# Discretization of continuous dynamical systems using UPPAAL

Stefano Schivo and Rom Langerak

Formal Methods and Tools Group, Faculty of EEMCS, University of Twente, Enschede, The Netherlands

**Abstract.** We want to enable the analysis of continuous dynamical systems (where the evolution of a vector of continuous state variables is described by differential equations) by model checking. We do this by showing how such a dynamical system can be translated into a discrete model of communicating timed automata that can be analyzed by the UPPAAL tool. The basis of the translation is the well-known Euler approach for solving differential equations where we use fixed discrete value steps instead of fixed time steps. Each state variable is represented by a timed automaton in which the delay for taking the next value is calculated on the fly using the differential equations. The state variable automata proceed independently but may notify each other when a value step has been completed; this leads to a recalculation of delays. The approach has been implemented in the tool ANIMO for analyzing biological kinase networks in cells. This tool has been used in actual biological research on osteoarthritis dealing with systems where the dimension of the state vector (the number of nodes in the network) is in the order of one hundred.

**Keywords:** discretization; Euler method; model checking; timed automata; systems biology

## 1 Introduction

In this introduction we first motivate our interest in discretizing continuous systems using UPPAAL, then we give a short characterization of our approach, and finally we give an overview of the paper.

Many important and interesting phenomena in nature and technology can be adequately modeled as continuous dynamical systems where the evolution of a vector of real state variables is governed by differential equations. The mathematical theory of continuous dynamical systems is a mature field with a history of centuries, and occupies a firm position in any science or engineering curriculum.

The last decades have seen a great interest in the analysis of continuous dynamical systems using techniques from computer science developed in the context of discrete systems. A prominent example of such a technique is model

checking as originated in the '80s [14, 9], where properties (often given in some kind of logical formalism) are checked against a model of a system, usually in the form of a discrete state transition system. The attractiveness of model checking lies in the fact that large and complex systems can be automatically and exhaustively checked. Model checking has fruitful applications to scheduling and control synthesis: reachability analysis may yield witness traces that contain the relevant information for a schedule or a control strategy. Our interest in model checking dynamical systems theory is aimed at biological applications; we refer to [5, 23] for an overview of model checking biological systems.

Systems where timing aspects are critical can be modeled by enhancing state transition systems by real time clocks, leading to the timed automata model [1]. The application of symbolic techniques to (networks of) timed automata has led to effective model checking tools, most notably UPPAAL [20]. The UPPAAL website [36] contains an ample collection of applications of UPPAAL to e.g. protocol analysis, hardware verification and model checking, controller design, and scheduling. The fact that UPPAAL is a mature and powerful tool that is widely used makes it an attractive infrastructure for model checking continuous dynamical systems.

When modeling continuous dynamical systems by timed automata two problems have to be addressed. Firstly, timed automata can directly model only very simple dynamics: clock variables with a slope of 1. And secondly, timed automata are basically a discrete model. So some abstraction technique has to be found in order to represent continuous dynamics and values by using simple clocks and discrete values. The literature on abstractions of continuous dynamics is too vast to be dealt with here (we refer to [2] for an early overview); moreover, here we are primarily interested in those approaches that aim at timed automata as a target model. We identify two possible types of approaches.

One type of approach is to exploit knowledge of special properties of the dynamics (possibly restricted to a special class of dynamics) in order to obtain a discrete abstraction of the dynamical system. An example of this approach is given by [33, 34, 38] where the state space is partitioned by level sets of a kind of Lyapunov function. Another recent example is [4] where an abstraction technique based on time-varying regions of invariance (so-called control funnels) is applied to linear systems.

A second type of approach takes as a starting point a partitioning of the state space into rectangular cells and uses the differential equations to obtain information about reachability between cells. Examples of this approach are [22, 32, 6, 8, 15] (they will be discussed in Section 2 of this paper). The approach in this paper falls in this category: we assume a discretization of a bounded part of the state space, even if we do not explicitly create cells in our approach. Conceptually our approach is just a slight modification of the well-known Euler approach for solving differential equations, where we take fixed value steps instead of fixed time steps. This leads to an arbitrary precision approximation of the dynamical system.

Our approach originated in the context of a biological application, viz. the analysis of kinase networks in living cells [29, 37, 30]. This means that we should be able to model check dynamical systems with dimensions in the order of 50-100. We do not just want to use model checking for the purpose of analyzing our systems (along the lines of e.g. [23]) but we also want to use model checking in the following way. Suppose we model a network with all possible stimuli on the network, and we express a certain therapy target as a property on the network. Now model checking that the target cannot be reached may lead to a counterexample, representing exactly the stimuli leading to the desired target. In this way model checking is being used for drug synthesis, which is becoming even more important in the context of personalized therapies. Prerequisite for this is that our models are amenable to model checking in an efficient way. Many of the existing approaches in the literature are mathematically sophisticated but do not (yet) scale up to systems with large dimensions. Our approach is conceptually simple, but has proven to be very efficient for a range of biological network models.

The field of nonlinear dynamical systems is complex and challenging. We would like to stress that our aim is not primarily to develop theory leading to a better understanding of this field. Instead, we obtain a discretization of a system using the Euler method, after which UPPAAL does most of the work. This tool–based approach works for an interesting application area (biological networks); it will be interesting to see whether it will work for other applications as well.

This paper is structured as follows. In Section 2 we describe the problem and discuss related approaches. In Section 3 we discuss the Euler method and our adaptation of it. In Section 4 we implement our version of the Euler method using timed automata and show an example. In Section 5 we discuss the correctness and make our implementation efficient. In Section 6 we describe how our approach has been implemented into the tool ANIMO that has already been used in biological research [28, 27, 26, 31], and Section 7 contains conclusions and directions for further research.

## 2 Problem statement and related work

We assume a dynamical system has a state vector $\mathbf{x} \in \mathbb{R}^n$, with $\mathbf{x} = (x_1, \ldots, x_n)$; $x_i$ $(1 \leq i \leq n)$ is called a component of $\mathbf{x}$. We consider $\mathbf{x}$ as a function of time, and assume the dynamics of $\mathbf{x}$ to be governed by the differential equation

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}) \quad (*)$$

The function $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$ defines a vector field, i.e. it associates a vector $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ to each point in $\mathbb{R}^n$ (note that $\mathbf{f}$ does not explicitly depend on $t$).

A *trajectory* of the dynamical system starting in initial state $\mathbf{x}_0$ is a function $\mathbf{x}(t) : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ such that $\mathbf{x}(0) = \mathbf{x}_0$ and $\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(\mathbf{x}(t))$ for all $t \geq 0$.

In this paper we are only interested in a bounded (and closed) part $\mathbf{M}$ of $\mathbb{R}^n$; for convenience we assume $\mathbf{M} = [0, max_1] \times \ldots \times [0, max_n]$. This restricts trajectories to remain within $\mathbf{M}$, which means that the time domain on which a trajectory is defined may be restricted: if a trajectory tries to leave $\mathbf{M}$ it has to be truncated (but we will show in Section 4 a pragmatic way of dealing with this).

We assume $\mathbf{f}$ is continuously differentiable on $\mathbf{M}$, which is sufficient to guarantee the existence of a unique solution of the differential equation $(*)$ for each initial value in $\mathbf{M}$.

We define a grid on $\mathbf{M}$ by defining a grid step $d_i$ for each component of the state. We assume that $max_i/d_i = m_i$ is an integer. Let $\mathbf{k} = (k_1, \ldots, k_n)$ be a vector of integers with $0 \leq k_i \leq m_i$ for all i; we denote a cell in the grid by

$$C(k_1, \ldots, k_n) = [k_1 \cdot d_1, (k_1 + 1) \cdot d_1] \times \ldots [k_n \cdot d_n, (k_n + 1) \cdot d_n]$$

The possibility to have different grid steps for different components is quite convenient (and this is what we have implemented); however, for ease of exposition we assume we have a step size $d_i = 1$ for all components. By a slight abuse of notation we denote a cell $C(k_1, \ldots, k_n)$ by $(k_1, \ldots, k_n)$ or $\mathbf{k}$. Two cells are called neighbors if their difference is 1 in exactly one component, so e.g. $(k_1, \ldots, k_i, \ldots, k_n)$ and $(k_1, \ldots, k_i + 1, \ldots, k_n)$ are neighbors.

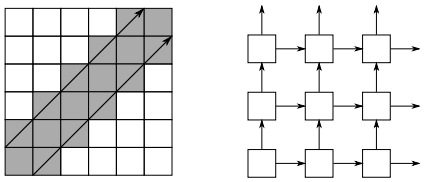

**Fig. 1.** A simple dynamical system with naive abstraction

Now the most simple idea for a discrete abstraction of the dynamical system would be to create a transition system with cells as locations, and transitions $\mathbf{k}_p \rightarrow \mathbf{k}_q$ between neighboring cells if there is a trajectory going from cell $\mathbf{k}_p$ to cell $\mathbf{k}_q$. A simple example (taken from [22]) shows that this naive abstraction contains too much spurious behavior (i.e. behavior that does not correspond to behavior in the original dynamical system) to be useful. Consider the simple dynamical system in Figure 1 with state in $\mathbb{R}^2$ and dynamics given by $\frac{d\mathbf{x}}{dt} = (1, 1)$. Trajectories starting in cell $(0, 0)$ may only reach the gray area, but in the abstracted transition system all cells are reachable from $(0, 0)$.

Moreover, this is independent from the granularity of the grid: no matter how small the grid step, the entire state space will always be reachable from cell $(0, 0)$. The reason for this inherent spurious behavior is that time has been completely disregarded in the abstraction. For instance, when going from cell

$(0,0)$ to $(0,100)$ at least 99 grid steps are taken in the vertical direction, and not even 1 grid step in the horizontal direction, which is impossible since the horizontal and vertical speeds are the same. This observation lies at the basis of [22]. In order to solve the problem, to each dimension of a cell a maximum dwell time is associated. This maximum dwell time is obtained by calculating the extremal values of the components of the derivative function (which can be efficiently calculated for the class of functions considered in [22]), an idea already presented in [35]. This may considerably reduce spurious behavior; however, as the authors remark, problems may arise when components of the derivatives are zero. In that case no bound on dwell times can be given, which still allows spurious behavior. Some of these problems have been solved in [8] in the context of inevitability analysis, but only for linear systems and low dimensions.

A different approach is presented in [6] where an analysis is made on the facets of a cell: by analyzing from which parts of facets other facets are reachable (taking the dynamics into account) cells are refined and spurious behavior is greatly reduced. The method is sophisticated but scaling it to our intended applications (with cells having dimensions in the order of 100) would lead to an explosion of refined cells. Using reachability between facets is also the basis of [11] where control theory is used in order to influence the reachability of facets from other facets as studied in [16, 17].

In our approach we do not explicitly create cells or transition relations between cells. Instead, we create a network of timed automata implementing the Euler approach for approximating a solution of a differential equation, and leave it to the UPPAAL tool to create a finite transition system as the underlying symbolic UPPAAL semantics. This approach has evolved from the IKNAT tool in [3] which models biological signaling networks. The timed automata created by IKNAT used a priori calculated delays between discrete activity levels of enzymes (a similar idea (but only for a one-dimensional system) has been used in [18] in the context of modeling battery lifetime). The IKNAT approach has been used by [15] in order to improve the quantitative modeling of gene regulatory networks in [32], thereby solving some instability problems of IKNAT. However, that approach is tied to a specific application and rather ad-hoc, conceptually not very clear, and the resulting timed automata are complicated and not efficient enough for effective model checking. The Euler-based approach we present in the next sections does not have these drawbacks.

## 3 The Euler method for solving differential equations

The Euler method is a well–known numerical procedure for solving differential equations. It can be found in most introductory books on calculus; for an extensive treatment we refer to [7].

The idea behind the Euler method is best illustrated on the one-dimensional case. Suppose we have a differential equation $\frac{dx}{dt} = f(x)$ and suppose we take time steps of size $h$. If at time $t_n$ we have an approximation $x_n$ of $x(t_n)$ we

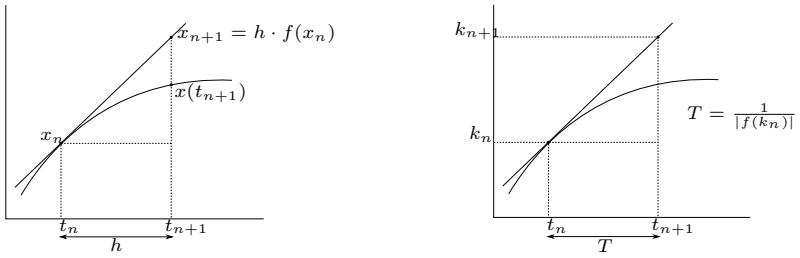


**Fig. 2.** The Euler method for a fixed time step (left) and for a fixed value step (right).

obtain the next approximation at time $t_{n+1} = t_n + h$ by $x_{n+1} = h \cdot f(x_n)$, see Figure 2 (left).

Starting at some initial value $x_0$ and then repeating this procedure yields a piecewise linear approximation of a trajectory of the dynamical system. The approximation error $|x(t_{n+1}) - x_{n+1}|$ goes to zero if the time step $h$ goes to zero, so this is an arbitrary precision approximation.

We use a variant of the Euler method where we take a fixed value step, and then calculate the time $T$ needed to arrive at this next value. This is illustrated in Figure 2 (right).

If the starting point is the discrete value $k_n$, then the next discrete value is $k_n + 1$ if $f(k_n) > 0$, and $k_n - 1$ if $f(k_n) < 0$; in both cases $T = 1/|f(k_n)|$.

We explain the procedure in higher dimensions on the case $\mathbb{R}^2$ (since it is easiest to depict), so we have equation $\frac{d\mathbf{x}}{dt} = (f_1(\mathbf{x}), f_2(\mathbf{x}))$:

- suppose we start at point $(p, q)$ with $p$ and $q$ integers
- the next cell $(p', q')$ reached will be either $(p \pm 1, q)$ (depending on the sign of $f_1(p, q)$) or $(p, q \pm 1)$ (depending on the sign of $f_2(p, q)$)
- the time $T$ to reach the next cell: $T = min\{1/|f_1(p, q)|, 1/|f_2(p, q)|\}$
- now repeat this from the point where the next cell is entered, with vector $\mathbf{f}(p', q')$

This is illustrated in Figure 3. Notice that when the procedure starts from a point $\mathbf{x}$ that has just been reached on the cell boundary of cell $\mathbf{k}$ we use the vector value $\mathbf{f}(\mathbf{k})$ instead of the value of $\mathbf{f}(\mathbf{x})$, since we also want to obtain a discretization of the state space. When the grid size tends towards zero, $\mathbf{f}(\mathbf{x})$ tends towards $\mathbf{f}(\mathbf{k})$, so this does not harm the arbitrary precision property of our approximation.

This version of the Euler method easily generalizes to dimension $n$:

**Initialization:**

Suppose we start at point $\mathbf{k}$. The waiting time for each component $i$: $T_i = 1/|f_i(\mathbf{k})|$ (the time needed for reaching the next value $k_i \pm 1$, depending on the sign of $f_i(\mathbf{k})$). Note that if $f_i(\mathbf{k}) = 0$, $T_i = \infty$.

**Repeat step:**

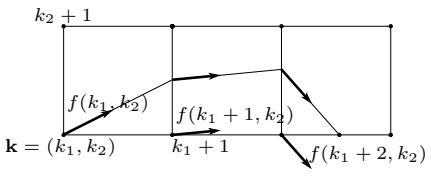


**Fig. 3.** example of the Euler-based method in two dimension

**begin**
For all $j$ for which $T_j = min\{T_1, \ldots, T_n\} = t$:
- $k'_j = k_j + 1$ if $f_i(\mathbf{k}) > 0$
- $k'_j = k_j - 1$ if $f_i(\mathbf{k}) < 0$

Update all waiting times, based on $t$ and $\mathbf{k}'$, as indicated below.
**end**

Waiting times are updated as follows:

- if $T_i = min\{T_1, \ldots, T_n\}$ or $f_i(\mathbf{k}) = 0$ or $f_i(\mathbf{k}') = 0$:
  $T'_i = 1/|f_i(\mathbf{k}')|$
- if $f_i(\mathbf{k}')$ and $f_i(\mathbf{k})$ have the same sign:
  This situation is represented in Figure 4 for a positive $f_i(\mathbf{k})$. At time $t$ a distance $(T_i - t) \cdot |f_i(\mathbf{k})|$ still needs to be covered before reaching $k_i + 1$, so the new delay time becomes $T'_i = (T_i - t) \cdot |f_i(\mathbf{k})|/|f_i(\mathbf{k}')|$. Similar reasoning leads to the same formula if $f_i(\mathbf{k})$ is negative.
- if $f_i(\mathbf{k}')$ and $f_i(\mathbf{k})$ have opposite sign:
  Suppose $f_i(\mathbf{k})$ is positive (see Figure 4). At time $t$ a distance $(T_i - t) \cdot |f_i(\mathbf{k})|$ would still have to be covered before reaching $k_1 + 1$. But now the direction is changed, so an extra $1 - (T_i - t) \cdot |f_i(\mathbf{k})|$ has to be covered before $k_i - 1$ is reached, so a total of $2 - (T_i - t) \cdot |f_i(\mathbf{k})|$. So the new delay time becomes $T'_i = (2 - (T_i - t) \cdot |f_i(\mathbf{k})|)/|f_i(\mathbf{k}')|$. Similar reasoning leads to the same formula if $f_i(\mathbf{k})$ is negative.

## 4 Translation into timed automata

We implement the Euler method of the previous section by a network of communicating timed automata. We first describe this implementation in an abstract way using pseudo-code and without worrying about syntactical and semantical issues. Then we show how to concretely implement this in UPPAAL, taking into account syntactical, semantical, and performance issues.

For each dimension $i$ of the state space we create a timed automaton $A_i$ containing a discrete state component $k_i$ and a clock variable $c_i$. Clock $c_i$ counts up to time $T_i = 1/|f_i(\mathbf{k})|$ which is the delay time for reaching next integer value $k_i \pm 1$, depending on the sign of $f_i(\mathbf{k})$. If $f_i(\mathbf{k}) = 0$ the component is for the

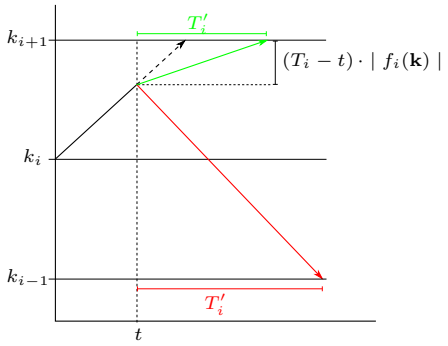


**Fig. 4.** Calculating a new delay time

time being quiescent, so $T_i = \infty$. While an automaton is waiting, two things may happen:

1. $c_i = T_i$, so the component has reached the new integer grid value $k_i' = k_i \pm 1$ (we call this *reaching*). Now all other automata are notified of this fact, and a new delay time $T_i'$ is calculated for reaching the next grid value: $T_i' = 1/|f_i(k_1, \ldots, k_i', \ldots, k_n)|$, and clock $c_i$ is reset.
2. a notification is received from automaton $A_j$ that it has reached a new grid value $k_j'$. Now a new delay $T_i'$ has to be calculated, based on the new value $f_i(k_1, \ldots, k_j', \ldots, k_n)$ and the time that has already been waited, i.e. the current value of clock $c_i$, in exactly the same way as in the Euler algorithm in the previous section. The clock $c_i$ is reset.

An abstract version of automaton $A_i$ with pseudo-code is given in Figure 5.

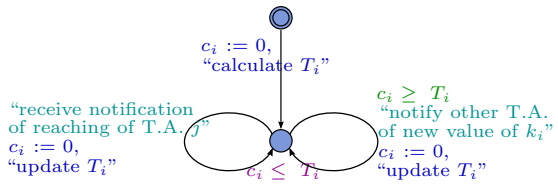


**Fig. 5.** Automaton $A_i$ in pseudo code

We now change the abstract timed automaton with pseudo-code of Figure 5 into a concrete UPPAAL automaton. The result is given in Figure 6; from now on we assume the automaton $A_i$ corresponding to component $x_i$ has id $i$ (and we often blur the distinction between components and their corresponding automata).

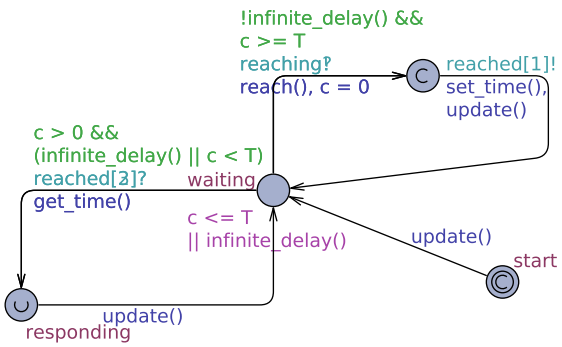


**Fig. 6.** The resulting UPPAAL automaton with id=1

We explain step by step the various issues in this automaton.

**Notifying when reaching.** When automaton $A_i$ reaches the grid-point, it is not the case that all other automata have to be notified. Only those components $x_j$ have to be notified whose derivate $f_j$ may change as a result of a change in $x_i$, i.e. $\frac{\partial f_j(\mathbf{x})}{\partial x_i} \neq 0$ for some value of $\mathbf{x}$. This can usually be detected at syntactic level: if variable $x_i$ does not occur in the mathematical formula for $f_j$, then $x_j$ is not dependent on $x_i$.

In Figure 6 the depicted automaton $A_1$ is dependent on $A_2$ and $A_3$ so there are transitions from location `waiting` labeled with `reached[2]?` and `reached[3]?`. Note that these transitions have been depicted in an overlapping fashion to enhance readability of the picture. These transitions then reach urgent location `responding` from which a transition is taken to update the delay time $T_1$.

**Multiple automata reaching** It is possible that several automata reach a new grid-point at the same time. In order to guarantee a consistent resulting update of the global state we want the resulting sequence of updates to be atomic. To achieve this we synchronize all reaching transitions from location `waiting` by having two transitions labeled with `reaching!` and `reaching?` (again these two transitions are depicted overlappingly). Then a committed location is reached from which a transition `reached[id]!` is performed to notify the dependent automata. Notice that this location is committed so the transition takes precedence over transitions in automata leaving urgent location `responding`.

If an automaton has just performed a `reached[id]!` transition and reached location `waiting` we do not want it to receive to a notification of another automaton that has just reached. Therefore we add predicate `c>0` to the guard on the transitions leaving location `waiting`.

**Calculating the clock time.** We saw in the last section that the value of clock $c_i$ needs to be known at the moment the delay time $T_i$ needs to be updated. However, UPPAAL does not allow the clock time to be read. Therefore we perform some global time administration in order to extract the clock time. We

introduce a global variable `currentTime` that records the global time. Each automaton has a local variable `lastUpdate` that records the global time at the last delay update. Now if an automaton $A_i$ reaches a new grid value this means it has waited the full delay time $T_i$. Therefore the new global time is set to `lastUpdate + `$T_i$; this happens in function `set_time()`. When a notification is received, the current clock time is `currentTime - lastUpdate`; this is calculated in function `get_time()`.

**The `reach()` function**. When an automaton has reached a new grid value this value should be communicated to the dependent automata. Since UPPAAL does not enable value passing in synchronizations this is done by writing the new value in a global variable that can be used by an `update()` function elsewhere. This happens in the function `reach()`. When the new grid value has reached one of the extremal values 0 or $max_i$ the automaton $A_i$ enters a quiescent mode by setting $T_i$ to infinite. It may leave this quiescent mode when changes in other components make the component move away from the extremal value (so we do not need to truncate the trajectory by stopping time). This is a pragmatic solution that makes sense in many applications (e.g. the biological application in Section 6), but needs to be evaluated in the light of each specific application area.

**The `update()` function.** The function `update` calculates the new time delay for reaching the next grid value, as described in the previous section. It is used in different contexts: when initializing, after reaching, and after having received a notification. In the latter case it has to make use of the old delay time, in the other two cases (characterized by the clock value being 0) it does not need the old delay time.

**Dealing with infinite waiting times.** When the derivative of a component is 0 the waiting time is infinite. This could be modeled by a very large number, but it is not a good idea to put very large numbers in UPPAAL clock guards. Therefore a boolean function `infinite_delay()` has been defined that is true when the waiting time is (conceptually) infinite. This function is updated by function `update()`.

**Numerical representations.** Our real number computations would require floating-point precision. Since 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 together with all other functions such as `update()` and `react()`, inside any UPPAAL model file generated by ANIMO [1].

[1] 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`

This completes our explanation of our UPPAAL implementation. The resulting UPPAAL specification can be used for simulation.

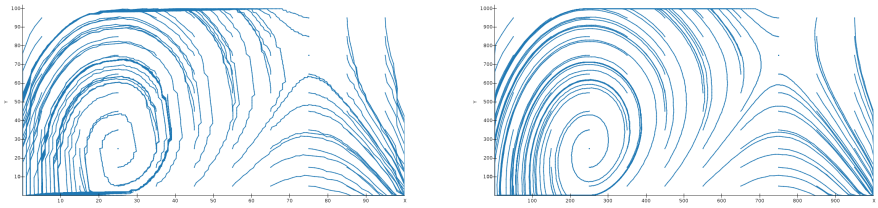

**Fig. 7.** Phase plane ($x$ vs $y$ plot) of the system in (1) obtained with the model from Figure 6 on the intervals $[0, 100]$ (left) and $[0, 1000]$ (right), consequently adapting the equations to fit the $\mathbb{R}^2$ subsets. Starting values for $x$ and $y$ are defined on the $[0, 100]$ interval by the equations $x_0 = 5 + 10 \cdot j, y_0 = 5 + 10 \cdot k$, with $j, k = \{0, 1, \ldots, 9\}$.

In Figure 7 we show a phase plane representation of a simulation of following non-linear system with two unstable equilibrium points:

$$\begin{cases} \frac{dx}{dt} = x - y \\ \frac{dy}{dt} = 1 - 16(x - 0.5)^2 \end{cases} \tag{1}$$

We produced the graphs in Figure 7 directly in ANIMO, by translating the equations back into relations between nodes and edges in ANIMO's user interface and analyzing them with multiple initial values for $x$ and $y$. Computing the resulting 100 simulation runs took about 11 seconds for the first graph in Figure 7 and 17 seconds for the second.

For comparison we show in Figure 8 the phase-plane representation of the same system. Note that the UPPAAL simulation succeeds in faithfully capturing the qualitative behavior.

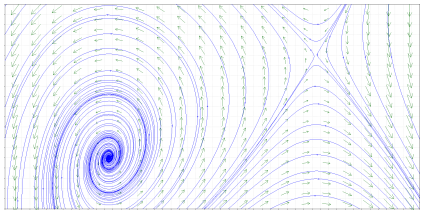

**Fig. 8.** Phase plane of the system in (1) obtained with the `pplane` software [24].

It is difficult to say something a priori about the accuracy of the Euler approximation; theoretical bounds on the truncation errors are not very helpful for

this (as they depend on properties of the differential equation that may be hard to establish). What these theoretical bounds do show is that the approximation error is linear in the step size [7]. However, rounding errors caused by the representation of numbers further complicate the picture. As a pragmatic way of dealing with the problem of accuracy we propose to experiment with different step sizes, and plot the resulting simulations, until one is sufficiently satisfied (when the plots do not change significantly anymore).

## 5 Correctness and efficiency

The translation of the previous section yields a set of $n$ timed automata (one for each component of the state vector) that synchronize via channels. We first discuss the correctness of this translation by showing how one step of the Euler algorithm in Section 3 relates to a sequence of transitions in the product of the timed automata.

We assume each automaton is in state `waiting`. Suppose an automaton $A_j$ has waited the allowed waiting time $T_j$ in state `waiting`, so $T_j = min\{T_1, \ldots, T_n\}$. Note that this may happen for several automata at the same moment; we call these automata the *reaching* automata.

Now the following sequence of transitions is performed (this is illustrated in Figure 9):

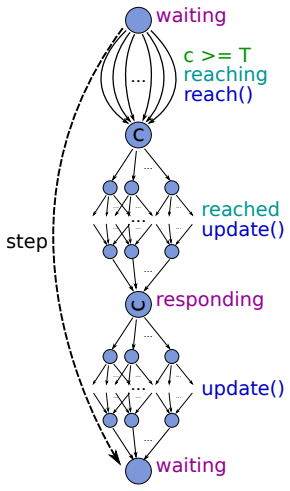


**Fig. 9.** Relation between timed automata transitions and Euler step.

– all the reaching automata synchronize on the `reaching` channel. Nondeterministically one of them performs `reaching!` and the others perform

`reaching?`, leading to several possible transitions that all end up in the same global state. In this global state all the reaching automata are in a committed state, and all non-reaching automata are still in state `waiting`. The value of **k** has been updated via the `reach()` function.
- then all transitions from the committed states are executed in an interleaved way. These transitions may synchronize via channel `reached` with non-reaching automata, while updating the waiting times of the reaching automata. When all `reached` transitions are finished each reaching automaton is in state `waiting`, and each non-reaching automaton is in the urgent state `responding`.
- finally all transitions from the urgent states `responding` are performed in an interleaved way, thereby updating the waiting times in the non-reaching automata. After this each automaton is in the state `waiting`.

So the three phases of transitions in Figure 9 taken together correspond to one step of the Euler algorithm in Section 3, showing the correctness of the translation.

The automaton in Figure 6 has all the required functionality and can satisfactorily be used for simulation. However, it is not yet suitable for model checking (especially for higher dimensional systems): because of interleavings of transitions in different automata, the resulting system would contain too many states. If in each automaton there is one transition that interleaves with the corresponding transitions in the other automata, and if the system is $n$-dimensional, then just that transition generates $2^n$ interleavings. Since it is our ambition to deal with systems where $n$ is in the order of 100, it is important to solve these problems. The problems, together with their remedies, are the following:

- from the `start` location: all `update` transitions interleave.
  Remedy: all update transitions are synchronized. The automaton with id 1 performs `do_update!` and all the other automata `do_update?`.
- from the `responding` location: all `update` transitions interleave.
  Remedy: all update transitions are synchronized. The automaton with the smallest id performs `do_update!` and all the other automata `do_update?`. The smallest responding id is established by `enter_responding()` and written into global variable `minIDresponding`. The smallest id performs `do_update!` and all the other automata `do_update?`.
- all transitions from the committed location interleave.
  Remedy: all `reaching` transitions are enqueued in a priority queue (implemented by a boolean array) by the function `enqueue(id)`. Now the `reached[`$i$`]!` transitions are performed by always choosing the minimum id in the queue, and removing the id from the queue by the function `exit_queue(id)`. In this way one interleaving is picked out of the exponentially many.

The resulting timed automaton is given in Figure 10. We will show in Section 6 that this model is amenable to model checking even for higher dimensions.

In many applications (like the biological application we deal with in Section 6) it is desirable to inject some imprecision in a model. This may be because of

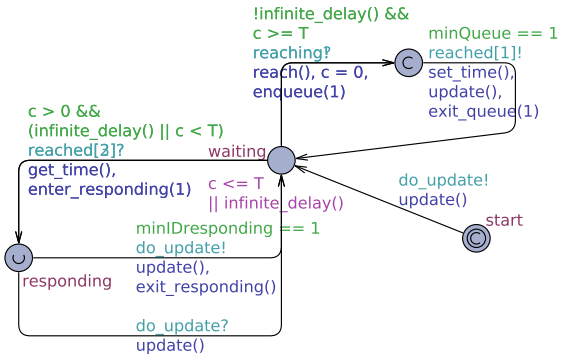


**Fig. 10.** Timed automaton with efficiency optimizations

inherent nondeterminism in the modeled phenomena, it may be because model parameters are not precisely known, or it may be because we want our model checking results to be robust against small perturbations in the behavior.

A simple pragmatic way of doing this is by turning the calculated delay times $T_i$ into intervals of possible delay times $[TL_i, TU_i]$ (where $L$ stands for Lower and $U$ stands for Upper). One might interpret such an interval as a uniform distribution of delay times (this approach was used for performance analysis in [39]). In the ANIMO tool in Section 6 such intervals are created by asking the user to specify an uncertainty percentage, say 10%, and then defining $TL = 0.9 \cdot T$ and $TU = 1.1 \cdot T$.

## 6 Application: ANIMO, a tool for analyzing kinase pathways

In this section we show how the approach has been implemented in the biological research tool ANIMO.

A signaling network in a biological cell describes the chain of reactions occurring between the reception of a signal and the response with which the cell reacts to such signal. The target of a signaling pathway is usually a transcription factor, a molecule with the task of controlling the production of some protein. Active molecules relay the signal inside the cell by activating other molecules until a target is reached. We define the activity level to represent the percentage of active molecules over an entire molecular species.

ANIMO (Analysis of Networks with Interactive MOdeling) [28, 27, 26, 31] is a software tool 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 ANIMO is accessible via the Cytoscape [19, 10] user interface (see Fig. 11). A user may insert a node for each molecular species and an edge for each reaction. The occurrence of a reaction modifies the activity level of its target reactant; the rate with which a reaction

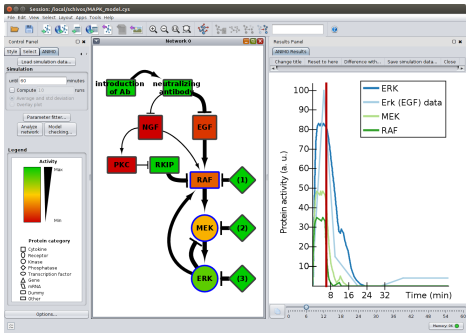


**Fig. 11.** The Cytoscape 3 interface for ANIMO

occurs depends on a formula selected by the user. For a more precise explanation on how reaction rates are computed in ANIMO, we refer to [27], for parameter setting in ANIMO to [25].

Once the user has created a model, this model is transformed into an UPPAAL model applying the discretizations described in the previous sections to the system of differential equations derived from the ANIMO network model. This model can then be analyzed via the Cytoscape/ANIMO interface that has facilities for model checking templates. ANIMO has also been used as a front-end for statistical model checking [12].

The ANIMO tool has been validated on several realistic biological case studies for which experimental data was available in the literature, and it has been demonstrated how to create models that faithfully fit experimental data [28, 27, 26, 31]. Moreover, ANIMO is being used in on-going research on chondrocyte signaling in relation to osteoarthritis, where the final objective is to enhance cartilage tissue engineering strategies [29, 30, 37].

Table 1 shows a comparison between some variants of the timed automata model when performing model checking on the model from [30] in UPPAAL. All the timed automata model variants contain 82 automata, i.e. they are a discretization of an 82-dimensional continuous dynamical system. "Approx. ±5%" is the approximated variant with a setting of 5% uncertainty level in the original ANIMO model.

| Model version | Time (s) | Memory (peak KB) |
|---|---|---|
| Standard | - | - |
| Approx. ±5% | - | - |
| Standard + optimizations | 139.61 | 3 880 040 |
| Approx. ±5% + optimizations | 63.13 | 1 107 552 |

**Table 1.** Performance comparison of different model versions.

The model was initially configured so that a large change in the node activities on the whole network would take place. We then asked a query to understand whether such change is inevitable. In the UPPAAL query language, this is written as `A<> R77 >= 80` (`R77` being the most interesting readout in the particular experiment). The answer was positive, except for the non-optimized models where the computation could not terminate after several hours.

## 7 Conclusions and Future Work

We have presented an arbitrary precision discretization of a continuous dynamical system as a network of UPPAAL timed automata. The implementation is conceptually based on the Euler method for solving differential equations. Mathematically this method is less sophisticated than many other discretizations in the literature; the main contribution of our approach is that is has been able to handle systems with dimensions in the order of 100. This efficiency is a prerequisite for the use of model checking of biological systems, especially with the objective of using the generation of counterexamples as a tool for aiding drug synthesis.

The approach has been used in the tool ANIMO for biological signal network analysis. ANIMO has been (and currently is being) used by biologists in actual biological research. The user interface (based on the tool Cytoscape) enables biologists to create their own models, perform analysis and interpret the results, all without intervention from computer scientists. Current research concentrates on automatic model generation from libraries, on analyzing parameter sensitivity, and on generating model improvements automatically from experimental data.

Our approach is based on an arbitrary precision approximation. However, the approximation error is hard to quantify, and it seems hard to qualify the approximation as an abstraction (in terms of either an over or under approximation). A CEGAR [21] type approach seems recommended: if model checking produces a trace, then this trace should be checked against a refined version of the model, i.e. an approximation with a smaller step size. In addition, it would be interesting to try to apply error estimation techniques like the one in [13].

The Euler method is known to have potential stability problems, especially for so-called stiff equations. The analysis of stiffness properties is a notorious difficult problem in mathematics; in the future we would like to evaluate (and possibly improve) the stability properties of our implementation, possibly by using more advanced versions of the Euler method.

We certainly do not expect our approach to be applicable to the whole universe of nonlinear dynamical systems. The biological applications we have encountered so far could be described by multiaffine systems that posed no difficulties to our tools. In the future we would like to see whether our approach can be applied succesfully to other application areas. In addition we would like to obtain a better idea of the apriori limitations of our method.

**Acknowledgements.** We thank Arend Rensink for an important comment on an earlier version of this work. We thank Wim Bos, Liesbeth Geris, Marcel

Karperien, Johan Kerkhofs, Jaco van de Pol, Janine Post, Jetse Scholma, Ricardo Urquidi Camacho, Paul van der Vet, and Brend Wanders for the fruitful and pleasant collaboration leading to ANIMO.

## References

1. R. Alur and D. L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126:183–235, 1994.
2. R. Alur, T. Henzinger, G. Lafferriere, George, and G. J. Pappas. Discrete abstractions of hybrid systems. In *Proceedings of the IEEE*, pages 971–984, 2000.
3. W. Bos. Interactive signaling network analysis tool. Master's thesis, University of Twente, 2009.
4. P. Bouyer, N. Markey, N. Perrin, and P. Schlehuber-Caissier. Timed-automata abstraction of switched dynamical systems using control funnels. In *Formal Modeling and Analysis of Timed Systems: 13th International Conference, FORMATS 2015, Madrid, Spain, September 2-4, 2015, Proceedings*, pages 60–75, Cham, 2015. Springer International Publishing.
5. L. Brim, M. Češka, and D. Šafránek. Model checking of biological systems. In *Formal Methods for Dynamical Systems: 13th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2013, Bertinoro, Italy, June 17-22, 2013. Advanced Lectures*, pages 63–112, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
6. L. Brim, J. Fabriková, S. Drazan, and D. Safránek. Reachability in biochemical dynamical systems by quantitative discrete approximation. *CoRR*, abs/1107.5924, 2011.
7. J. Butcher. *Numerical Methods for Ordinary Differential Equations; 2nd ed.* Wiley, Chichester, 2008.
8. R. Carter and E. M. Navarro-López. Dynamically-driven timed automaton abstractions for proving liveness of continuous systems. In *Formal Modeling and Analysis of Timed Systems: 10th International Conference, FORMATS 2012, London, UK, September 18-20, 2012. Proceedings*, pages 59–74, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
9. E. Clarke. Model checking. In S. Ramesh and G. Sivakumar, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1346 of *Lecture Notes in Computer Science*, pages 54–56. Springer Berlin / Heidelberg, 1997.
10. Cytoscape 3 ANIMO app. `http://apps.cytoscape.org/apps/animo`.
11. A. David, J. D. Grunnet, J. J. Jessen, K. G. Larsen, and J. I. Rasmussen. Application of model-checking technology to controller synthesis. In *Formal Methods for Components and Objects: 9th International Symposium, FMCO 2010, Graz, Austria*, pages 336–351, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
12. A. David, K. G. Larsen, A. Legay, M. Mikučionis, D. B. Poulsen, and S. Sedwards. Statistical model checking for biological systems. *International Journal on Software Tools for Technology Transfer*, 17(3):351–367, 2015.
13. A. Donzé, B. H. Krogh, and A. Rajhans. Parameter synthesis for hybrid systems with an application to simulink models. In *Hybrid Systems: Computation and Control, 12th International Conference, HSCC 2009, San Francisco, CA, USA, April 13-15, 2009. Proceedings*, pages 165–179, 2009.
14. E. A. Emerson and E. M. Clarke. Characterizing correctness properties of parallel programs using fixpoints. In *Proceedings of the 7th Colloquium on Automata,*

*Languages and Programming*, pages 169–181, London, UK, UK, 1980. Springer-Verlag.
15. S. V. Goethem, J.-M. Jacquet, L. Brim, and D. Šafránek. Timed modelling of gene networks with arbitrarily precise expression discretization. *Electronic Notes in Theoretical Computer Science*, 293:67 – 81, 2013. Proceedings of the Third International Workshop on Interactions Between Computer Science and Biology (CS2Bio'12).
16. L. C. G. J. M. Habets and J. H. van Schuppen. Control of piecewise-linear hybrid systems on simplices and rectangles. In *Hybrid Systems: Computation and Control: 4th International Workshop, HSCC 2001 Rome, Italy, March 28–30, 2001 Proceedings*, pages 261–274, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
17. L. C. G. J. M. Habets and J. H. van Schuppen. Control to facet problems for affine systems on simplices and polytopes - with applications to control of hybrid systems. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 4175–4180, Dec 2005.
18. M. Jongerden, B. Haverkort, H. Bohnenkamp, and J. Katoen. Maximizing system lifetime by battery scheduling. In *39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2009*, Los Alamitos, June 2009. IEEE Computer Society Press.
19. S. Killcoyne, G. W. Carter, J. Smith, and J. Boyle. Cytoscape: a community-based framework for network modeling. *Methods in molecular biology (Clifton, N.J.)*, 563:219–239, 2009.
20. K. G. Larsen, P. Pettersson, and W. Yi. UPPAAL in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1:134–152, 1997.
21. E. M., O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *Proceedings of CAV*, pages 154–169, 2000.
22. O. Maler and G. Batt. Approximating continuous systems by timed automata. In J. Fisher, editor, *Formal Methods in Systems Biology*, volume 5054 of *Lecture Notes in Computer Science*, pages 77–89. Springer Berlin / Heidelberg, 2008.
23. P. T. Monteiro, D. Ropers, R. Mateescu, A. T. Freitas, and H. de Jong. Temporal logic patterns for querying dynamic models of cellular interaction networks. *Bioinformatics*, 24(16):i227–i233, 2008.
24. pplane web page. `http://math.rice.edu/~dfield/dfpp.html`.
25. S. Schivo, J. Scholma, H. B. J. Karperien, J. N. Post, J. C. van de Pol, and R. Langerak. Setting parameters for biological models with ANIMO. In E. André and G. Frehse, editors, *Proceedings 1st International Workshop on Synthesis of Continuous Parameters, Grenoble, France*, volume 145 of *Electronic Proceedings in Theoretical Computer Science*, pages 35–47. Open Publishing Association, April 2014.
26. S. Schivo, J. Scholma, P. E. van der Vet, M. Karperien, J. N. Post, J. van de Pol, and R. Langerak. Modelling with ANIMO: between fuzzy logic and differential equations. *BMC Systems Biology*, 10(1):56, 2016.
27. S. Schivo, J. Scholma, B. Wanders, R. Urquidi Camacho, P. van der Vet, M. Karperien, R. Langerak, J. van de Pol, and J. Post. Modelling biological pathway dynamics with Timed Automata. *IEEE Journal of Biomedical and Health Informatics*, 18(3):832–839, 2013.
28. S. Schivo, J. Scholma, B. Wanders, R. A. Urquidi Camacho, P. E. van der Vet, H. B. J. Karperien, R. Langerak, J. C. van de Pol, and J. N. Post. Modelling biological pathway dynamics with timed automata. In *12th IC on Bioinformatics and Bioengineering (BIBE 2012)*, pages 447–453. IEEE Computer Society, 2012.

29. J. Scholma, J. Kerkhofs, S. Schivo, R. Langerak, P. E. van der Vet, H. B. J. Karperien, J. C. van de Pol, L. Geris, and J. N. Post. Mathematical modeling of signaling pathways in osteoarthritis. In S. Lohmander, editor, *2013 Osteoarthritis Research Society International (OARSI) World Congress, Philadelphia, USA*, volume 21, Supplement, pages S123–S123, Amsterdam, April 2013. Elsevier.
30. J. Scholma, S. Schivo, J. Kerkhofs, R. Langerak, H. B. J. Karperien, J. C. van de Pol, L. Geris, and J. N. Post. ECHO: the executable chondrocyte. In *Tissue Engineering & Regenerative Medicine International Society, European Chapter Meeting, Genova, Italy*, volume 8, pages 54–54, Malden, June 2014. Wiley.
31. J. Scholma, S. Schivo, R. Urquidi Camacho, J. van de Pol, M. Karperien, and J. Post. Biological networks 101: Computational modeling for molecular biologists. *Gene*, 533(1):379–384, 2014.
32. H. Siebert and A. Bockmayr. Temporal constraints in the logical analysis of regulatory networks. *Theoretical Computer Science*, 391(3):258 – 275, 2008. Converging Sciences: Informatics and Biology.
33. C. Sloth and R. Wisniewski. Verification of continuous dynamical systems by timed automata. *Formal Methods in System Design*, 39(1):47–82, 2011.
34. C. Sloth and R. Wisniewski. Complete abstractions of dynamical systems by timed automata. *Nonlinear Analysis: Hybrid Systems*, 7(1):80 – 100, 2013. {IFAC} World Congress 2011.
35. O. Stursberg, S. Kowalewski, and S. Engell. On the generation of timed discrete approximations for continuous systems. *Mathematical and Computer Modelling of Dynamical Systems*, 6(1):51–70, 2000.
36. UPPAAL website. `www.uppaal.org`.
37. R. Urquidi Camacho. Modeling osteoarthritic cartilage with ANIMO: an executable biology approach to osteoarthritic signaling and gene expression. Master's thesis, University of Twente, the Netherlands, July 2013.
38. R. Wisniewski and C. Sloth. Abstraction of dynamical systems by timed automata. *Modeling, Identification and Control*, 32(2):79, 2011.
39. J. S. Xing, B. D. Theelen, R. Langerak, J. C. van de Pol, G. J. Tretmans, and J. P. M. Voeten. UPPAAL in practice: Quantitative verication of a RapidIO network. In J. P. Katoen, editor, *ISoLA 2010 - 4th IS On Leveraging Applications of Formal Methods, Verification and Validation*, LNCS 6416, pages 160–174. Springer, 2010.