Improving the Timed Automata approach to biological pathway dynamics

Stefano Schivo[*], Jetse Scholma[*], Paul E. van der Vet, Marcel Karperien, Janine N. Post, Jaco van de Pol[**], and Rom Langerak[**]

University of Twente, Enschede, The Netherlands

Abstract. Biological systems such as regulatory or gene networks can be seen as a particular type of distributed systems, and for this reason they can be modeled within the Timed Automata paradigm, which was developed in the computer science context. However, tools designed to model distributed systems often require a computer science background, making their use less attractive for biologists. ANIMO (Analysis of Networks with Interactive MOdeling) was built with the aim to provide biologists with access to the powerful modeling formalism of Timed Automata in a user friendly way. Continuous dynamics is handled by discrete approximations.

In this paper we introduce an improved modeling approach that allows us to considerably increase ANIMO's performances, opening the way for the analysis of bigger models. Moreover, this improvement makes the introduction of model checking in ANIMO a realistic feature, allowing for reduced computation times. The user interface of ANIMO allows to rapidly build non-trivial models and check them against properties formulated in a human-readable language, making modeling a powerful support for biological research.

1 Introduction

To understand the possible causes of a disease and design effective cures it is necessary to closely study the behavior exhibited by biological cells under particular conditions. A *signaling pathway* describes the chain of interactions occurring between the reception of a *signal* and the response with which the cell reacts to such signal. A signal is typically represented by a substance which can bind to specific receptors on the cell surface, activating them. *Active* molecules relay the signal inside the cell by activating other molecules until a target is reached. The target of a signaling pathway is usually a transcription factor, a molecule with the task of controlling the production of some protein. Such regulation is considered to be the response of the cell to the received signal.

The current knowledge on signaling pathways (mostly organized in databases such as KEGG [1] or PhosphoSite [2]) suggests that the interactions involved in a

[*] These authors contributed equally to this work.
[**] Corresponding authors.

cellular response assume more often the shape of a network than that of a simple chain of signal relays. Such networks are typically highly connected, involving feedback loops and crosstalk between multiple pathways, making it difficult to grasp their dynamic behavior. For this reason, computational support is required when studying non-trivial biological networks.

A number of software tools are available for modeling complex networks of biochemical interactions [3–7]. These tools significantly contribute to the process of formalizing the knowledge on biological processes, rendering them amenable to computational analysis. However, a lack of familiarity with the formalisms underlying many available tools hampers their direct application by biology experts. ANIMO (Analysis of Networks with Interactive MOdeling, [8–10]) is a software tool based on the formalism of Timed Automata [11] that supports the modeling of biological signaling pathways by adding a dynamic component to traditional static representations of signaling networks. ANIMO allows to compare the behavior of a model with wet-lab data, and to explore such behavior in a user-friendly way. In order to achieve a good level of user-friendliness for a public of biologists, the complexity of Timed Automata is hidden *under the hood*, presenting ANIMO as an app for Cytoscape [12], a tool specifically developed for visualizing and elaborating biological networks.

Previously, ANIMO supported only interactive exploration of network dynamics based on simulation runs. Model checking queries could be answered through the UPPAAL tool [13], but the required knowledge of temporal logic together with the usually long response times slowed down the investigation process. In order to encourage the use of model checking on non-trivial models of signaling networks, we updated ANIMO with a new Timed Automata model (presented here). This marks a relevant improvement in terms of performances with respect to the model previously used in ANIMO. Moreover, consistently with the intents of our tool, we implemented also a user interface for the definition of model checking queries in ANIMO. This allows a user to interrogate an ANIMO model without requiring previous experience in temporal logics. These new features are available in the latest version of ANIMO, which was recently reimplemented as a Cytoscape 3 app [14]: this lets users profit from the additional analysis features made available by the other apps in the Cytoscape environment.

The paper continues as follows. After introducing the basic concepts in Section 2, we illustrate in Section 3 a new way of using Timed Automata in ANIMO. In Section 4 we present a comparison between the new modeling approach and the one previously used in ANIMO, focusing on model analysis performances. In Section 5 we describe how model checking was made accessible to ANIMO users. We conclude the paper with Section 6, discussing future work.

2 Preliminaries

2.1 Signaling pathways in biology

A signaling pathway is an abstract representation of the reactions occurring inside a biological cell when, e.g., a signaling substance comes in contact with the cell surface receptors. In this setting, a reaction is the interaction between two components: the upstream enzyme (the molecule holding the active role in the reaction) and the downstream substrate (the passive molecule). The enzyme can be for example a kinase, which attaches a phosphate group to its substrate, performing a phosphorylation: this determines a change in the shape of the substrate and consequently a new function (Fig. 1). The new state reached by the substrate is often called *active*: if the substrate of the reaction is itself a kinase, it can then proceed in passing on the signal by activating its own target molecule, continuing a chain of reactions leading to the target of the signaling chain. Such target is usually a transcription factor, i.e. a molecule that influences the genetic response of the cell, for example promoting the production of a particular protein.

Pathways are traditionally represented in a nodes-edges form (see Fig. 1a), with nodes representing molecular species and edges standing for reactions, where $\rightarrow$ represents activation and $\dashv$ represents inhibition (i.e. inactivation).

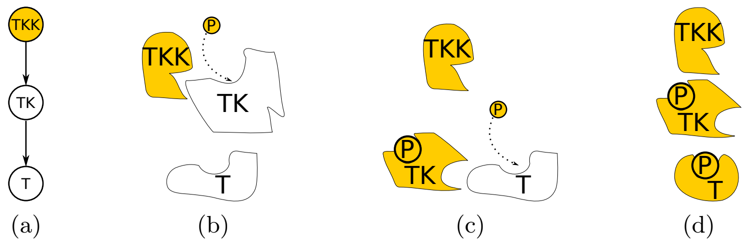


Fig. 1. A kinase signaling network represented in the nodes-edges form **(a)** and its evolution represented by abstract molecular interactions (**b-d**). T is the target of the signaling pathway, TK is the kinase that activates T, and TKK is the kinase that activates TK. **(b)** An already active (yellow) TKK can bind to an inactive (empty) TK, catalyzing its phosphorylation (i.e. binding of a phosphate group, P). This causes a change in the shape of TK, activating its enzymatic function **(c)**. The active TK can in turn activate its target T, enabling it to carry out its function **(d)**.

The current knowledge on signaling pathways [1, 2] evidences the fact that signaling interactions are rarely a simple chain of activations as represented in Figure 1a. More often, they assume the shape of a network with multiple feedback loops and crosstalk from different signaling sources. This complexity is an added difficulty for the study of such networks, reducing the possibilities to deduce the dynamic behavior of a signaling network by inspecting its static representation. For this reason, efficient computational support is essential when representing and analyzing the behavior of complex signaling networks.

2.2 Timed Automata

Timed Automata (TA) are finite-state automata enriched with real-valued clocks and synchronization channels. All clocks in a TA system advance with the same rate, and transitions between the locations of an automaton depend on conditions on clocks. In particular, a *guard* defines when a transition may be taken, while an *invariant* is the condition for permanence in a location. A transition can also allow two automata to *synchronize*, with each participant performing one of two complementary actions (*input* and *output*) on a synchronization channel. A set of clocks may also be reset by a transition, causing them to restart from 0.

The models we will present here were implemented using the software tool UPPAAL [13], which adds a number of features to the basic definition of TA. Some of these extensions include: support for integer variables in addition to clocks, broadcast synchronization channels (one sender, many receivers), definition of C-like functions to perform more operations besides clock resets. UPPAAL also allows for a special type of locations, named *committed* (marked with a C in the graphical representation). As long as an automaton is in a committed location, time is not allowed to flow. This feature can be used for example to perform immediate updates to local variables before letting the computation proceed. Examples of the listed features are found in the TA model in Section 3.2.

2.3 Activity-based models in ANIMO

ANIMO allows the definition of *activity-based* models. This means that we assume each signaling molecule in a cell to be at any time in one of two states: active or inactive. Active molecules can take an active role in reactions, changing the state of other molecules, activating inactive molecules or inhibiting (i.e. deactivating) active molecules. In a kinase-based signaling network an activation process can be a phosphorylation, and it is usually countered by the corresponding dephosphorylation. However, our models are not limited to kinase networks: other features like different post-translational modifications or gene promotion can be likewise represented, as long as their role has immediate effects on the ability of a target to perform its task.

As ANIMO is a Cytoscape app, models are defined through the Cytoscape user interface (see Fig. 2), where the user inserts a node for each molecular species and an edge for each reaction, with $\rightarrow$ indicating activation and $\dashv$ indicating inhibition similarly to traditional representations of signaling networks (Fig. 1a).

We consider a *molecular species* (also called *reactant*) to include all the molecules of the same substance in both their active and inactive state inside the cell. In order to distinguish between the two activity states in which each molecule can be, we define the *activity level* to represent the percentage of active molecules over an entire molecular species. In an ANIMO model, this value is discretized on a given integer interval. The user can choose the granularity for each molecular species separately, on a scale between 2 (the reactant is seen as either completely inactive or completely active) and 101 levels (allowing to represent activity as $0, 1\%, 2\% \ldots\ 100\%$). The activity level of a molecular species is

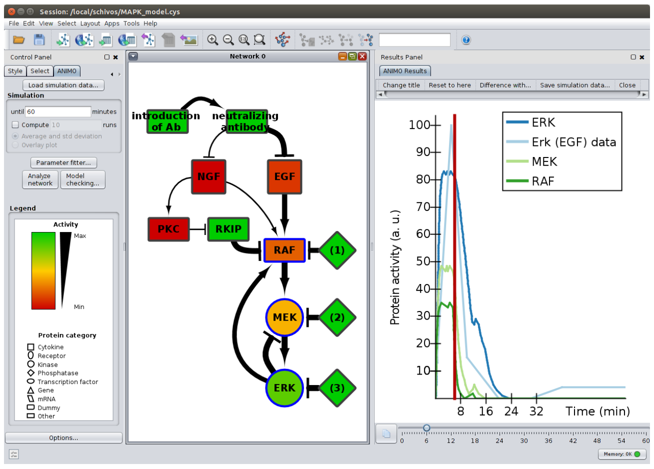


Fig. 2. The Cytoscape 3 user interface running the new ANIMO app (model from [9]). The *Network* panel in the center contains the nodes-edges model of the cross-talking signaling pathways of NGF and EGF, with colors indicating node activity levels and shapes representing different protein categories (see the *Legend* on the left). The *Results Panel* on the right contains a graph plotting activity levels of selected nodes during the first hour of evolution of the model. The slider under the graph allows the user to select the time instant (marked as a vertical red line in the graph) on which the colors of the nodes in the *Network* are based. The edge thickness is used to give an idea of the reactions' speed at the selected time instant. The series Erk (EGF) data in the graph is the experimental data from [15] for the 100 ng/ml EGF treatment.

represented in the ANIMO network by coloring the corresponding node according to the scale shown in the Activity legend in Figure 2, where the minimum indicates that all molecules of the given species are inactive.

The occurrence of a *reaction* modifies by one discrete step the activity level of its target reactant, making it increase or decrease depending on whether the reaction is defined, respectively, as activating or inhibiting. The rate with which a reaction occurs depends on a formula selected by the user. Choosing one of three available scenarios allows the user to make the reaction rate depend on the activity of one or two reactants. The rate of each reaction can be scaled by modifying the value of one kinetic constant k, possibly using a qualitative measure from a predefined set (very slow, slow, medium, fast, very fast). The approximation allows us to reduce the dependence of a model from often unavailable quantitative parameters for biochemical reaction kinetics, while keeping a precision level that is still high enough to be useful. For a more precise explanation on how reaction rates are computed in ANIMO, we recommend [9], where the previously used TA model is presented. The reader interested in the current methods for parameter setting in ANIMO can refer to [16].

3 A new way of modeling signaling pathways with TA

We present here a novel model to represent signaling pathways with TA in ANIMO. We define the model previously used in ANIMO to be *reaction-centered*, as for each reaction in the network an instance of a TA template is generated to mimic the occurrences of that reaction. Observing that signaling networks tend to be highly connected, containing noticeably more reactions than reactants, we shift the focus on reactants instead, achieving what we call a *reactant-centered* model. This change of view is inspired by the classical way in which biological events are modeled with ordinary differential equations (ODEs) [17].

3.1 The reactant-centered approach

The reactant-centered model presented here is based on the concept of *net effect* of a set of reactions on a reactant: instead of considering each reaction in isolation, we consider their combined influence on each reactant. As an example, consider a reactant A activated by reactions R_1 and R_2, and inhibited by reaction R_3 (see Fig. 3a). The net effect of these three reactions on A defines the *net reaction* $R_A = R_1 + R_2 - R_3$. Applying a concept similar to the definition of an ODE, where the rate of change of each reactant depends on the rate of the reactions influencing it, the rate of R_A is computed as the sum of the rates of the reactions influencing A:

$$r_A = r_1 + r_2 - r_3$$

where r_i is the rate of reaction R_i and is defined as follows. Consider R_1 to be the reaction $B \rightarrow A$ with kinetic constant k_1. Suppose the settings in the ANIMO network for R_1 make its rate depend only on the activity level of B. Then we compute the rate of R_1 as $r_1 = [B] \times k_1$, with $[B]$ the current activity level of B.

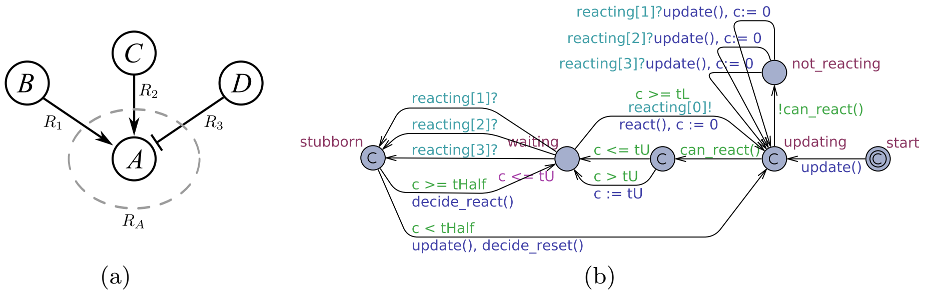


Fig. 3. **(a)** A signaling network where one node is influenced by three distinct reactions. The (virtual) reaction R_A is defined in the reactant-centered model as the algebraic sum of the reactions influencing A. (**b**) TA template generated by ANIMO for the reactant-centered model of the network in **(a)**. The template has been edited to increase readability. All the functions used in **(b)** are described in Section 3.2 and can be found in Appendix A.

If r_A is positive the activity level of A will increase; otherwise, A will decrease its activity level. The absolute value of r_A determines the speed with which such change happens. The value of the reaction rate is thus translated into a time value to be used as time bound in the TA representing R_A (see Fig. 3b) by computing

$$T_A = \frac{1}{\text{abs}(r_A)}$$

with abs(r_A) the absolute value of r_A. In order to represent a natural uncertainty or variability in reaction timing, we relax the exact value of T_A by defining bounds of $\pm$ 5% (which can be changed by the user):

$$\mathsf{tL} = T_A \times 0.95 \qquad \mathsf{tU} = T_A \times 1.05$$

Analogously to the reaction-centered model of [9], we will call tL the *lower time bound* and tU the *upper time bound*. So we replace an exact reaction time by an interval of possible reaction times, implicitly assuming a uniform distribution over this interval.

3.2 The new Timed Automata model

The TA model we propose uses one TA template to represent each reactant: in Figure 3b we show the automaton that models how the activity level of reactant A is changed by the (virtual) reaction R_A. Note that automata for B, C and D are not needed in our example, as no reactions influence them. We will now explain how the TA template for R_A works and how its discrete behavior approximates a continuous model.

The proposed TA template fulfils two main tasks:

- perform the reaction, changing the activity level of the modelled reactant A
- update the time bounds tL and tU in response to changes in the network

The occurrence of the reaction is modelled with the central location waiting, where the automaton waits for the internal clock c to reach the time interval [tL, tU]; when the condition $\mathsf{tL} \leq c \leq \mathsf{tU}$ is verified, the reaction can occur. This takes the automaton to location updating and simultaneously changes the activity level of A (react() in the transition). Together with all the remaining locations, updating is used to update the time bounds tL and tU to reflect the new state of the network and to determine how the reaction R_A will continue.

Note that all locations apart from waiting and not_reacting are declared as committed (C inside the location's circle): we use them to perform *instant* updates to the automaton and its variables. In UPPAAL, no clock advances as long as an automaton is in a committed location, and transitions exiting from committed locations have the precedence over other transitions.

Location updating: decide whether *A* can change. All transitions entering location updating call the function update() (a call to update() is performed inside react() before returning), which performs the computations described in Section 3.1 and thus determines the new values of r_A, tL and tU. At this point, one of the following conditions is true:

1. the newly computed reaction rate r_A is 0
2. A has reached its maximum activity level and $r_A > 0$
3. A has reached the activity level 0 and $r_A < 0$
4. the boundaries for A's activity level have not been reached and $r_A \neq 0$

In the first three cases, the function can_react() will return false, meaning that the activity level of A cannot be updated further in the current conditions. This enables the transition to not_reacting, as the guard !can_react() is true.

In the case 4, the reaction R_A can still occur (can_react() evaluates to true), so the automaton returns to the waiting location to perform a new step of the reaction. The passage through the interposed committed location allows us to ensure that the invariant $\mathsf{c} \leq \mathsf{tU}$ of location waiting is always respected. In fact, it is possible that c has not been reset upon entering location updating (transition from stubborn, explained later on).

Channel reacting[]: adapt to changes in the environment. When the reaction R_A occurs, A is updated to its new activity level (call to react() when exiting from waiting) and a synchronization is performed, using reacting[0]! to indicate that the activity level of A has changed. reacting[] is an array of broadcast channels, each one associated to a reactant in the network: in the example network, the channel with index 0 corresponds to A, 1 to B, and so on. The reception of a communication along one of those channels indicates that the corresponding reactant has changed its activity level. The automaton representing R_A can receive communications on the channels associated to the three reactants influencing A (see the transitions exiting from not_reacting and those entering stubborn). Note that in UPPAAL synchronizations along broadcast channels allow output (!) actions to be performed even if no receiver is present. However, whenever an automaton performs an output action, any other automaton currently in a location where the corresponding input (?) action is enabled will necessarily synchronize with the sender. In our example, this means that A is able to perform the output action reacting[0]! even if no other automaton is present in the network, and would be able to react to any changes in B, C or D if automata representing those reactants were added to the model.

Location stubborn: reduce the approximation error. The transitions entering location stubborn after a synchronization on a reacting[] channel allow to respond to a change in the environment while R_A is occurring. In this event, one of the following conditions is true:

1. at least half of the current reaction step has been performed
2. less than half of the current reaction step has been performed

In case 1 ($\mathsf{c} \geq \mathsf{tHalf} = \frac{T_A}{2}$, transition to waiting), the current reaction step will be completed immediately if the effect of the changes occurred in the reactants influencing A is *dramatic*. We define a change to be dramatic if it causes the new value of r_A to be at least twice the old value, or if it changes the direction of the reaction (r_A changes sign). The comparison between the possible values of r_A, together with the actions to immediately enable[1] R_A, are taken in the function decide_react(). From this behavior comes the name of location stubborn.

[1] In practice, we set $\mathsf{tL} = \mathsf{tU} = \mathsf{c} = 0$.

In case 2 ($\mathsf{c} < \mathsf{tHalf}$, transition to updating), the new status of the system is immediately acknowledged (call to function update()) without performing the reaction first. Here, a decision is taken on whether to reset clock c, i.e. whether to throw away the "job already done" or not. Again, the decision depends on the change: a dramatic change causes clock c to be reset, while a non-dramatic change implies that the work will proceed at a slightly different speed instead of being restart. This allows to avoid starvation for R_A in case of a series of changes with minor effects. Note that in some conditions we could have $\mathsf{c} > \mathsf{tU}$: this means that with the new configuration the reaction should already have occurred. Thanks to the committed location leading to location waiting we make sure that the reaction occurs as soon as possible without violating the invariant.

Locations not_reacting and start. While the automaton cannot perform any change to A's activity level, it waits in location not_reacting for changes in the ractants from which A depends. In the event of any such changes, location updating is reached to check on the feasibility of R_A.

Location start is used to perform the first step of the model, which is to initialize the parameters of the automaton.

3.3 Precision level of the model

Thanks to the features described in Section 3.2, the precision of the TA model for R_A is linked to the granularity of reactant A: the larger number of discrete steps are used to represent A's activity level, the shorter each reaction step involving A will be. This reduces the error made when taking decisions in location stubborn: that error is always less than half a granularity step (cf. the use of tHalf). Moreover, we can show that having a series of *dramatic* changes in the environment will not increase the error, which is limited by $1 - \frac{1}{2^d}$, with d the number of consecutive dramatic changes occurring before R_A can complete the current reaction step.

3.4 Computations on the fly

Reaction rates are all computed at run-time by the function update(), and are based on the user-chosen scenarios, their kinetic constant k and the activity level of the involved reactants. As such computations require floating-point precision but UPPAAL only provides integer variables and operators, we use a significand-and-exponent notation with 4 significant figures, which allows for an error in the order of 0.1% while avoiding integer overflow in UPPAAL's engine. For example, the floating point number $a = 1.23456$ will be represented as the pair $\langle 1235, -3 \rangle$, which is translated back as $a = 1235 \times 10^{-3} = 1.235$. The interested reader can find the UPPAAL definitions and functions needed to compute rate and time values for the TA templates, together with all other functions such as update()

and react(), in Appendix A or inside any UPPAAL model file generated by ANIMO with a reactant-centered model type[2].

4 Reaction-centered VS reactant-centered

We will now apply some basic model checking queries to the case study presented in [9], measuring the performances of the two modeling approaches. This will allow us to evaluate the benefit brought by the shift in perspective from a reaction- to a reactant-centered model.

All experiments were carried out on an Intel® Core™ i7 CPU at 2.80GHz equipped with 4 Gb RAM and running Ubuntu GNU/Linux 14.04.1 64bit. UPPAAL version 4.1.19 64bit was used to compute the result of the queries, asking for "some trace" with random depth-first search order when an execution trace was expected to be produced. For the simulation queries using the statistical model checking engine, we left all options at their default values.

The case study we use as a testbed is the network model shown in the *Network* panel in Figure 2, which represents signaling events downstream of growth factors EGF (epidermal growth factor) and NGF (nerve growth factor) in PC12 cells (a cell line used to study neuronal differentiation). The model topology proposed in [15] was analyzed with an ANIMO model based on the reaction-centered approach, reproducing the experimentally observed ERK (extracellular signal-regulated kinase) activity changes [9].[3] In particular, a 10 minutes stimulation with EGF resulted in transient behavior (i.e. peak-shaped, see also the graph in Fig. 2), while NGF stimulation led to sustained activity (data not shown).

4.1 Simulation cost

We start by evaluating the cost of simulation with the two different models. This is a particularly important aspect to consider, as during the model building phase a user may need to perform a large number of simulations, continuously adapting the topology or quantitative parameters of a network model. In order to make the modeling approach in ANIMO as interactive as possible, it is desirable to decrease waiting times, and this translates into reducing the computational cost of model analysis as much as possible. UPPAAL's statistical model checking engine [18] is based on the generation of random walks (i.e., simulation runs) from a TA model. In this experiment, we query UPPAAL for simulation runs on the models generated by ANIMO applying the reaction- and reactant-centered modeling approaches to the case study. To define the initial state of the model, we consider the starting condition to be the treatment with 50 ng/ml NGF, which translates into setting the activity level of node NGF to 15/15, while changing EGF to be at 0/15 activity. This configuration was chosen as it generates a more

[2] Models generated by ANIMO are saved in the system's temporary directory. Further details are available in the ANIMO manual at `http://fmt.cs.utwente.nl/tools/animo/content/Manual.pdf`

[3] Model available at `http://fmt.cs.utwente.nl/tools/animo/models.html`

interesting behavior from the biological point of view, also w.r.t. the model checking queries in Section 4.2; the treatment with 100 ng/ml EGF was also tested and gave similar performance results. Table 1 illustrates the computation time and memory usage when performing 100 simulation runs on each of the two considered models. Computing the simulation runs took about 91% less time with the reactant-centered model, using 97% less memory. This decrease in computation time for long simulation runs brings the approach nearer to the idea of interactive exploration of a network.

Model type	Time (s)	Memory (peak KB)
Reaction-centered	30.72	291 576
Reactant-centered	2.86	9 768

Table 1. UPPAAL processor time and memory usage for reaction- and reactant-centered modeling approaches when computing the query `simulate 100 [36000] { R1, R2, ..., R11 }` on the model from [9] with starting condition NGF = 15/15, EGF = 0/15, corresponding to a treatment with 50 ng/ml NGF. The query asks for 100 time series of the activity levels of all reactants in the model over the first 60 minutes of execution.

The scalability of the reactant-centered model was further tested performing 100 simulation runs on a much larger network, consisting of 90 nodes and 283 edges [19]. The network represents the main signaling and transcription events involved in osteoarthritis in human chondrocytes (cells involved in the production and maintainance of cartilage). In the test, we analyzed a particularly complex scenario, which models a possible path from healthy to osteoarthritic chondrocyte. Using the reactant-centered approach required 757.14 seconds of CPU time and 128 840 KBs of memory, while the analysis of the reaction-centered model took seconds and used KBs of memory. The resulting states were the same for the two model types, only showing minor ($< 5\%$) differences in activity along the simulation traces.

4.2 Model checking performances

Next, we set out to test the model checking performances on the two versions of the TA model, comparing the execution times and memory requirements for a number of interesting queries:

- (1) and (2): `A[] not deadlock`. The model continues to execute indefinitely (`[]` refers to all possible paths in the transition system of the model, and `A` asks the property to always hold along a path).
- (3): `RKIP < 10 --> ERK >= 40`. After RKIP (Raf kinase inhibitory protein) activity has been lowered, ERK activity increases. As in the model RKIP has 20 levels of granularity and ERK has 100 levels, `RKIP < 10` means that RKIP is less than half active, and `ERK >= 40` means that ERK activity is at least 40%.

– (4): `E<> RKIP < 10`. Find a point when RKIP is low (`<>` asks for the existence of at least one path for which the property holds, while `E` requires the property to hold at least once in a given path). This query is expected to generate a trace, the last point of which will be used as initial configuration for model checking queries (5) and (6).
– (5): `A[] ERK < 70` and (6): `A[] ERK > 35`. Once RKIP activity has significantly decreased, ERK activity is sustained at an intermediate level.

The initial conditions are:

– (1): EGF = 15/15 and NGF = 0/15, all others as the original configuration, corresponding to the treatment condition with 100 ng/ml EGF.
– (2) - (4): EGF = 0/15, NGF = 15/15, all others as the original configuration, corresponding to the treatment condition with 50 ng/ml NGF[4].
– (5) and (6): all activities as in the last state of the trace computed from query (4).

We note that performing model checking means dealing with state space explosion problems, and model reduction is recommendable in order to obtain any result within adequate time limits. One of the most user-accessible ways available in ANIMO to reduce the size of a model is setting the "natural uncertainty" defined in Sect. 3.1 to 0 before performing model checking queries. In order to still consider some biological variability, the user can manually perform multiple model checking queries with changed interaction parameters. The tests performed in this section have an uncertainty level set to 0, instead of the 5% recommended for simulation-based experiments, to make model checking feasible within seconds.

Query	Reaction-centered		Reactant-centered		Improvement	
	Computation time (s)	Memory usage (peak KB)	Computation time (s)	Memory usage (peak KB)	Time (n-fold)	Memory (n-fold)
(1)	126.56	523 448	1.04	9 236	122	57
(2)	159.29	436 496	1.73	11 480	92	38
(3)	146.09	439 484	1.04	10 384	140	42
(4)	0.74	293 508	0.06	7 484	12	39
(5)	581.79	448 764	6.86	16 880	85	27
(6)	561.01	449 248	6.42	15 852	87	28

Table 2. UPPAAL processor time and memory usage for reaction- and reactant-centered modeling approaches when computing the given queries on the case study from [9].

The queries were used with both the reaction- and reactant-centered TA models, and returned the same results as expected: in particular, query (1) returned false and all other queries returned true. From the biological point of

[4] In the laboratory experimental setting, NGF is used at a lower concentration than EGF, but it is still enough to saturate all NGF receptors, which are rarer than EGF receptors.

view, the answer to query (1) confirms that under EGF treatment no other activity is observed in the model after the initial peak, while query (2) confirms that with NGF activity continues indefinitely. Moreover, queries (3) - (6) confirm the result of the simulations shown in [9], with NGF treatment leading to sustained ERK activity.

The results of the model checking performance test are shown in Table 2.

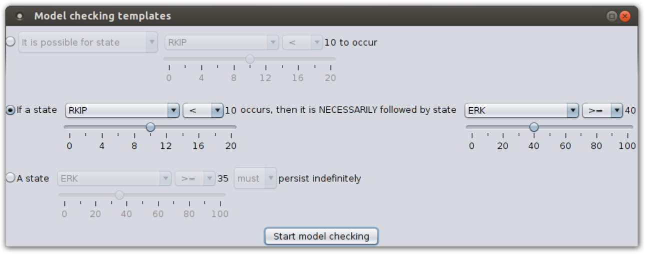
4.3 Analysis of the results

Requesting a full inspection of the state space as we do when using a query of the type `A[]`ϕ returning true in cases (5) and (6), allows us to indirectly compare the state space size of the two model versions. As the computation time improvements in Table 2 show, the reactant-centered model produces indeed a noticeably smaller state space, allowing for a higher level of interactivity also when performing non-trivial model checking. Moreover, our experiments point out that the reactant-centered approach considerably lowers the memory requirements for the model. This is not only due to the absence of possibly large precomputed time tables, which can contain thousands of elements each. Indeed, this point was further investigated by implementing a reaction-centered model which avoids the use of tables and instead makes on-the-fly computations of the time bounds with the same number representation as in the reactant-centered model. This resulted in improved performances in the cases of reachability and simulation-based queries, with memory requirements closer to the ones for the reactant-centered model (0.69 s and 24 796 KB for query (4)). However, in all other cases a much larger amount of memory (around 2 Gb) was used with respect to the table-based implementation of the same model, without leading to appreciable benefits in terms of execution time: in some cases performances were noticeably deteriorated (600-800 s for queries (1)-(3)). These findings seem to support the idea that reactant-centered models have a smaller state space.

5 Model checking in ANIMO

In order to allow a non-expert user to profit from the power of model checking, we have implemented a template-based user interface to define queries directly inside the ANIMO Cytoscape App: Figure 4 shows the interface for composing a model checking query in ANIMO. The mappings between user interface templates and actual model checking queries were inspired by the ones proposed in [20], and are shown in Table 3.

If the answer to a model checking query contains a (counter-) example trace, the trace is automatically parsed by ANIMO and presented to the user in form of a graph of activity levels, in the same fashion as is normally done with simulation runs. Finally, a button positioned near the time slider under a simulation graph allows the user to easily change the initial activity levels of the whole network by setting them as in the currently selected time instant. This feature was used after executing query (4) to set the initial conditions for queries (5)-(6). Such an



( ) It is possible for state RKIP < 10 to occur

(●) If a state RKIP < 10 occurs, then it is NECESSARILY followed by state ERK >= 40

( ) A state ERK >= 35 must persist indefinitely

Fig. 4. The interface used in ANIMO to compose a model checking query. The settings on the three lines correspond, from top to bottom, to queries (4), (3) and (6).

ANIMO template	UPPAAL formula
It is possible for state ϕ to occur	`E<>` ϕ
State ϕ never occurs	`E<> !(`ϕ`)`
If a state ϕ occurs, then it is necessarily followed by a state ψ	ϕ `-->` ψ
A state ϕ can persist indefinitely	`E[]` ϕ
A state ϕ must persist indefinitely	`A[]` ϕ

Table 3. Mapping between queries as presented in ANIMO user interface and the corresponding model checking queries in UPPAAL syntax. State formulas indicated by ϕ and ψ are all in the form $R \bowtie n$, with R the identifier of a reactant in the model, $\bowtie \in \{<, \leq, =, \geq, >\}$ and $n \in [0, g(R)]$ a valid activity level value between 0 and the granularity (number of discrete levels) of R.

addition makes it easier to inspect the behavior of a network by using a sequence of model checking interrogations.

6 Conclusions and future work

We have presented here how the ANIMO tool was improved to provide a more interactive modeling process. Thanks to the increased performances of the new reactant-centered modeling approach, we are able to obtain answers to model checking queries in a matter of seconds. The features of model checking are made accessible without the need to directly deal with TA models. In this way, ANIMO acts as an intermediary between the biologist and a formal representation of biological signaling pathways, letting the experts concentrate on investigating the mechanisms of cellular responses.

In order to enforce the concept of user interaction as a primary focus of the tool, we plan to extend ANIMO with support for parameter sensitivity analysis and parameter fitting, as a follow-up to what was presented in [16]. Moreover, inspired by a work on automata learning [21], we plan to add also the possibility to automatically derive a network topology based on experimental data and previous knowledge.

We aim at widening the available set of model checking queries, in order to allow biologists to perform in silico experiments on an already fitting model and to obtain answers to more relevant questions. This would increase the usefulness of a model as a help to drive wet-lab investigation. In order to allow for meaningful in silico experiments, we plan to purposefully introduce user-defined non-deterministic parts in our models, which would allow for drug dosage investigations through model checking. This can be done e.g. through the definition of intervals for the values of some reaction kinetic constants, adding considerable uncertainty in the timing of those reactions. More useful results can be achieved through proper application of the statistical model checking part of UPPAAL [18] to an extended version of our model to include realistically significant stochastic behavior. Finally, in order to further improve performances, the extension of ANIMO with support for a multi-core model checking approach based on the work by Dalsgaard et al. [22] is under study.

References

1. Kanehisa, M., Goto, S.: KEGG: Kyoto Encyclopedia of Genes and Genomes. Nucleic Acids Research **28**(1) (January 2000) 27–30
2. Hornbeck, P.V., Chabra, I., Kornhauser, J.M., Skrzypek, E., Zhang, B.: PhosphoSite: A bioinformatics resource dedicated to physiological protein phosphorylation. Proteomics **4**(6) (2004) 1551–1561
3. Ciocchetta, F., Hillston, J.: Bio-PEPA: A framework for the modelling and analysis of biological systems. Theor. Comput. Sci. **410** (August 2009) 3065–3084
4. Dematté, L., Priami, C., Romanel, A.: Modelling and simulation of biological processes in BlenX. SIGMETRICS Perform. Eval. Rev. **35** (March 2008) 32–39
5. Mendes, P., Hoops, S., Sahle, S., Gauges, R., Dada, J., Kummer, U.: Computational modeling of biochemical networks using COPASI systems biology. Volume 500 of Methods in Molecular Biology. Humana Press, Totowa, NJ (2009) 17–59
6. Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T.S., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S., Yugi, K., Venter, J.C., III, C.A.H.: E-CELL: software environment for whole-cell simulation. Bioinformatics **15**(1) (January 1999) 72–84
7. de Jong, H., Geiselmann, J., Hernandez, C., Page, M.: Genetic Network Analyzer: qualitative simulation of genetic regulatory networks. Bioinformatics **19**(3) (February 2003) 336–344
8. ANIMO. `http://fmt.cs.utwente.nl/tools/animo`
9. Schivo, S., Scholma, J., Wanders, B., Urquidi Camacho, R., van der Vet, P., Karperien, M., Langerak, R., van de Pol, J., Post, J.: Modelling biological pathway dynamics with Timed Automata. IEEE Journal of Biomedical and Health Informatics **18**(3) (2013) 832–839
10. Scholma, J., Schivo, S., Urquidi Camacho, R., van de Pol, J., Karperien, M., Post, J.: Biological networks 101: Computational modeling for molecular biologists. Gene **533**(1) (2014) 379–384
11. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126** (April 1994) 183–235
12. Killcoyne, S., Carter, G.W., Smith, J., Boyle, J.: Cytoscape: a community-based framework for network modeling. Methods in molecular biology (Clifton, N.J.) **563** (2009) 219–239
13. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer (STTT) **1** (1997) 134–152
14. Cytoscape 3 ANIMO app. `http://apps.cytoscape.org/apps/animo`

15. Santos, S.D.M., Verveer, P.J., Bastiaens, P.I.H.: Growth factor-induced MAPK network topology shapes Erk response determining PC-12 cell fate. Nature Cell Biology **9**(3) (February 2007) 324–330
16. Schivo, S., Scholma, J., Karperien, H.B.J., Post, J.N., van de Pol, J.C., Langerak, R.: Setting parameters for biological models with ANIMO. In André, E., Frehse, G., eds.: Proceedings 1st International Workshop on Synthesis of Continuous Parameters, Grenoble, France. Volume 145 of Electronic Proceedings in Theoretical Computer Science., Open Publishing Association (April 2014) 35–47
17. Daigle, B.J., Srinivasan, B.S., Flannick, J.A., Novak, A.F., Batzoglou, S.: Current progress in static and dynamic modeling of biological networks. In Choi, S., ed.: Systems Biology for Signaling Networks. Volume 1 of Systems Biology. Springer New York (2010) 13–73
18. David, A., Larsen, K., Legay, A., Mikučionis, M., Wang, Z.: Time for statistical model checking of real-time systems. In Gopalakrishnan, G., Qadeer, S., eds.: Computer Aided Verification. Volume 6806 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2011) 349–355
19. Scholma, J., Schivo, S., Kerkhofs, J., Langerak, R., Karperien, H.B.J., van de Pol, J.C., Geris, L., Post, J.N.: ECHO: the executable chondrocyte. In: Tissue Engineering & Regenerative Medicine International Society, European Chapter Meeting, Genova, Italy. Volume 8., Malden, Wiley (June 2014) 54–54
20. Monteiro, P.T., Ropers, D., Mateescu, R., Freitas, A.T., de Jong, H.: Temporal logic patterns for querying dynamic models of cellular interaction networks. Bioinformatics **24**(16) (2008) i227–i233
21. Tretmans, J.: Model-based testing and some steps towards test-based modelling. In Bernardo, M., Issarny, V., eds.: Formal Methods for Eternal Networked Software Systems. Volume 6659 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2011) 297–326
22. Dalsgaard, A.E., Laarman, A.W., Larsen, K.G., Olesen, M.C., van de Pol, J.C.: Multi-core reachability for timed automata. In Jurdzinski, M., Nickovic, D., eds.: 10th International Conference on Formal Modeling and Analysis of Timed Systems, FORMATS 2012, London, UK. Volume 7595 of Lecture Notes in Computer Science., London, Springer Verlag (September 2012) 91–106

Appendix A UPPAAL functions for the example model

A.1 Globlal functions and declarations

```
clock globalTime;
const int INFINITE_TIME = −1;
const int N_REACTANTS = 4;
broadcast chan reacting[N_REACTANTS];

//Reactant A
int R0 := 0;
const int R0Levels := 100;

//Reactant B
int R1 := 100;
const int R1Levels := 100;

//Reactant C
int R2 := 100;
const int R2Levels := 100;

//Reactant D
int R3 := 100;
const int R3Levels := 100;


typedef struct {
  int[−99980001, 99980001] b; //99980001 = 9999 * 9999
  int e;
} double_t; //"double" is a reserved word


const double_t INFINITE_TIME_DOUBLE = {−1000, −3}, //INFINITE_TIME (−1)
    translated into double
          ZERO = {0, 0}, // 0
          TWO = {2000, −3}; // 2
const double_t LOWER_UNC = {9500, −4}, //Lower and upper scale factors to
    apply uncertainty. E.g. for +/− 5% uncertainty, we have lower uncertainty
     = 0.95, upper uncertainty = 1.05
               UPPER_UNC = {1050, −3};

typedef int[−1, 1073741822] time_t;


//Reaction R1 (B) −−> R0 (A)
timeActivity R1_R0;
```

```
const double_t k_R1_R0 = {4000, -6};

//Reaction R2 (C) --> R0 (A)
timeActivity R2_R0;
const double_t k_R2_R0 = {4000, -6};

//Reaction R3 (D) --| R0 (A)
timeActivity R3_R0;
const double_t k_R3_R0 = {4000, -6};



double_t subtract(double_t a, double_t b) { // a - b
  double_t r = {-1000, -1000};
  if (a.b == 0) {
    r.b = -b.b;
    r.e = b.e;
    return r;
  }
  if (b.b == 0) {
    return a;
  }
  if ((a.e - b.e) >= 4) return a;
  if ((b.e - a.e) >= 4) {
    r.b = -b.b;
    r.e = b.e;
    return r;
  }
  if (a.e == b.e) {
    r.b = a.b - b.b;
    r.e = a.e;
  }
  if (a.e - b.e == 1) {
    r.b = a.b - b.b/10;
    r.e = a.e;
  }
  if (a.e - b.e == 2) {
    r.b = a.b - b.b/100;
    r.e = a.e;
  }
  if (a.e - b.e == 3) {
    r.b = a.b - b.b/1000;
    r.e = a.e;
  }
  if (b.e - a.e == 1) {
    r.b = a.b/10 - b.b;
    r.e = b.e;
  }
  if (b.e - a.e == 2) {
    r.b = a.b/100 - b.b;
    r.e = b.e;
  }
  if (b.e - a.e == 3) {
    r.b = a.b/1000 - b.b;
    r.e = b.e;
  }
  if ((r.b > 0 && r.b < 10) || (r.b < 0 && r.b > -10)) {
    r.b = r.b * 1000;
    r.e = r.e - 3;
  } else if ((r.b > 0 && r.b < 100) || (r.b < 0 && r.b > -100)) {
    r.b = r.b * 100;
    r.e = r.e - 2;
  } else if ((r.b > 0 && r.b < 1000) || (r.b < 0 && r.b > -1000)) {
    r.b = r.b * 10;
    r.e = r.e - 1;
  } else if (r.b > 9999 || r.b < -9999) {
    r.b = r.b / 10;
    r.e = r.e + 1;
  }
  return r;
}

double_t add(double_t a, double_t b) { // a + b
  double_t r = {-1000,-1000};
  if (a.b == 0) {
    return b;
  }
  if (b.b == 0) {
    return a;
  }
  if ((a.e - b.e) >= 4) return a;
  if ((b.e - a.e) >= 4) return b;
  if (a.e == b.e) {
    r.b = a.b + b.b;
    r.e = a.e;
  }
  if (a.e - b.e == 1) {
    r.b = a.b + b.b/10;
    r.e = a.e;
  }
  if (a.e - b.e == 2) {
    r.b = a.b + b.b/100;
    r.e = a.e;
  }
  if (a.e - b.e == 3) {
    r.b = a.b + b.b/1000;
    r.e = a.e;
  }
  if (b.e - a.e == 1) {
    r.b = a.b/10 + b.b;
    r.e = b.e;
  }
  if (b.e - a.e == 2) {
    r.b = a.b/100 + b.b;
    r.e = b.e;
  }
  if (b.e - a.e == 3) {
    r.b = a.b/1000 + b.b;
    r.e = b.e;
  }
  if ((r.b > 0 && r.b < 10) || (r.b < 0 && r.b > -10)) {
    r.b = r.b * 1000;
    r.e = r.e - 3;
  } else if ((r.b > 0 && r.b < 100) || (r.b < 0 && r.b > -100)) {
    r.b = r.b * 100;
    r.e = r.e - 2;
  } else if ((r.b > 0 && r.b < 1000) || (r.b < 0 && r.b > -1000)) {
    r.b = r.b * 10;
    r.e = r.e - 1;
  } else if (r.b > 9999 || r.b < -9999) {
    r.b = r.b / 10;
    r.e = r.e + 1;
  }
  return r;
}

double_t multiply(double_t a, double_t b) { // a * b
  double_t r;
  r.b = a.b * b.b;
  if (r.b % 1000 < 500) {
    r.b = r.b / 1000;
  } else {
    r.b = 1 + r.b / 1000;
  }
  r.e = a.e + b.e + 3;
  if ((r.b > 0 && r.b < 10) || (r.b < 0 && r.b > -10)) {
    r.b = r.b * 1000;
    r.e = r.e - 3;
  } else if ((r.b > 0 && r.b < 100) || (r.b < 0 && r.b > -100)) {
    r.b = r.b * 100;
    r.e = r.e - 2;
  } else if ((r.b > 0 && r.b < 1000) || (r.b < 0 && r.b > -1000)) {
    r.b = r.b * 10;
    r.e = r.e - 1;
  } else if (r.b > 9999 || r.b < -9999) {
    r.b = r.b / 10;
    r.e = r.e + 1;
  }
  return r;
}

double_t inverse(double_t a) { // 1 / a
  double_t r;
  if (a.b == 0 || a.e < -9) { // 1 / 1e-9 is still ok, but 1 / 1e-10 is too
      large (> 2^30 - 2, the largest allowed constant for guards/invariants)
    return INFINITE_TIME_DOUBLE;
  }
  r.b = 1000000 / a.b;
  r.e = -6 - a.e;
  if ((r.b > 0 && r.b < 10) || (r.b < 0 && r.b > -10)) {
    r.b = r.b * 1000;
    r.e = r.e - 3;
  } else if ((r.b > 0 && r.b < 100) || (r.b < 0 && r.b > -100)) {
    r.b = r.b * 100;
    r.e = r.e - 2;
  } else if ((r.b > 0 && r.b < 1000) || (r.b < 0 && r.b > -1000)) {
    r.b = r.b * 10;
    r.e = r.e - 1;
  }
  return r;
}

time_t power(int a, int b) { // a ^ b (b >= 0)
  time_t r = 1;
  while (b > 0) {
    r = r * a;
    b = b - 1;
  }
  return r;
}
```

```
time_t round(double_t a) { // double --> integer
  if (a == INFINITE_TIME_DOUBLE) { // Don't need to translate literally if
      we have infinite
    return INFINITE_TIME;
  }
  if (a.e < -3) {
    if (a.b < 5000) return 0;
    else return 1;
  }
  if (a.e == -1) {
    if (a.b % 10 < 5) {
      return a.b / 10;
    } else {
      return 1 + a.b / 10;
    }
  }
  if (a.e == -2) {
    if (a.b % 100 < 50) {
      return a.b / 100;
    } else {
      return 1 + a.b / 100;
    }
  }
  if (a.e == -3) {
    if (a.b % 1000 < 500) {
      return a.b / 1000;
    } else {
      return 1 + a.b / 1000;
    }
  }
  return a.b * power(10, a.e);
}

double_t scenario1(double_t k, double_t r1, double_t r1Levels, bool r1Active
    ) {
  double_t E;
  if (r1Active) { //If we depend on active R1, the level of activity is the
      value of E
    E = r1;
  } else { //otherwise we find the inactivity level via the total number of
      levels
    E = subtract(r1Levels, r1);
  }
  return multiply(k, E);
}


double_t scenario2_3(double_t k, double_t r2, double_t r2Levels, bool
    r2Active, double_t r1, double_t r1Levels, bool r1Active) {
  double_t E, S;
  if (r1Active) { //If we depend on active R1, the level of activity is the
      value of E
    E = r1;
  } else { //otherwise we find the inactivity level via the total number of
      levels
    E = subtract(r1Levels, r1);
  }
  if (r2Active) { //Same for R2
    S = r2;
  } else {
    S = subtract(r2Levels, r2);
  }
  return multiply(k, multiply(E, S));
}


double_t int_to_double(int a) { //Used to translate an activity level into
    double.
  double_t r;
  if (a < 10) {
    r.b = a * 1000;
    r.e = -3;
  } else if (a < 100) {
    r.b = a * 100;
    r.e = -2;
  } else if (a < 1000) {
    r.b = a * 10;
    r.e = -1;
  } else if (a < 10000) { //We allow up to 100 levels
    r.b = a;
    r.e = 0;
  }
  return r;
}
```

A.2 Functions and declarations local to each automaton

```
//Local variables to the automaton representing the reactions influencing R0
    (also called A)

clock c;
int[-1, 1] delta = 0;
time_t tL = INFINITE_TIME,
       tU = INFINITE_TIME,
       tHalf = INFINITE_TIME;
double_t rateA = INFINITE_TIME_DOUBLE,
         oldRateA = INFINITE_TIME_DOUBLE;

double_t compute_rate() {
  // B --> A
  double_t R1_R0_r = scenario1(k_R1_R0, int_to_double(R1), int_to_double(
     R1Levels), true);
  // C --> A
  double_t R2_R0_r = scenario1(k_R2_R0, int_to_double(R2), int_to_double(
     R2Levels), true);
  // D --> A
  double_t R3_R0_r = scenario1(k_R3_R0, int_to_double(R3), int_to_double(
     R3Levels), true);
  return subtract(add(R1_R0_r, R2_R0_r), R3_R0_r);
}

void update() {
  oldRateA = rateA;
  rateA = compute_rate();
  oldDelta = delta;
  if (rateA.b < 0) {
    delta = -1;
    rateA.b = -rateA.b;
  } else {
    delta = 1;
  }
  if (rateA.b != 0) {
    tL = round(multiply(inverse(rateA), LOWER_UNC));
    tU = round(multiply(inverse(rateA), UPPER_UNC));
  } else {
    tL = INFINITE_TIME;
    tU = INFINITE_TIME;
  }
  tHalf = round(inverse(multiply(rateA, TWO))); //tHalf = T_A / 2
}
```

```
void react() {
  if (0 <= R0 + delta && R0 + delta <= MAX) {
    R0 = R0 + delta;
  }
  update();
}

//True if the new rate is at least double the old one,
//or if the sign of the rates is different (the reaction changes direction)
bool rate_significantly_changed(double_t oldRate, double_t newRate) {
  double_t diff = subtract(multiply(newRate, inverse(oldRate)), TWO);
  if ((oldRate.b < 0 && newRate.b >= 0 || oldRate.b >= 0 && newRate.b < 0)
      || diff.b >= 0) {
    return true;
  }
  return false;
}

//If the updated conditions have significantly changed the rate,
//restart the reaction from the beginning, without considering the work
    already done.
void decide_reset() {
  if (rate_significantly_changed(oldRate, rateA)) {
    c = 0;
  }
}

//If the update in conditions would significantly change the rate,
//rush the reaction (we have completed at least 50% of it: see the guard c
    >= tHalf)
void decide_react() {
  if (rate_significantly_changed(rateA, compute_rate())) {
    tL = 0;
    tU = 0;
    c = 0;
  }
}

bool can_react() {
  return T != INFINITE_TIME && T != 0 && ((delta > 0 && R0 < R0Levels) || (
      delta < 0 && R0 > 0));
}
```