide range of biological phenomena. More precisely, our approach can handle biochemical reactions that rely on the interaction of molecules of different species. In one approach, the models based on *elemental reactions* with *mass action kinetics* may be *simulated exactly* as a continuous time Markov chain (CTMC) having discrete states. Alternatively, in case of huge amount of molecules, approximate analysis may be preferable (or necessary) using sets of coupled ODEs that assume continuous states. The method of conversion between these two paradigms, along with explanations of the emphasised terms, is given in [16].

We then see how to connect ANIMO and MATLAB SimBiology® to UPPAAL SMC. UPPAAL SMC is a stochastic and statistical model checking extension of UPPAAL that relies on the SHA model described above. 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 engine has been applied to a wide range of examples ranging from networking and Nash equilibrium[7] through systems biology [14, 11], real-time scheduling [13] to energy aware systems [12]. Our connection is based on intermediary translations to XGMML (used by ANIMO) or SBML (used by SimBiology®) to the CTMCs and ODEs format of UPPAAL SMC. By connecting ANIMO and SimBiology® to UPPAAL SMC, we not only unify their expressive power, but also make the powerful simulation engine of SMC available for efficient verification of complex behaviours. It is worth observing that our transformation is general and can be used to connect UPPAAL SMC to other libraries to describe biological phenomena, including the BioNetGen framework of Faeder et al. [15].

Finally, we compare the performances of ANIMO, SimBiology®, and UPPAAL SMC on several biology examples. The structure of the paper is as follows. Section 2 presents the formalisms used in ANIMO and Simbiology, Section 3 reviews the SHA modelling formalism of UPPAAL SMC as well as application of its SMC engine, Section 4 presents our translators from ANIMO and SimBiology® (SBML format in fact) to UPPAAL SMC, and Section 5 focuses on two case-studies and compares the different tools.

2 Modelling Formalisms for Biology

Here we introduce a simple example and use it to demonstrate features of ANIMO and MATLAB SimBiology®.

Suppose we have two reactants A and B which produce C with a rate $k_{on} = 0.2m^{-1}s^{-1}$, the reaction can be reversed with a rate $k_{off} = 0.1s^{-1}$ and the product C can decay into some other materials with rate $k_{deg} = 1.0s^{-1}$. The system can be described by the following chemical reactions:

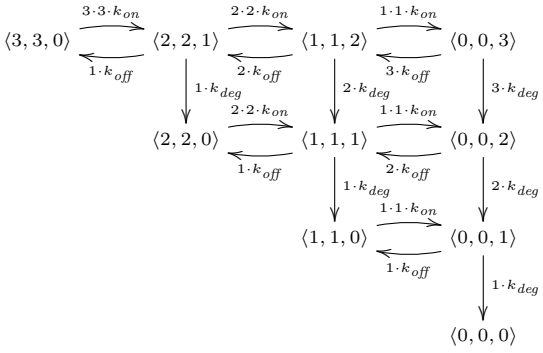
$$\mathsf{A} + \mathsf{B} \underset{k_{off}}{\overset{k_{on}}{\rightleftharpoons}} \mathsf{C} \xrightarrow{k_{deg}}$$

Due to results of [16], such a system of chemical reactions can be modelled as a CTMC under the assumptions that at most two molecules can participate in one

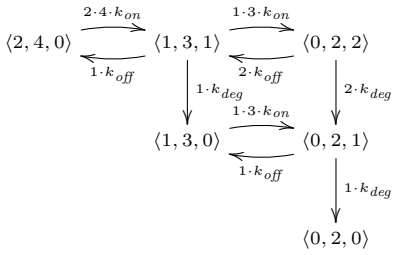
[1] http://www.mathworks.se/products/simbiology/

[2] http://www.danse-ip.eu/home/

[3] http://www.idea4cps.dk/



(a) Balanced quantities.



(b) Unbalanced quantities.

Figure 1. Continuous Time Markov Chain models of reactions.

reaction instance and that molecules are well mixed (uniform reaction rates). Fig. 1(a) shows a CTMC model of our reactions where the state $\langle[\mathsf{A}], [\mathsf{B}], [\mathsf{C}]\rangle$ corresponds to the number of molecules of each reactant and transition probability is proportional to the reaction rate and available reactant molecules. For example, if we start with a mix containing [A]=3, [B]=3 and [C]=0 molecules (state $\langle 3, 3, 0\rangle$ in Fig. 1(a)), then the molecules of A and B may react with a probability rate of $3 \cdot 3 \cdot k_{on}$ by consuming one molecule of each A and B and producing one C, thus the resulting target state contains [A]=2, [B]=2 and [C]=1 molecules (state $\langle 2, 2, 1\rangle$). The other reactions can be modelled likewise. Eventually all of our molecules decay into other materials which do not participate in our system resulting in a dead-end state $\langle 0, 0, 0\rangle$. If the initial quantities are not so well balanced, then the final state may contain some of the initial reactants like in Fig. 1(b). It is easy to see that the modelling approach can be generalised by encoding the quantities of species as variables and disregarding the reactions where the reaction rate is zero due to some reactant having zero molecules.

Alternatively, when there is an abundance of reactants (the number of molecules is large enough that changes can be viewed as continuous) the system of reactions can be modelled as a dynamical system using

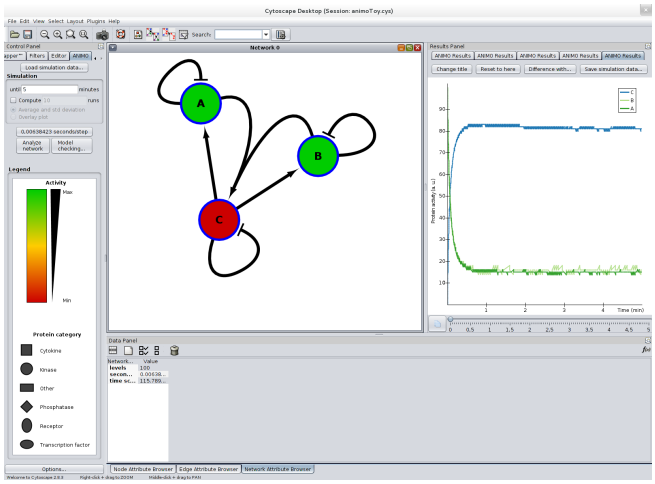


Figure 2. Screenshot from ANIMO.

ordinary differential equations (ODE):

$$\frac{d[\mathsf{C}]}{dt} = +k_{on}[\mathsf{A}] \cdot [\mathsf{B}] - (k_{off} + k_{deg}) \cdot [\mathsf{C}] \quad (1)$$

$$\frac{d[\mathsf{A}]}{dt} = -k_{on}[\mathsf{A}] \cdot [\mathsf{B}] + k_{off} \cdot [\mathsf{C}] \quad (2)$$

$$\frac{d[\mathsf{B}]}{dt} = -k_{on}[\mathsf{A}] \cdot [\mathsf{B}] + k_{off} \cdot [\mathsf{C}] \quad (3)$$

In our examples we will assume the initial conditions as follows: [A]=50m, [B]=80m, [C]=0m.

2.1 ANIMO

ANIMO [21, 22] is a tool developed for modelling biological pathways. In Fig. 2 we provide a screenshot of the main window of the tool. At the center of the screen is a model of our running example. In this model the nodes represent species and the edges represent reactions. To the right of this is presented a single simulation of the model. Below this simulation is a slider that the user can use to zoom into a specific time point of the simulation: during this movement the representation of the model is updated such that the colouring of the nodes represent the quantity(colouring scheme is shown in the left of the screen) of each species and the thickness of the edges represent how likely that reaction is to occur next. In this manner ANIMO gives a visual representation of the current state of the pathway.

In ANIMO a molecule can be in an inactive or active state and the reactions of the pathway may alter the state of a single molecule[4]. If a reaction activates a molecule we call it an activation reaction and if it deactivates a molecule, we call it an inhibitory reaction. In the graph an edge $A \to C$ means A activates C and $A \dashv C$ means A inhibits C. In ANIMO the amount of

[4] In fact the colouring of the nodes represent how large a fraction of each species is active

each species is fixed and given as a user defined value known as the number of activation levels. An activation reaction increases the current activation level of a species and inhibitory reactions decrease the current activation level. Initially the species starts at an activity level defined by the user.

The example in Fig. 2 is the model of our running example. Because reactions in ANIMO can only influence one species (and condition its behaviour on others) the model does not correspond exactly to the description. For instance, the reaction binding A and B molecules into a C molecule is split into three separate reactions: one increasing the activity level of C and two decreasing the activity level of A and B respectively. Similarly, the splitting of a C molecule into A and B is given as three reactions. A final difference from the description is that the model does not incorporate the decay of C molecules into "something else".

The graph view of a pathway shows how species influence each other but provides no visual means to indicate how the species influence the time before a reaction occurs. The user controls this by choosing an appropriate reaction type and by setting a reaction rate k. Also the user should provide a scaling factor `timeScale` that translates the model time units to real-world time units.

ANIMO supports three kinds of reactions:

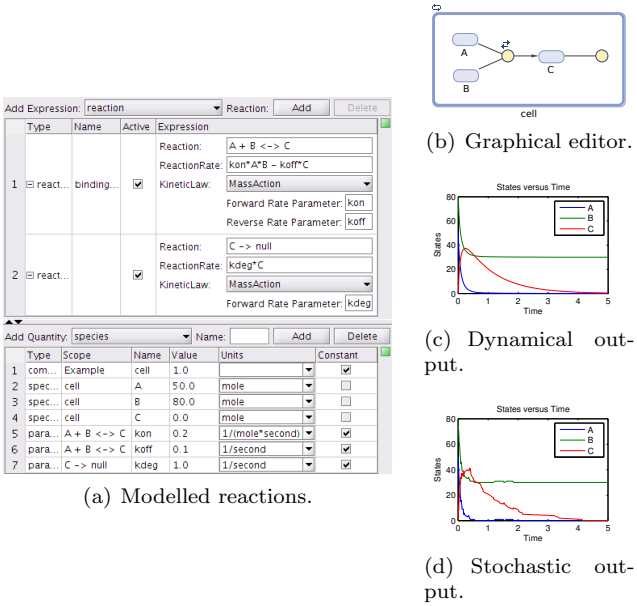
- A reaction $C \rightarrow A(\ C \dashv A)$ is a type-1 activation (inhibition) reaction, if the time before the reaction occurs only depend on the activity level of A ,
- a reaction $E \rightarrow F$ $(E \dashv F)$ is a type-2 activation (inhibition) reaction, if the time before the reaction occurs depends on the activity level of E and the inactivity (activity) level of F and
- the last reaction type supported by ANIMO is a type-3 reaction. In this reaction scheme two species, called reactants, inhibits/activates a third species - an example of this reaction is the double arrow from A and B to C in Fig. 2. The time before the reaction occurs depend on the activity level of the reactant, the inactivity levels of the reactants or the activity of one and inactivity level of the other reactant.

The time before a reaction occurs is selected in the interval $[0.95 \times f, 1.05 \times f]$ where f is calculated differently depending on the reaction type as decribed by Schivo et al. [21].

In ANIMO the pathway is translated into a network of timed automata. We notice that it is more natural to use exponential distributions for reaction times compared to the uniform distributions in ANIMO.

2.2 MATLAB *SimBiology®*

MATLAB SimBiology® is a tool for modelling, simulation and analysis of dynamic systems with a focus on pharmacokinetics/pharmacodynamics and systems biology. The



(a) Modelled reactions.

(b) Graphical editor.

(c) Dynamical output.

(d) Stochastic output.

Figure 3. MATLAB SimBiology® modelling and simulation using different solvers.

tool features include an editor with textual and graphical notation to model chemical reactions. For example, the screenshot in Fig. 3(a) shows that reactants, reaction rates and kinetic laws of our basic example can be specified at the upper half, and the quantities with units can be declared at the lower half. Besides simple reaction specification, the tool provides means of grouping reactions into compartments allowing to model physical isolation of materials, also the coefficient declarations can be separated by a reaction scope allowing reuse of variable names. The reactions can be specified using graphical notation interchangeably with textual notation. For example our reactions were automatically rendered in graphical diagram shown in Fig. 3(b): species are drawn as blue ellipses, reactions appear as yellow circles, lines denote participating reactants and arrows point to reaction products. Lines are dashed if a reactant is only participating as a catalyst but is not consumed (a few instances are shown in Fig. 12).

Once the model is complete, SimBiology® can simulate the model as either dynamical system using stiff numerical differentiation formula solver (`ode15s` stiff/NDF, the plot is shown in Fig. 3(c)) or stochastic simulation using `ssa` solver (plot shown in Fig 3(d)). The simulation can be accelerated by compiling the model into native executable code. Note that the solver has to be selected for entire simulation and thus stochastic and dynamical phenomena cannot be combined in one simulation.

3 UPPAAL SMC

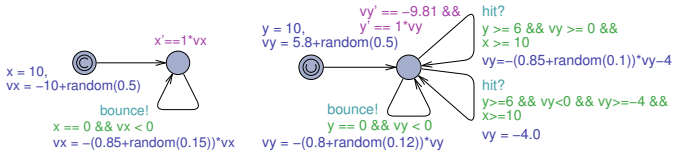
The verification tool UPPAAL [19, 5] provides support for modelling and efficient analysis of real-time systems

modelled as networks of timed automata [1]. To ease modelling, the tool comes equipped with a user-friendly GUI for defining and simulating models. Also, the modelling formalism extends basic timed automata with discrete variables 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 *timed computation tree logic* supporting a range of safety, liveness and bounded liveness properties.

UPPAAL SMC is a recent branch of UPPAAL which supports statistical model checking of *stochastic hybrid systems*, based on a natural stochastic semantics [11]. Firstly, 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. Secondly, UPPAAL SMC comes equipped with a stochastic semantics [10] that refine the non-deterministic choices that may exist with respect to delay, output as well as next state. For delay of individual components, 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 $[0, b]$). The stochastic semantics of a network is given in terms of repeated races between the constituent components, where in each round the component choosing to output at the earliest time-point is the winner.

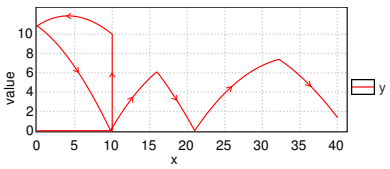
The specification formalism of UPPAAL SMC is that of *weighted metric temporal logic*(WMTL) [9, 8] with respect to which four different (simulation-based) statistical model checking queries are supported: hypothesis testing, probability estimation, probability comparison and simulation. Here the user 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 [9].

Throwing & Bouncing Ball To give an illustration of the expressive power of the modelling formalism of UPPAAL SMC, we consider a variant of the well-known bouncing ball. In our version the ball is initially thrown against a wall, bounces against it, and then continues its trajectory by falling and bouncing against the floor. In addition, an inexperienced player tries to hit the ball randomly according to an exponential distribution. The model is depicted in Fig. 4(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. 4(a)) is initialised to 10 with an uncertain derivative `vx` uniformly distributed between $[-10, -9.5]$



(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 4. Models and a trajectory of a thrown/bouncing ball hit by a player.

(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. 4(b)) 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 Fig. 4(d). 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:

$$\Pr[\text{<=20}](\text{<> time>=12 and y>=4})$$

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

$$\Pr[\text{<=20}](\text{<> time>=12 and y>=4}) >= 0.45,$$

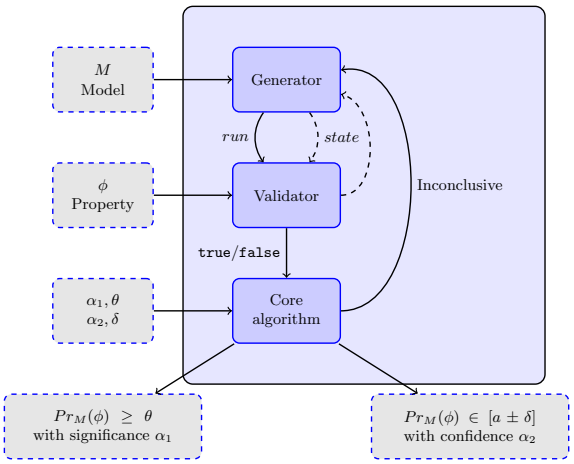


Figure 5. High level view of a statistical model checker for probability estimation and hypothesis testing. The dashed lines correspond to UPPAAL SMC optimised generation/validation loop and the solid lines to a generic statistical model checker.

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. In the queries the [<=20] tells the engine that the property should be true within the first 20 model time units hence the sample runs should only have a length of 20 model time units..

UPPAAL SMC, and in principle any statistical model checker, consist of three components: A generator, a validator and a core algorithm as depicted in Fig. 5. The generator takes as input a model M and produces a single run of a given length, the validator takes as input a property ϕ and a run and returns `true` if the run satisfies the property and `false` otherwise. This control flow is shown by the solid lines in Fig. 5. The core algorithm essentially "keeps score" of the number of `true` and `false` results, in order to perform hypothesis testing or estimation with respect to the unknown probability $Pr_M(\phi)$ using classical statistical algorithms. For hypothesis testing, additional information with respect to required significance level (α_1) and threshold probability (θ) is required. For estimation, information concerning the size of the confidence interval (δ) and the required confidence level (α_1) is required.

In order to increase performance, UPPAAL SMC is validating a run while it is generated. Here the generator passes a single state forward to the validator and the validator can ask for the next state in the run being generated or may pass `true` or `false` to the core algorithm. This one-step validation is shown by the dashed lines i Fig. 5 and is essentially made to avoid generating and storing long runs, e.g. 1000 steps, if the property may already be settled after a few, e.g. two, steps.

Weighted Metric Temporal Logic Besides pure reachability properties UPPAAL SMC also supports Weighted MTL (WMTL) properties with a point-wise semantics. An example of a WMTL property is that the ball within 4 but not earlier than 2 time units should reach a height greater than 6 and afterwards within 1 time unit should drop below 4. In WMTL we can describe this as

$$(\texttt{true}\, U_{[0;4]}\, (y > 6 \wedge (\texttt{true}\, \mathrm{U}_{[0;2]}\, y < 4)),$$

in which an expression as $\phi_1 \mathrm{U}_{[a;b]} \phi_2$ should be interpreted as ϕ_1 must be true until ϕ_2 is true and ϕ_2 must be true before b time units have passed but not before a time units.

The syntax of WMTL extends propositional logic with time- and weight-constrained Until and Release modalities:

$$\begin{aligned}\varphi ::= & p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \\ & O\varphi \mid \varphi_1 \mathrm{U}^x_{[a;b]} \varphi_2 \mid \varphi_1 \mathrm{R}^x_{[a;b]} \varphi_2 \mid \\ & \texttt{true} \mid \texttt{false}\end{aligned}$$

where x is a clock that for any infinite run will exceed any given bound of the system (we omit this clock when we bound over the global time), p is an atomic expression, $a, b \in \mathbb{Q}$ and $a < b$. An expression $\phi_1 R_{[a;b]} \phi_2$ means that ϕ_2 must be true until both ϕ_1 and ϕ_2 are true and they should be true after a time units and before b time units. Alternatively ϕ_2 is true from now and until b time units have passed. The expression $O\varphi$ means φ should be true in the next observation.

In UPPAAL SMC there are two different ways of using WMTL. For the MTL fragment of WMTL a MTL property can be passed directly to the engine[8]. For the full WMTL language, the formula can be converted into an observer automaton and that automaton be parallel composed into the system[9]. The observer is guaranteed to reach a specific location if the property is satisfied and another if its not satisfied thus we can use the optimised reachability engine of UPPAAL SMC to verify WMTL. There is a catch though - sometimes an exact observer cannot be constructed and only an over or underapproximation can be made.

In adition to verify properties in WMTL, we can also use the observers to, at run time, detect certain phenomena of the system (peaks for instance). By coordinating multiple observers we can detect the period of the periodic behaviours of the observed system. In Section 5.2 we will see an example of this usage.

4 Translators

4.1 Overview

We have implemented two tools to translate (i) network description files (XGMML) with the semantics of the ANIMO plugin, and (ii) a subset of the standard format

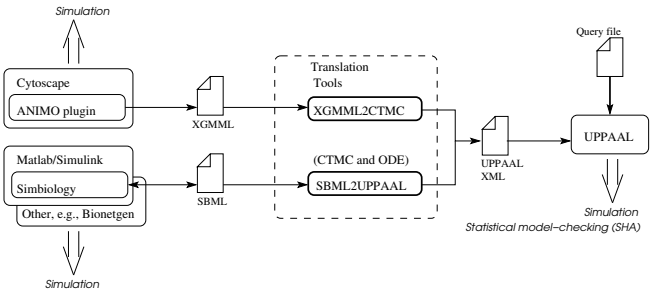


Figure 6. Overview of our tool chain.

SBML[5] to UPPAAL SMC stochastic hybrid automata (XML). Fig. 6 gives an overview of our translations. We can export ANIMO models to XGMML within the Cytoscape tool and translate this format into a CTMC UPPAAL SMC model. From the SimBiology® plugin we export SBML that we can translate to both a CTMC or an ODE UPPAAL SMC model. We note that we could connect to other tools that use this standard format, such as *BioNetGen*. The translators complement the functionalities of other tools by offering statistical model-checking. In addition, we can envision that UPPAAL SMC could be used as a backend to other tools that translate models from a domain specific formalism to stochastic hybrid automata.

Implementation and Availability The translators are implemented in C++. The XGMML translator uses the *rapidxml* library and the SBML translator the *libsbml* library. Our translators are available with the distribution of UPPAAL version 4.1.14. The translator for XGMML interprets the network according to the semantics of ANIMO only. The translator for SBML is general and has been tested against the BioModels database of biological models[6].

4.2 General Principle of The Translations

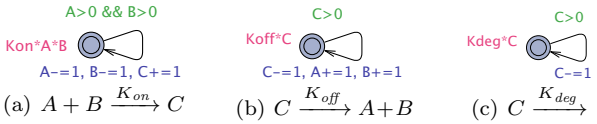
To explain the general principles of our translations, we consider the following example that is representative of the types of reactions we suppot:

$$A + B \underset{k_{off}}{\overset{k_{on}}{\rightleftharpoons}} C \xrightarrow{k_{deg}}$$

Here, A and B are reactants, C is the product of the first reaction, and C is itself a reactant for a second reaction where it degrades. In general, we can have more reactants or products. To model these reactions we need to pick a kinetic law V for each of them (in function of the concentrations of the reactants). If we separate the reactions we have:

$$\begin{aligned} A + B &\xrightarrow{k_{on}} C & V_{on} &= k_{on}[A][B] \\ C &\xrightarrow{k_{off}} A + B & V_{off} &= k_{off}[C] \\ C &\xrightarrow{k_{deg}} & V_{deg} &= k_{deg}[C] \end{aligned}$$

We note that the kinetic laws may contain some extra terms, e.g., concentrations of catalysts that are not explicitly present in the reactions. These reactions can be Modelled as continuous Markov chains (the stochastic model) or with ordinary differential equations (the ODE model).



(a) $A + B \xrightarrow{K_{on}} C$ (b) $C \xrightarrow{K_{off}} A+B$ (c) $C \xrightarrow{K_{deg}}$

Figure 7. The three stochastic hybrid automata to model the three reactions of our example.

Basic Stochastic Model The stochastic model for these reactions is a network of simple stochastic hybrid automata. The template for each reaction has one transition with:

- A guard that ensures there is enough reactant, in our case A>0 && B>0 for the first reaction of our example.
- Updates that encode the actual reaction, e.g., A−=1, B−=1, C+=1.

Furthermore, the kinetic law is encoded in the exponential rate that define the distribution of the delay to take this transition. For this example we have Kon∗A∗B. Fig. 7 shows the three stochastic automata used to model the three separate reactions of our example. The automatic generation of these automata is straight-forward from the reactions. The variables used are integers since we only need counters. This may require rescaling the values of the initial reactions.

Basic ODE Model To model with ODEs, we derive the equations from the reactions and their kinetic laws. The derivative for each reactant is the weighted sum (positive if produced or negative if consumed) of the kinetic laws of the reactions that involve this reactant. For our example, we obtain the derivatives of equations 1, 2 and 3 in Section 2.

To model these equations in UPPAAL SMC, we use one automaton with one transition from an initial state to a main state to initialise the reactants, and then the equations are declared in the invariant of the main state as shown in Fig. 8. In practice, the values of the constants are inlined in the model.

The generation of the model is done by first collecting the reactants and the different sums for their derivatives and second by writing the resulting list of rates.

[5] `http://sbml.org`
[6] `http://www.ebi.ac.uk/biomodels-main/publmodels`

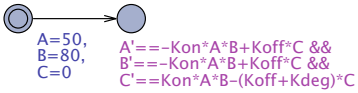
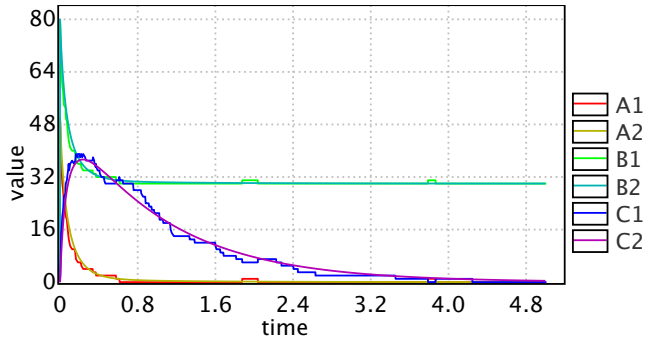
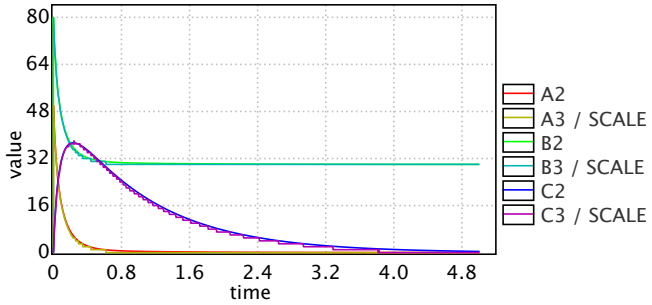


Figure 8. The ODE hybrid automaton to model the three reactions of our example.



(a) Unscaled CTMC and ODE simulations.



(b) Scaled CTMC and ODE simulations.

Figure 9. Simulation results of the CTMC and ODE models for our running example. The $\{A, B, C\}1$ variables are from the non-scaled CTMC model, the $\{A, B, C\}2$ variables are from the the ODE model and the $\{A, B, C\}3$ variables are from the scaled CTMC model.

The variables are clocks here. We do not need to rescale values as for the stochastic model.

Example We show the results of the simulation of our example in Fig. 9. Fig. 9(a) compares the simulations of the CTMC and the ODE models (the ODE corresponding to the smooth plot). To increase the precision of the CTMC simulation (done with small numerical values), we can scale the amounts by 100 (and correct the kinetic laws). Fig. 9(b) plots the scaled-down values from the scaled model. The plot shows how the CTMC is now much closer to the ODE solution and is in fact its discretization. This illustrates that by taking large amount of reactants, the behaviour converges to the solution of the ODEs.

4.3 ANIMO to Uppaal *SMC*

ANIMO models are exported to the XGMML format, which is a general XML based format for representing networks. Our tool recognises the types of nodes and their semantics used by ANIMO. The translation from ANIMO follows our general principle for translating the reactions into CTMCs. Let A, B and C be species with nA, nB and nC activation levels and let the time scale in ANIMO be ts, then the translation is performed as shown in Table 1.

	Type	Guard Exponential	Update
$A \dashv B$	1	A > 0 && B−1>0 nB/(nA∗ts)∗k∗A	B=B−1
	2	A>0 && B−1 >=0 && B>0 1/(nA∗ts)∗k∗A∗B	B=B−1
$A \to B$	1	A > 0 && B+1<=nB nB/(nA∗ts)∗k∗A	B=B+1
	2	A>0 && B+1 <=nB && (nB−B)>0 1/(nA∗ts)∗k∗A∗(nB−B)	B=B+1
$A, B \dashv C$	3	f(A,actA) > 0 && f(B,actB)>0 && C−1>0 nB/(nA∗nB∗ts)∗k∗f(A,actA)∗f(B,actB)	C=C−1
$A, B \to C$	3	f(A,actA) > 0 && f(B,actB)>0 && C−1>0 nB/(nA∗nB∗ts)∗k∗f(A,actA)∗f(B,actB)	C=C+1

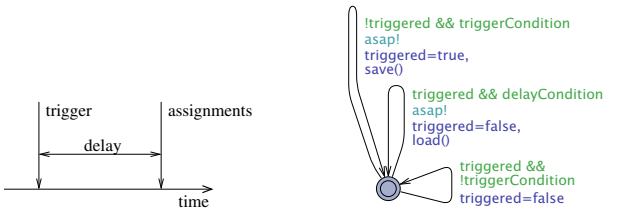
Table 1. Overview of the translation where $actA, actB \in \{active, inactive\}$, $f(X, active) = X$ and $f(X, inactive) = nX - X$.

Recall that the reaction time in ANIMO was chosen uniformly in the interval $[0.95 \times f; 1.05 \times f]$, f being calculated differently depending on the type of reaction. The exponential rate we use in our translation to CTMCs correspond to $\frac{1}{f}$ thus we ensure the average reaction time is the same for our translation and the semantics for ANIMO.

4.4 SimBiology® to Uppaal *SMC*

SimBiology® models are exported to SBML, a standard format to describe biological systems. The SBML language has been designed for biologists and can be used by several tools, in particular *BioNetGen* or *SimBiology®*. SBML is rich and is unfortunately not used in a standard way nor do different simulators agree on simulation results [6], which shows that handling general models is difficult: As an example, for the model #24, only six out of 12 simulation packages returned a result *and* only two of these seemed to agree on a specific behaviour. We choose to support a subset of SBML and to judge if it is relevant and to assess the validity of our translations, we apply our tool to all the 436 models of the BioModels database. According to [6], *The curated models in the BioModels Database cover a wide range of features of the SBML language and are therefore an optimal choice as a base set of models for simulator comparison.*

Translation To translate SBML models we use both the basic stochastic model and the ODE models that we need to extend to accommodate the extra features of SBML. We mention the following features that need extra handling:



(a) Time frame of events.

(b) Simplified template for encoding events.

Figure 10. Events in SBML and their encodings.

- Functions: User functions in the form of *lambda expressions* can be defined. These functions return floating point values, which is not yet supported by UPPAAL SMC, so we inline them in the model.
- Compartments: Species exist in compartments that have volumes, e.g., a cell. We use compartments as namespaces.
- Species: Species (names for the reactants and products) are declared with either initial amounts or initial concentrations (this is inconsistent across models). We choose on-the-fly the right initial state. In addition, quantities are not integers in general so our tool suggests a scaling factor (option `--scale value`) to scale the values and update the kinetic laws automatically.
- Parameters: Models can be parameterized by global or local parameters (local to kinetic laws). We inline the definition of these parameters in formulas where they are used if they are constants. If parameters are not constants, they are typically used in event assignments, in which case they are added to the model as extra variables.
- Assignment and rate rules: The models may contain reactions and explicit equations that define how species evolve. The explicit definition of $[A] = expr$ is an assignment rule and $d[A]/dt = expr$ is a rate rule. Sometimes assignment or rate rules are present together with reactions involving the same species and they may conflict. When rules are given, we consider that they override any equation that we may infer from the reactions. Assignment rules are inlined in formulas and rate rules override the derivatives. It is forbidden to have both assignment and rate rules for the same species.
- Names: Species are not used directly in the model, but rather species references (to some identifier). Sometimes the identifier is misused as the name and sometimes the name of the species is the right name. By default our translator uses IDs but there is an option (`--name`) to use the names of the species instead. This helps to understand the output.
- Events: External events to the reactions can occur, e.g., an operator pours a reactant or a cell divides. They are akin to discrete transitions in hybrid systems. An event is triggered when its trigger condition is true. Event assignments are performed after an optional delay with either the state or the state when the trigger occured (that means it has to be saved). Events can be *persistent*. A non-persistent event means that the trigger condition has to hold until the event assignment takes place otherwise it is cancelled. Fig. 10(a) shows the time frame of such events. Fig. 10(b) shows the extra transitions for each event that are added to the basic ODE model[7]. The flag `triggered` records the occurrence of the event, `triggerCondidion` is the trigger condition, `delayCondition` is the optional delay condition, `save()` is replaced by saving the clocks needed for the assignment if the values at the trigger are needed, and `load()` is replaced by the actual event assignments that read either the current values or the values saved. The bottom transition can disable the event if it is not persistent.

We do not support the other features. In particular, we currently assume that the units in the model are consistent. Units can be handled via rescaling and this is not a fundamental limitation. On the other hand, algebraic rules that define equations of the form $f(x_i) = A$ where $f(x_i)$ is some arbitrary function depending on species and A a constant, are more problematic. In practice they are rarely used (not at all in the BioModels database) so this is not a limitation either.

Validating the Tool To validate and assess the tool we use the following methodology. We download all curated models from the BioModels database. A curated model is a manually sanitised model. We run our translator on all of them and then we (manually) simulate all of those that were successfully translated. We record the following results:

- We found models with negative initial amounts or that use reaction identifiers as variables. They cannot be translated and are considered to be *inconsistent*.
- We failed to translate some models that use unsupported features.
- For the models we successfully translate (72.3%), we can simulate CTMC or ODE models (or both)[8]. The simulations may also fail mainly due to numerical problems.

The detailed results of our validation experiments can be found in Table 6 at the end of this paper and are summarised in Table 2. These experiments show that we can obtain meaningful simulations *automatically out of 40.4% of all the models from this database*, which is a positive result considering that these models are not easy to handle even by specialised tools in biology. This

[7] They can be added to the CTMC model as well but this is not yet implemented.

[8] The individual scaling or simulation steps are not reported here for brevity.

Inconsistent models	2.7%
Unsupported features	25%
CTMC and ODE	10.1%
CTMC or ODE	30.3 %
Failed simulations	31.9%

Table 2. Synthetic results on the database in percentage of the 436 models.

shows that Uppaal SMC can realistically be used as a backend tool to handle real-world models. In the next section, we study in details one of these models (number 35).

Remark 1. Some of the models used for the experiments are stiff and the step size has to be adjusted for the integration to work. In particular, the biological oscillator (model 35) needs a step size of 10^{-4}.

5 Case Studies

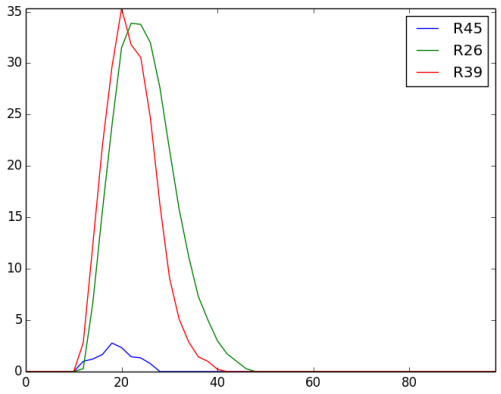
In this section, we demonstrate the intended usage and benefit of the developed tool chain on two case studies.

5.1 PC-12 Neural Pathways

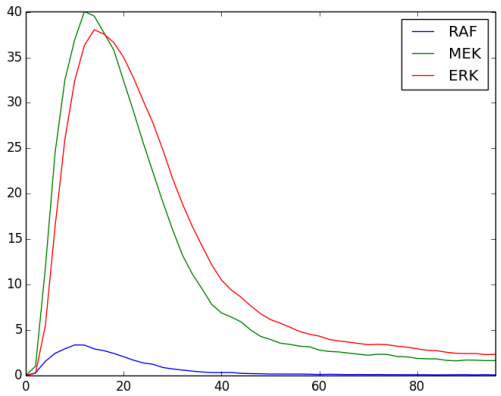
We consider the case that the authors of [22] used to exemplify the use of ANIMO. The case is a pathway that coordinates the neural differentiation of PC-12 cells. The original ANIMO model was obtained from the authors and afterwards translated into a Uppaal SMC model using `xgmml2ctmc`. We use this model to compare the model of ANIMO with our translated model. For the experiments we have used version 2.55 of ANIMO.

For the ANIMO model we generated 100 runs and took the average on each sampling point to obtain an "expected" run. We did similarly for our translated but for 10000 runs - we used a greater number of runs for our model to accommodate for the higher variability that the exponential distributions give.

In Fig. 11(a) we have plotted the expected run from ANIMO and in Fig. 11(b) we show the expected run of Uppaal SMC. By visual inspection we notice that the graphs of ANIMO and Uppaal SMC are not lining up perfectly. However, they do share a common structure where ERK, and MEK rises rapidly in the beginning and then drops and approaches zero as the time progresses. Similarly RAF rises in the beginning and drops afterwards. The primary difference in the runs is that the Uppaal SMC run takes longer to reach the maxima for the species and equivalently takes longer in approaching zero. This difference may easily be explained by our use of exponential distribution versus the uniform distributions of ANIMO.



(a) A simulation from ANIMO.



(b) Average run from Uppaal SMC.

Figure 11. Comparison between Uppaal SMC and ANIMO. The time is on the x-axis and on the y-axis we have the activity levels of the variables. In (a) R45 = RAF, R26 = MEK and R39 = ERK .

5.2 Genetic Oscillator

We show how the genetic oscillator [2, 25] can be Modelled and simulated in Matlab SimBiology® exactly as a stochastic process with discrete states, approximated using ODEs that assume continuous states, and then demonstrate statistical model-checking approach using Uppaal SMC.

The synthetic genetic oscillator distills the essence of several real circadian oscillators to demonstrate how nature constructs a reliable system in the face of inherent stochasticity: the oscillator has been shown to exhibit a kind of regularity referred to as stochastic coherence [17]. The model is based on elemental reactions with mass action kinetics shown in Table 3 .

Oscillations arise in the model as a result of a phase shift between competing processes of production and sequestration of protein A. Genes D_A and D_R are transcribed to messenger RNA M_A and M_R, that are trans-

$$\mathsf{A} + \mathsf{D}_A \xrightarrow{\gamma_A} \mathsf{D}'_A \qquad \mathsf{M}_A \xrightarrow{\beta_A} \mathsf{M}_A + \mathsf{A}$$
$$\mathsf{D}'_A \xrightarrow{\theta_A} \mathsf{D}_A + \mathsf{A} \qquad \mathsf{M}_R \xrightarrow{\beta_R} \mathsf{M}_R + \mathsf{R}$$
$$\mathsf{A} + \mathsf{D}_R \xrightarrow{\gamma_R} \mathsf{D}'_R \qquad \mathsf{A} + \mathsf{R} \xrightarrow{\gamma_C} \mathsf{C}$$
$$\mathsf{D}'_R \xrightarrow{\theta_R} \mathsf{D}_R + \mathsf{A} \qquad \mathsf{C} \xrightarrow{\delta_A} \mathsf{R}$$
$$\mathsf{D}'_A \xrightarrow{\alpha'_A} \mathsf{M}_A + \mathsf{D}'_A \qquad \mathsf{A} \xrightarrow{\delta_A} \emptyset$$
$$\mathsf{D}_A \xrightarrow{\alpha_A} \mathsf{M}_A + \mathsf{D}_A \qquad \mathsf{R} \xrightarrow{\delta_R} \emptyset$$
$$\mathsf{D}'_R \xrightarrow{\alpha'_R} \mathsf{M}_R + \mathsf{D}'_R \qquad \mathsf{M}_A \xrightarrow{\delta_{M_A}} \emptyset$$
$$\mathsf{D}_R \xrightarrow{\alpha_R} \mathsf{M}_R + \mathsf{D}_R \qquad \mathsf{M}_R \xrightarrow{\delta_{M_R}} \emptyset$$

$$\alpha_A = 50\text{h}^{-1} \qquad \delta_{M_R} = 0.5\text{h}^{-1}$$
$$\alpha'_A = 500\text{h}^{-1} \qquad \theta_A = 50\text{h}^{-1}$$
$$\alpha_R = 0.01\text{h}^{-1} \qquad \theta_R = 100\text{h}^{-1}$$
$$\alpha'_R = 50\text{h}^{-1} \qquad \mathsf{D}_A = 1\text{m}$$
$$\beta_A = 50\text{h}^{-1} \qquad \mathsf{D}'_A = 0$$
$$\beta_R = 5\text{h}^{-1} \qquad \mathsf{D}_R = 1\text{m}$$
$$\gamma_A = 1\text{m}^{-1}\text{h}^{-1} \qquad \mathsf{D}'_R = 0$$
$$\gamma_R = 1\text{m}^{-1}\text{h}^{-1} \qquad \mathsf{M}_A = 0$$
$$\gamma_C = 2\text{m}^{-1}\text{h}^{-1} \qquad \mathsf{M}_R = 0$$
$$\delta_A = 1\text{h}^{-1} \qquad \mathsf{A} = 0$$
$$\delta_R = 0.2\text{h}^{-1} \qquad \mathsf{R} = 0$$
$$\delta_{M_A} = 10\text{h}^{-1} \qquad \mathsf{C} = 0$$

Table 3. Reactions and initial values for constants and species from the genetic oscillator [25].

lated to proteins A and R, respectively. A and R dimerise to produce complex C. Protein A acts as a promoter for its own gene, creating a positive feedback loop that causes the production of A to increase rapidly. Protein A also promotes the production of protein R. Hence, as A increases, so too does R, but after a delay due to the two step transcription-translation process. As R increases it sequesters A via the second order dimerisation reaction, thus limiting the maximum amount of protein A. The delay in this negative feedback causes the system to oscillate.

MATLAB SimBiology® is designed specifically for biochemical processes, thus modelling is straightforward and simulation benefits from a similar set of solvers with an important addition of the stochastic one. The remarkable feature of SimBiology® is that simulations can be performed in ensembles and the simulation can be accelerated further by compiling into native executable through translation into C, but eventually the behavioural data is recorded as large MATLAB arrays for later post-processing.

We have modelled the genetic oscillator [25] in our previous study [11, 14] and in this paper we compare the results and performance of UPPAAL SMC with MATLAB SimBiology® and Simulink®. Fig. 12 shows the reaction model and simulation plots using a deterministic ODE and a stochastic solver. The simulated behaviour is identical with our previous results and the translated models to UPPAAL SMC through SBML are equivalent to the ones we studied before. We also Modelled the differential equation model in Simulink® and got identical results.

Table 4 summarises the timing measurements of 100h of model time simulations (containing about four periods of oscillations) with graphical plotting turned off. MATLAB family of tools provides a wide range of solvers with varying quality and behaviour. We tried a fixed time step ode1 Euler solver because it is the same method used by the current release of UPPAAL SMC. The results

Tool	Simulation	Performance, s
Simulink®	fixed 10^{-4} step ode1 (Euler)	3.7900 ±0.11
	variable step ode45 (Runge-Kutta)	0.7700 ±0.02
	variable step ode15s (stiff/NDF)	0.0783 ±0.0065
SimBiology®	ode15s (stiff/NDF)	0.1805 ±0.0017
	ssa (stochastic)	0.2575 ±0.0090
	accelerated ode15s (stiff/NDF)	0.0203 ±0.0015
	accelerated ssa (stochastic)	0.2476 ±0.0054
UPPAAL SMC	fixed 10^{-4} step ODE (Euler)	1.7520 ±0.0056
	CTMC (stochastic)	1.0400 ±0.24

Table 4. Genetic oscillator simulation performance using various tools: the intervals computed using 20 measurements with 95% confidence.

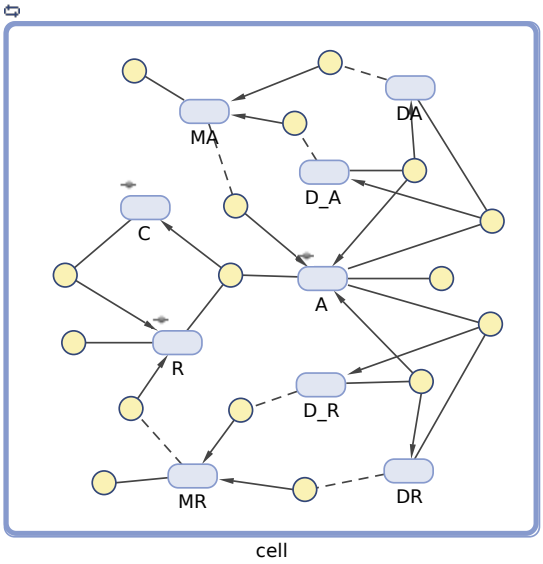
Tool	Simulation	Tuples	Min memory, KB
SimBiology®	ode15s	747	58.4
	ssa	286080 ±7674	22350.0
UPPAAL SMC	ODE	5354	418.3
	stochastic	4437 ±34	346.6

Table 5. Simulation data comparison.

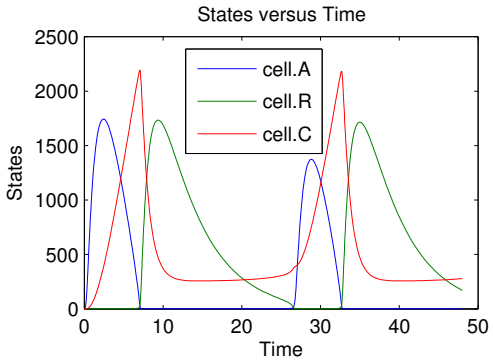
show that UPPAAL SMC implementation is about two times faster. A default choice in Simulink® is a variable step ode45 solver which is more accurate and faster than UPPAAL SMC because it can leap in larger time steps when dynamics change little. Solver ode15s seems to be even a better choice here because it is designed for stiff functions and still fast. The problem with MATLAB Simulink® is that resulting models are large[9] and thus the modelling process is tedious and difficult to debug.

Table 5 shows amounts of data generated for simulation purposes. A tuple of data consists of ten double precision numbers (one for time and nine for species quantities). SimBiology® works very well for deterministic ODE simulations and acceleration can be dramatic, but the stochastic simulations hardly get any benefit from acceleration and can be problematic due to vast amount of generated data while performing small time steps and

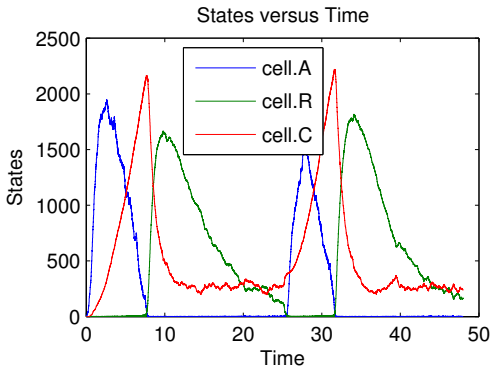
[9] e.g. StateSpace approach is not applicable due to multiple species coupling.



(a) Graphical overview of reactions.



(b) Output of ODE solver.

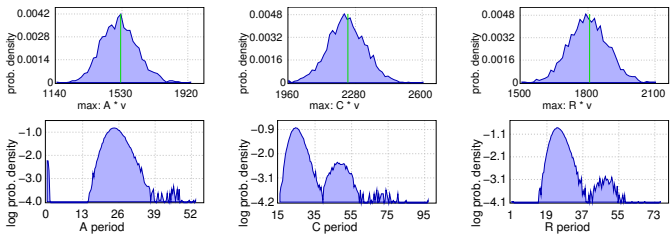


(c) Output of stochastic solver.

Figure 12. Modelling and simulating the genetic oscillator in SimBiology®.

even simple ensemble simulation becomes a memory bottleneck in the process. For example, generating ensemble simulation of just 100 runs (without plots), MATLAB's virtual memory grows from 1329MB to 5633MB (504MB to 4600MB of resident memory). The small time steps are inevitable in stochastic simulations of biological systems because the reaction rates are proportional to molecular quantities and some of them can get very high. While it is possible to use the memory more efficiently by analysing one run at a time, it still requires programming and is still constrained to storing at least one entire run.

UPPAAL SMC contains twice as fast Euler ODE integrator, but it is still much slower than variable step solvers like ode45 and ode15s. On the other hand, stochastic simulation is more than 64 times efficient in space due to data filtering and thus is more suitable for interactive exploration. In addition, UPPAAL SMC provides a query language and evaluates the statistical properties on-the-fly by storing only one state at a time, and thus does not have a storage bottleneck. UPPAAL SMC may also terminate simulation earlier due to feedback from its *Validator* and *Core algorithm* to *Generator* that it has acquired enough information about a single run or observed enough runs.



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

Figure 13. Estimated probability density distributions for amplitude and period.

Next, we demonstrate statistical query language application. For example, the amplitude of each protein quantity can be measured by the following query:

E[<=100; 2000](**max**: Q)

where 100 is the time bound for simulation, 2000 is the number of simulations and Q is a model expression of interest: in our case it is simply one of variables A, C or R. As a result, UPPAAL SMC renders a probability density plot of possible amplitudes and a vertical line for an average value shown in upper plots of Fig. 13. The memory consumption remains about the same: around 40MB of virtual and 3.7MB of resident.

UPPAAL SMC can also estimate a distance between peaks by the monitors WMTL. The idea of the approach

is to translate a WMTL formula that describes the shape of a peak into a monitoring automaton. By resetting a clock x and start a secondary monitor when a peak is detected we can estimate the distance as the value of x when the second monitor detects a peak[14].

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[<=100] (A>1100 && true U[<=5] A<=1000)
```

Then the distance between peaks can be estimated by measuring maximal value of clock x with similar queries we used to estimate amplitude. The result is rendered by the tool as a logarithm of probability density shown in the second row of Fig. 13. 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 the WMTL 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 at all, hence the next one is registered instead.

6 Conclusions

The present paper proves three main points:

1. Uppaal SMC can handle a wide range of biological systems.
2. The simulation engine of Uppaal SMC is of comparable performance with Simulink® and SimBiology®.
3. The essential benefit of Uppaal SMC comes with the power of statistical model checking with respect to a range of temporal logic properties.

In order to show the range of applicability we have connected the Uppaal SMC toolset to ANIMO and SimBiology®– two tools that can be used to specify and analyse biological phenomena. In particular we support SBML, used in many other research projects. Our implementation works via a translation from the input languages of those tools to the one of Uppaal SMC. This approach allows us not only to exploit the efficient ODEs solver of Uppaal SMC, but also to apply powerfull techniques such as Statistical Model Checking to the systems.

As future work, we would like to implement new functionalities to capture a wider range of biological phenomenas. We also plan to improve the ODE solving capabilities of Uppaal SMC by implementing more advanced ODE solvers such as `CVODE` and `ode15s`.

Acknowledgement

The authors of this paper are grateful for the detailed comments from the anonymous reviewers.

Nr.	Res.	Nr.	Res.	Nr.	Res.	Nr.	Res.	Nr.	Res.	Nr.	Res.	Nr.	Res.
001	-	002	x	003	ode	004	ode	005	ctmc	006	*ode*	007	x
008	ode	009	x	010	both	011	both	012	both	013	x	014	both
015	x	016	x	017	x	018	x	019	-	020	x	021	ode
022	ode	023	ode	024	-	025	-	026	ode	027	both	028	both
029	both	030	both	031	ode	032	x	033	x	034	-	035	both
036	ode	037	both	038	x	039	x	040	x	041	ode	042	ode
043	ctmc	044	ctmc	045	ctmc	046	x	047	-	048	x	049	both
050	both	051	-	052	both	053	x	054	ode	055	-	056	x
057	ode	058	ode	059	x	060	ode	061	ctmc	062	ode	063	!
064	ctmc	065	ode	066	ode	067	ode	068	x	069	ode	070	ctmc
071	x	072	x	073	ode	074	ode	075	-	076	both	077	-
078	-	079	ode	080	x	081	-	082	x	083	-	084	ode
085	x	086	x	087	x	088	-	089	x	090	ctmc	091	ctmc
092	both	093	both	094	both	095	-	096	-	097	-	098	ode
099	ode	100	x	101	-	102	ode	103	ode	104	-	105	both
106	ode	107	ode	108	ctmc	109	-	110	ode	111	x	112	-
113	ode	114	ode	115	ode	116	ctmc	117	-	118	x	119	x
120	-	121	-	122	-	123	x	124	-	125	-	126	-
127	-	128	-	129	-	130	-	131	-	132	-	133	-
134	-	135	-	136	-	137	-	138	x	139	-	140	-
141	-	142	-	143	x	144	ode	145	ode	146	ode	147	ode
148	ctmc	149	-	150	x	151	x	152	-	153	-	154	-
155	-	156	ode	157	ode	158	x	159	ode	160	ode	161	-
162	-	163	x	164	-	165	-	166	ode	167	-	168	x
169	ode	170	both	171	-	172	x	173	-	174	-	175	x
176	x	177	x	178	x	179	-	180	-	181	ode	182	x
183	x	184	ode	185	ode	186	-	187	-	188	-	189	-
190	*ode*	191	both	192	x	193	-	194	-	195	x	196	-
197	both	198	ode	199	x	200	x	201	x	202	*ode*	203	x
204	x	205	x	206	ode	207	*ode*	208	-	209	x	210	x
211	ode	212	x	213	x	214	-	215	-	216	ode	217	ode
218	x	219	x	220	x	221	x	222	x	223	x	224	ode
225	ode	226	x	227	-	228	ode	229	ode	230	ode	231	ctmc
232	x	233	ode	234	x	235	-	236	ode	237	-	238	x
239	ode	240	ode	241	-	242	ode	243	both	244	-	245	!
246	x	247	x	248	!	249	ode	250	x	251	x	252	x
253	x	254	*ode*	255	-	256	-	257	ode	258	ode	259	both
260	both	261	both	262	-	263	-	264	-	265	x	266	x
267	both	268	-	269	ode	270	x	271	both	272	x	273	-
274	*ode*	275	ode	276	*ode*	277	*ode*	278	x	279	x	280	x
281	-	282	x	283	both	284	ode	285	x	286	-	287	-
288	x	289	x	290	x	291	x	292	ode	293	both	294	ode
295	-	296	ode	297	x	298	*ode*	299	*ode*	300	x	301	-
302	x	303	x	304	x	305	!	306	x	307	x	308	x
309	both	310	x	311	x	312	-	313	ode	314	ode	315	both
316	-	317	-	318	-	319	ode	320	ode	321	ode	322	ode
323	ode	324	x	325	!	326	-	327	-	328	both	329	*ode*
330	x	331	x	332	ode	333	ode	334	ode	335	x	336	x
337	-	338	-	339	-	340	-	341	*ode*	342	-	343	x
344	x	345	x	346	x	347	x	348	x	349	x	350	-
351	-	352	-	353	*ode*	354	ctmc	355	x	356	-	357	both
358	both	359	ode	360	ode	361	both	362	x	363	ode	364	ode
365	ode	366	ode	367	*ode*	368	x	369	x	370	x	371	!
372	x	373	!	374	!	375	!	376	!	377	!	378	!
379	*ode*	380	x	381	*ode*	382	x	383	x	384	x	385	x
386	x	387	x	388	ode	389	ctmc	390	x	391	x	392	x
393	x	394	ode	395	ode	396	ode	397	both	398	both	399	-
400	x	401	*ode*	402	*ode*	403	*ode*	404	-	405	x	406	x
407	x	408	-	409	ode	410	ode	411	-	412	-	413	both
414	ode	415	x	416	x	417	both	418	ode	419	ode	420	ode
421	ode	422	-	423	x	424	x	425	ode	426	x	427	ode
428	x	429	-	430	both	431	both	432	both	433	both	434	x
435	ode	436	-										

Table 6. Detailed results on all 436 (curated) SBML models from the database available at `http://www.ebi.ac.uk/biomodels-main/publmodels`. The results are reported as follows: "!" means the model was inconsistent, "x" means the simulation failed, "-" means the translation failed, "ctmc" means only the CTMC simulation worked, "ode" means only the ODE simulation worked, and "both" means both simulations worked. Italic "ode" means that the CTMC model could not be generated because of the presence of rate rules in the SBML file (it is not possible to get a CTMC model)

References

1. Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theor. Comput. Sci.*, 126 (2):183–235, 1994. doi: 10.1016/0304-3975(94)90010-8. URL `http://dx.doi.org/10.1016/0304-3975(94)90010-8`.
2. Naama Barkai and Stanislas Leibler. Biological rhythms: Circadian clocks limited by noise. *Nature*, 403:267–268, 2000.
3. Ananda Basu, Saddek Bensalem, Marius Bozga, Benot Caillaud, Benot Delahaye, and Axel Legay. Statistical Abstraction and Model-Checking of Large Heterogeneous Systems. In John Hatcliff and Elena Zucca, editors, *Formal Techniques for Distributed Systems*, volume 6117 of *Lecture Notes in Computer Science*, pages 32–46. Springer Berlin / Heidelberg, 2010. ISBN 978-3-642-13463-0. doi: 10.1007/978-3-642-13464-7_4.

4. G. Behrmann, A. David, and K. G. Larsen. A tutorial on Uppaal. *Lecture Notes in Computer Science*, pages 200–236, 2004.
5. Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. Developing UPPAAL over 15 years. *Softw., Pract. Exper.*, 41 (2):133–142, 2011. doi: 10.1002/spe.1006.
6. Frank T. Bergmann and Herbert M. Sauro. Comparing simulation results of SBML capable simulators. *Bioinformatics*, 24(17):1963–1965, 2008. doi: 10.1093/bioinformatics/btn319. URL `http://bioinformatics.oxfordjournals.org/content/24/17/1963.full`.
7. Peter Bulychev, Alexandre David, Kim G. Larsen, Axel Legay, and Marius Mikučionis. Computing Nash Equilibrium in Wireless Ad Hoc Networks: A Simulation-Based Approach. In Johannes Reich and Bernd Finkbeiner, editors, *Second International Workshop on Interactions, Games and Protocols*, volume 78 of *EPTCS*, pages 1–14, 2012. doi: 10.4204/EPTCS.78.
8. Peter E. Bulychev, Alexandre David, Kim G. Larsen, Axel Legay, Guangyuan Li, and Danny Bøgsted Poulsen. Rewrite-Based Statistical Model Checking of WMTL. In Shaz Qadeer and Serdar Tasiran, editors, *RV*, volume 7687 of *Lecture Notes in Computer Science*, pages 260–275. Springer, 2012. ISBN 978-3-642-35631-5, 978-3-642-35632-2. doi: 10.1007/978-3-642-35632-2_25.
9. Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Axel Legay, Guangyuan Li, Danny Bøgsted Poulsen, and Amélie Stainer. Monitor-Based Statistical Model Checking for Weighted Metric Temporal Logic. In Nikolaj Bjørner and Andrei Voronkov, editors, *LPAR*, volume 7180 of *Lecture Notes in Computer Science*, pages 168–182. Springer, 2012. ISBN 978-3-642-28716-9. doi: 10.1007/978-3-642-28717-6_15.
10. Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Jonas van Vliet, and Zheng Wang. Statistical Model Checking for Networks of Priced Timed Automata. In Uli Fahrenberg and Stavros Tripakis, editors, *FORMATS*, volume 6919 of *Lecture Notes in Computer Science*, pages 80–96. Springer, 2011. ISBN 978-3-642-24309-7. doi: 10.1007/978-3-642-24310-3_7.
11. Alexandre David, Dehui Du, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical Model Checking for Stochastic Hybrid Systems. In Ezio Bartocci and Luca Bortolussi, editors, *HSB*, volume 92 of *EPTCS*, pages 122–136, 2012. doi: 10.4204/EPTCS.92.9.
12. Alexandre David, DeHui Du, Kim G. Larsen, Marius Mikučionis, and Arne Skou. An evaluation framework for energy aware buildings using statistical model checking. *Science China Information Sciences*, 55:2694–2707, 2012. ISSN 1674-733X. doi: 10.1007/s11432-012-4742-0. URL `http://dx.doi.org/10.1007/s11432-012-4742-0`.
13. Alexandre David, Kim Guldstrand Larsen, Axel Legay, and Marius Mikučionis. Schedulability of Herschel-Planck Revisited Using Statistical Model Checking. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA (2)*, volume 7610 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2012. ISBN 978-3-642-34031-4. doi: 10.1007/978-3-642-34032-1_28.
14. Alexandre David, Kim Guldstrand Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Runtime Verification of Biological Systems. In *ISoLA (1)*, pages 388–404, 2012. doi: 10.1007/978-3-642-34026-0_29.
15. James R. Faeder, Michael L. Blinov, and William S. Hlavacek. Rule-Based Modeling of Biochemical Systems with BioNetGen. *Systems Biology*, 500, January 2009. doi: 10.1007/978-1-59745-525-1_5.
16. D. T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry*, 81:2340–2361, 1977. doi: 10.1021/j100540a008.
17. Robert C. Hilborn and Jessie D. Erwin. Stochastic coherence in an oscillatory gene circuit model. *Journal of Theoretical Biology*, 253(2):349 – 354, 2008. ISSN 0022-5193. doi: 10.1016/j.jtbi.2008.03.012. URL `http://www.sciencedirect.com/science/article/pii/S0022519308001264`.
18. Cyrille Jegourel, Axel Legay, and Sean Sedwards. A Platform for High Performance Statistical Model Checking – PLASMA. In Cormac Flanagan and Barbara König, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *Lecture Notes in Computer Science*, pages 498–503. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-28755-8. doi: 10.1007/978-3-642-28756-5_37.
19. Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a Nutshell. *STTT*, 1(1-2):134–152, 1997. doi: 10.1007/s100090050010.
20. Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical Model Checking: An Overview. In *RV*, volume 6418 of *Lecture Notes in Computer Science*, pages 122–135. Springer, 2010. doi: 10.1007/978-3-642-16612-9_11.
21. S. Schivo, J. Scholma, B. Wanders, R.A. Urquidi Camacho, P.E. van der Vet, M. Karperien, R. Langerak, J. van de Pol, and J.N Post. Modelling biological pathway dynamics with Timed Automata. *Biomedical and Health Informatics, IEEE Journal of*, PP(99):1–1, 2013. ISSN 2168-2194. doi: 10.1109/JBHI.2013.2292880.
22. Stefano Schivo, Jetse Scholma, Brend Wanders, Ricardo A. Urquidi Camacho, Paul E. van der Vet, Marcel Karperien, Rom Langerak, Jaco van de Pol, and Janine N Post. Modelling biological pathway dynamics with Timed Automata. In *Proceedings of the 2012 IEEE 12th International Conference on Bioin-*

formatics & Bioengineering (BIBE), pages 447–453, 2012.

23. Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical Model Checking of Black-Box Probabilistic Systems. In *CAV*, LNCS 3114, pages 202–215. Springer, 2004. doi: 10.1007/978-3-540-27813-9_16.
24. Paul Shannon, Andrew Markiel, Owen Ozier, Nitin S Baliga, Jonathan T Wang, Daniel Ramage, Nada Amin, Benno Schwikowski, and Trey Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome research*, 13(11):2498–2504, 2003. doi: 10.1101/gr.1239303.
25. José M. G. Vilar, Hao Yuan Kueh, Naama Barkai, and Stanislas Leibler. Mechanisms of noise-resistance in genetic oscillators. *Proceedings of the National Academy of Sciences*, 99(9):5988–5992, 2002. doi: 10.1073/pnas.092133899. URL `http://www.pnas.org/content/99/9/5988.abstract`.
26. Håkan L. S. Younes and Reid G. Simmons. Probabilistic Verification of Discrete Event Systems Using Acceptance Sampling. In *Proc. of 14th Int. Conference on Computer Aided Verification (CAV)*, LNCS 2404, pages 223–235. Springer, 2002.
27. Håkan L.S. Younes. Ymer: A Statistical Model Checker. In Kousha Etessami and Sriram K. Rajamani, editors, *Computer Aided Verification*, volume 3576 of *Lecture Notes in Computer Science*, pages 429–433. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-27231-1. doi: 10.1007/11513988_43.