Predicting Events for the Simulation of Hybrid Systems

Kerstin Bauer and Klaus Schneider
Embedded Systems Group
Department of Computer Science
University of Kaiserslautern, Germany

Abstract
The quality of the numeric simulation of hybrid systems highly depends on the capability of the simulator to detect discrete events during continuous evolutions. Due to the interaction of discrete and continuous dynamics, failures to detect such events may have catastrophic impacts on the global simulation. Current methods employ numeric algorithms/solvers for the underlying systems of differential equations, and therefore inherently suffer from the accumulation of numeric inaccuracies. In this paper, we propose a method for simulation of hybrid systems that is based on a combination of symbolic and numeric computations, which allows one to predict discrete events without accumulating numeric inaccuracies during the continuous evolution. Our method is applicable for a wide range of hybrid systems like the subclass of linear hybrid automata.

1. Introduction
Hybrid automata [2, 13] provide an appropriate paradigm for the modeling of so-called hybrid systems where continuous variables interact with discrete modes. Such models are frequently used in engineering embedded, robotic, avionic, and aeronautic systems [1, 3, 12, 28]. In hybrid automata, the interaction between discrete and continuous dynamics is naturally expressed by associating a set of ordinary differential equations (ODEs) to every state of a finite automaton.

Due to the combination of discrete and continuous dynamics, hybrid automata are hard to analyze since essentially all interesting verification problems are undecidable. In particular, the reachability problem is undecidable for most families of hybrid automata [2, 17, 18, 21], and the few available decidability results are built upon strong restrictions [4, 14].

As hybrid systems occur also in safety-critical areas, there is nevertheless the need to analyze the behavior of these systems. A frequent approach used here is the numeric simulation of such systems. The simulation of pure continuous systems [30] and pure discrete systems is thereby well understood [8, 23]: There exist various numeric simulation methods for systems of ODEs. Variable step-size approaches ensure that step sizes are chosen w.r.t. the continuous dynamics of the underlying system such that the resulting simulations meet predetermined accuracy requirements [10, 11, 24]. Efficient simulation methods for discrete systems mainly depend on the representation of such systems like finite state machines or Petri nets [27, 29].

However, similar to formal verification of these systems, the combination of discrete and continuous dynamics also leads to difficult problems for simulation [9, 22, 24]. Most approaches are based on discrete-time models that employ numeric solvers for evaluating the occurring ODEs until the next discrete transition has to take place. A major issue is thereby the so-called discrete event detection [19, 20] during continuous evolutions that determines the point of time when the next discrete transition must take place. Due to numeric inaccuracies, it may be the case that a discrete event is missing and that therefore the entire simulation is incorrect. For this reason, algorithms are developed to reduce the effects of numeric inaccuracies and to effectively detect discrete events [11, 19, 20, 31]. Another approach for the simulation of hybrid systems has replaced the time discretization by state variables quantization and a discrete event simulation model [5, 16, 25]. Further issues in the field of simulation of hybrid systems such as sensitivity analysis, zeno hybrid automata and chattering behavior are e.g. addressed in [5, 15, 31].

In this paper, we present a new approach for the simulation of hybrid systems. In contrast to existing numeric simulation tools, our approach does not exploit numeric ODE solvers to detect events for discrete transitions and therefore our approach does not suffer from the accumulation of numeric inaccuracies. Instead, our approach is based on a symbolic analysis and conservative estimations which produce only a constant numeric inaccuracy that is independent on the duration of the continuous step.

The paper is organized as follows: In Section 2, we demonstrate by a motivating example some problems that may arise by simulation of hybrid systems by
2. Motivating Example

The quality of a numeric simulation of a hybrid system highly depends on the capability of the simulator to detect discrete events during continuous evolutions, so that the desired discrete transitions take place instead of the further continuous behavior. Due to the interaction of discrete and continuous dynamics, failures to detect such events may have disastrous effects on the entire simulation.

To illustrate such an effect, consider the following simple example shown in Figure 2. In this example, we consider an object at height \( h(0) = 0 \) m below a ceiling at height \( h_{\text{ceil}} = 19.61 \text{m} \). The object has an initial velocity of \( v(0) = 19.62 \frac{m}{s} \) which is reduced due to gravity on earth with \( g = 9.81 \frac{m}{s^2} \) when time proceeds. According to Newtonian mechanics, we therefore have the following continuous behavior in the initial state of the hybrid system when time \( t \) proceeds:

\[
\begin{align*}
    v(t) &:= v(0) - g \cdot t \\
    h(t) &:= \int v(t) dt = v(0) \cdot t - \frac{1}{2} g \cdot t^2
\end{align*}
\]

This continuous behavior takes place until one of two possible discrete events takes place at a time \( t_0 \) that is to be determined: either (1) the object hits the ceiling, i.e., \( h(t_0) = h_{\text{ceil}} \) or (2) the velocity becomes zero, i.e., \( v(t_0) = 0 \). In case (1) should occur before (2) would occur, the system shall be reset to its initial values, i.e., it should start again from the initial position with velocity \( 19.62 \frac{m}{s} \) (which is unrealistic, but simplifies our example). If (2) should occur instead, the continuous behavior is continued.

One can easily see that (1) holds iff \( h_{\text{ceil}} - h(t_0) = 0 \) holds, which means \( \frac{1}{2} g \cdot t_0^2 - v(0) \cdot t_0 + h_{\text{ceil}} = 0 \) and therefore that \( t_0^2 - (4s) \cdot t_0 + \frac{h_{\text{ceil}}}{2g} = 0 \) holds. One can now compute that at time \( t_0 \approx 1.9548476359014104 \ldots \) the object hits the ceiling, and thus the same behavior should be repeated from this point of time as shown on the right of Figure 1.

The leftmost part of Figure 1 shows however a description of this system in the well-known simulation tool Matlab/Simulink. Depending on the simulation parameters, ODE solver and global time-span of the simulation, the simulator fails to detect the event of hitting the ceiling. For this reason, the Matlab/Simulink simulation results in completely different simulation runs, as depicted in the middle of Figure 1 where the ceiling is crossed, and the object reaches at time \( t_1 = 2s \) the highest point and hits the ground at time \( t_2 = 4s \).

3. Problem Description and Related Work

Figure 3 presents a graphical illustration of the behavior of a hybrid system during simulation. In each step, a set of differential equations \( \dot{x} = d_{\ell}(x) \) is active that defines the continuous flow of the system variables \( x \). These variables evolve w.r.t. the differential equations until a guard function \( g(x) \) is violated. In that case, a discrete transition takes place which leads to an immediate assignment of values of the variables and the initiation of new continuous dynamics \( \dot{x} = d_{\ell}(x) \) for the new discrete state.

Formally, a continuous transition is defined as follows. Let \( x = (x^1, ..., x^n) \) be the system variables of the

Figure 1: Simulink Model of our Motivating Example (left), the Erroneously Simulated Behavior (middle), and the Correct Behavior (right)
hybrid system\(^1\) and let \(x_0 \in \mathbb{R}^n\) be the valuation of these variables at the starting point of the continuous transition. Then, the continuous flow
\[
x: \mathbb{R}^n \times \mathbb{R}_0^+ \to \mathbb{R}^n, \quad (x_0, t) \mapsto x(x_0, t)
\]
is defined by the initial value problem
\[
\dot{x} = d(x), \quad x(0) = x_0.
\]

Guards for continuous transitions are given as boolean combinations \(G(x)\) of atomic guard functions
\[
g_i(x) = 0 \text{ or } g_i(x) \leq 0 \text{ with } g_i: \mathbb{R}^n \to \mathbb{R}.
\]
The continuous flow \(x(t), t \geq 0\) is released at the first point of time \(t \geq 0\), where the guard \(G(x(t))\) evaluates to true.

Current industry and academic tools for the modeling and simulation of hybrid systems are for example Matlab/Simulink, Modelica/Dymola, Shift, HyVisual and Charon. An overview of existing tools together with further references is given in [7]. These tools employ various numeric ODE solvers for the simulation of the continuous evolutions. To this end, the system variables are stepwise numerically integrated. Dynamic step sizes ensure that step sizes adapt themselves depending on the behavior of the continuous dynamics.

The handling of discrete events within continuous evolutions usually is divided into event detection and event localization phase. The event detection phase is formulated as a zero-crossing detection and determines whether a guard function \(g(x)\) changes its sign during a time step \(t_n \to t_{n+1}\) of the numeric integration. The localization phase now determines the discrete event more accurately, for example by bisection methods.

Due to numeric inaccuracies and the non-avoidable sparse sampling of evolution points, the zero-crossing detection encounters several problems [31].

The first, most obvious problem is that any numeric integration method yields unavoidable numeric inaccuracies, which can be fatal if the guard is set and continuous evolution hardly overlap or just touch each other.

Another problem is given by so-called even roots. If there is an even number of roots within one integration step, then no change of sign will be detected, and thus

1. For ease of notation, we consider also the discrete variables as continuous variables, which do not change their values during continuous evolutions.

the event will not be located at all. This problem occurs due to large step sizes and too ‘thin’ guard sets.

A way to reduce this problem has been proposed e.g. in [11, 26], where the dynamics of guard functions are integrated in the numeric integration process so that step sizes not only take the dynamics of the system variables into account, but also that of the guard functions.

Nevertheless, as long as numeric inaccuracies accumulate over time due to the numeric solving of the ODEs, no guarantees can be given that roots of guard functions will be found with only a predefined constant numeric inaccuracy.

Further research on the simulation of hybrid systems considers the problems of zero-behaviors and chattering behaviors, which we will not discuss here.

4. Predicting Events by Analytic Methods

In this section, we present a new approach for the zero-crossing detection. It is based on an algorithm to detect events with a constant precision that is independent on the duration of the continuous evolution. To this end, we proceed as follows: In Section 4.1, we present the general idea of our approach as well as the mathematical background needed to detect a zero-crossing of an atomic guard function. The algorithmic implementation of the idea is given in Section 4.2. In Section 4.3, we then consider arbitrary boolean combinations of atomic guard functions. We conclude by discussing advantages and disadvantages of our approach in Section 4.4.

4.1. General Idea

Many classes of hybrid automata only consider subclasses of ODEs which can be solved symbolically using algebraic methods. Such classes are e.g. linear hybrid automata, where the continuous dynamics are given in the form of linear ODEs.

Thus, it is possible to symbolically solve the ODEs such that in each location \(\ell\) of the hybrid automaton, the continuous dynamics are not encoded via a set of differential equations \(\dot{x} = d(x)\). Instead, the continuous behavior is directly given in the form of continuous functions
\[
x(t) = f_\ell(x_0, t)
\]
where \(x_0\) denotes the initial values and \(\ell\) the current location. Since the initial values \(x_0\) are known at the beginning of each continuous transition during simulation, we will omit \(x_0\) for ease of notation.

Consider now an atomic guard \(g_\ell \leq 0\) or \(g_\ell = 0\) enabled at this location. Replacing all occurrences of the system variables by their corresponding continuous function results in a guard function \(g_\ell(t)\) that only
depends on the time (the initial values \( x_0 \) of \( x \) are again omitted here). According to the definition of a continuous transition, we now need to find the first point of time \( t \geq 0 \), where \( g(t) \) evaluates to 0.

As \( g(t) \) is continuous and defined on the time interval \([0, \infty)\), the same holds for \( g(t) \). Thus, \( g(t) \) is Lipschitz-continuous within any finite interval \([t_1, t_2] \subseteq [0, \infty)\). This implies that the slope of \( g(t) \) within a given interval \([a, b]\) is limited by two constants \( \min, \max \in \mathbb{R} \). Together with the function values \( g(a) \) and \( g(b) \) at the endpoints of the interval, all function values \( g(t), t \in [a, b] \) are contained in a rhomboid as depicted below.

Using the containment of all function values within these rhomboids, it is possible to determine a superset of all possible candidate points for zero-crossing detection. Without loss of generality, we assume that the function \( g(t) \) is non-negative at \( g(a) \) (otherwise, we consider \( -g(t) \)), thus we need to detect the first point of time, where the guard \( g(t) \) reaches/crosses the x-axis.

For example, let us consider the case, where the maximal slope \( \max \) of \( g(t) \) is negative. All possible variants for the relative placement of the rhomboid and the x-axis are depicted in Figure 4. All candidates for zero-crossing are now enclosed by the interval

\[
[a + t_1, b - t_2]
\]

where

\[
t_1 \geq -\frac{g(a)}{\min}, \quad t_2 \geq \frac{g(b)}{\max}
\]

are conservative estimations for all four cases.

In the same manner, one can evaluate \( t_1 \) and \( t_2 \) for all combinations of \( \min, \max \) and \( g(b) \), which is summarized in Table 1. The correctness of the table entries is stated by the following lemma and can be verified using elementary calculus.

**Lemma 1:** Let \( g(t) : [a, b] \rightarrow \mathbb{R} \) be a continuously differentiable function with

- \( g(t) > 0 \)
- \( \min \leq \min_{t \in [a,b]} g(t) \)
- \( \max \geq \max_{t \in [a,b]} g(t) \)

Then, the summary given in Table 1 is correct, i.e.

1) All possible roots of \( g(t) \) lie in the interval

\[
[g(a) + t_1, g(b) - t_2]
\]

If no \( t_1, t_2 \) are given or the interval is empty, then \( g(t) \) has no roots in \([a, b]\).

2) The number of roots in the interval \([a, b]\) is correctly determined by

- \( 0: \) no roots
- \( \geq 0: \) an even number of roots, possibly none
- \( \geq 1: \) change of sign in \([a, b]\), thus at least one root

### 4.2. Implementation

The implementation of the ideas described in the previous section is based on three algorithms. The first algorithm \( \text{MinMax} \) computes lower and upper bounds for the function values of a function \( g \) within the interval \([a, b]\). The computation in \( \text{MinMax} \) is done recursively on the structure of the input function. Its correctness is straightforward and stated in the following Lemma.

**Lemma 2:** Let \( g : [0, \infty) \rightarrow \mathbb{R} \) be a function built with the basic expressions \( \sin, \cos, \exp, +, \ast, \text{power}(_-, n) \). Then, \( \text{MinMax}(g, [a, b]) \) returns lower and upper bounds for the function values of \( g \) in the interval \([a, b]\).

Thus, \( \text{MinMax}(g(t), [a, b]) \) computes lower and upper bounds for the minimal and maximal slope of \( g \). The second algorithm \( \text{FindRoots} \) simply implements the results depicted in Table 1 and is not presented here.

Finally, the last function zero-crossing is the main algorithm and combines these two functions in a kind of bisection process. zero-crossing repeatedly checks, whether there could be a zero-crossing in the current time interval. If so, this interval is recursively checked by bisection of the candidate subinterval computed in \( \text{FindRoots} \), otherwise the algorithm proceeds with the subsequent interval. For the next iteration step, the step size is halved, when recursive computations have been done, as a possible zero-crossing is expected. If no recursive function calls have been done and thus possibly no zero-crossing is in sight, the step size is doubled. Termination criteria of this algorithm are either that the tolerance computed in \( \text{FindRoots} \) is below the given threshold, or there exist no roots in the interval to be checked. Note here that naturally termination can only be guaranteed if the initial interval is finite.

Theorem 1 below summarizes the properties of the Algorithm zero-crossing.

**Theorem 1:** Let \( g \in C^1(\mathbb{R}_+^\ast) \) based on the functions \( \sin, \cos, \exp, +, \ast, \text{power}(_-, n) \). Let \( \Delta, \epsilon > 0 \) and let \([a, b] \subseteq \mathbb{R}_+^\ast \). Let furthermore the results of the function calls of \( \text{MinMax} \) and \( \text{FindRoots} \) be correct within a numeric inaccuracy of \( \epsilon \). Then, the following holds for algorithm zero-crossing:

1) If \( b \neq \infty \), then zero-crossing terminates.
2) If \( t_{\min} = \infty \), then there provably exist no roots for \( g \) in \([a, b]\).
3) If \( t_{\min} \neq \infty \) then:
   - There exists no root of \( g \) in the interval \([a, t_{\min}]\).
   - type = \( +-- \): In the interval \([t_{\min}, t_{\max}]\), there exists at least one root (change of signs).
- type = ++: In the interval \([t_{\text{min}}, t_{\text{max}}]\), there exists an even number of roots (possibly 0).
- \(\forall t \in [t_{\text{min}}, t_{\text{max}}]: |g(t)| < 3\epsilon\).

As a complete proof would be rather lengthy, we just give a sketch of the key points.

**Proof:** (Sketch)

- **Termination:**
  To prove termination, it suffices to show that the number of recursive function calls in each step is finite. Let \(g_{\text{max}}\) and \(g_{\text{min}}\) be maximal and minimal values of \(g\) in \([t_1, t_2]\) and let \(\text{min}\) and \(\text{max}\) be the maximal slopes. Then, it holds
  \[
  g_{\text{max}} \leq g(t_1) + (t_2 - t_1)\text{max}
  \]
  \[
  g_{\text{min}} \geq g(t_1) + (t_2 - t_1)\text{min}
  \]
  If \text{zero-crossing} is called recursively, then \(g_{\text{min}} \leq 0\). Thus,
  \[
  \text{max}\{g_{\text{min}}|, g_{\text{max}}|\} = (t_2 - t_1) \cdot \text{max}\{|\text{min}|, |\text{max} - \text{min}|\}
  \]
  By definition of MinMax, we have \(\text{min}_{\text{orig}} < \text{min}_{\text{rec}}\) and \(\text{max}_{\text{orig}} > \text{max}_{\text{rec}}\) in each recursive function call. Thus, \(\text{min}, \text{max}\) can be safely estimated by their initial evaluation and thus can be considered as constants. Additionally, because of bisection, \((t_2 - t_1)\) at least halves in each recursive step, thus also \(\text{tol}\). Considering now additionally the possible numeric inaccuracy while computing \(\text{min}, \text{max}\) and function evaluations \(g(t)\), we have that \(\text{tol}\) must be below \(3\epsilon\) after a finite number of steps.

- **Correctness:**
  The proof of the correctness can be divided into two parts:
  - \(\forall t \in [t_{\text{min}}, t_{\text{max}}]: |g(t)| < 3\epsilon\): obvious.
  - correctness of type: \(\text{type} = +-\) can only be assigned, if \(g(t_2) < -\epsilon\), then even considering the possible numeric inaccuracy of \(\epsilon\), \(g\) is negative at \(t_2\). Together with the fact, that \(g(t_1) \geq \epsilon\) (otherwise \text{zero-crossing} would have had to report that as a possible candidate the previous step), this guarantees a change of the sign.

### 4.3. Boolean Combination of Guard Functions

In the previous section, we described how to deal with atomic guard functions of the form \(g = 0\) or \(g \leq 0\). However, transitions can be guarded by any boolean combination \(G(x)\) of such atomic guards. In the following, we briefly discuss how to effectively adapt the above algorithms to deal with such boolean combinations. To that end, it suffices to consider disjunctions and conjunctions of guard functions.

A disjunction of guard functions is just processed in parallel, where the parallelism here refers to the considered time intervals. Note, however, that each guard is processed on its own speed as specified in \text{zero-crossing}.

The conjunction of guards is a bit trickier, as one needs to slightly adjust the existing algorithm. However, the adjustments only need to be done in the subroutine **FindRoots**:

1. One must additionally consider the (trivial) case where atomic guards can be directly evaluated to true.
2. candidate intervals are just the intersection of all candidate intervals of the atomic guard functions, as all participating guards must be fulfilled at the same time.

Thus, it is possible to effectively deal with boolean combinations of atomic guard functions in a way that...
Input: \( g : [0, \infty) \rightarrow \mathbb{R}, t \mapsto g(t) \) based on \( \sin, \cos, \exp, +, \ast, \text{power}(\_ , n) \)
Interval \([a, b] \)
Output: \( \text{min, max} \) with
\( \text{min} \leq \text{min} \in [a,b] \hat{g}(t) \)
\( \text{max} \geq \text{max} \in [a,b] \hat{g}(t) \)
Algorithm MinMax:
\- Case \( g = c \): Return \((c, c)\)
\- Case \( g = f \): Return \((a, b)\)
\- Case \( g = \sin(g) \vee g = \cos(g_1) \): Return \((-1,1)\)
\- Case \( g = \exp(g_1) \):
  \- \((\text{min, max}) = \text{MinMax}(g_1, [a,b])\)
  \- \text{Return} \((\exp(\text{min}), \exp(\text{max}))\)
\- Case \( g = \text{power}(g_1, n) \):
  \- \((\text{min, max}) = \text{MinMax}(g_1, [a,b])\)
  \- If \((n \text{ is odd})\) then Return \((\text{min}^n, \text{max}^n)\)
  \- else \( \ast \text{ max} = \max\{\text{min}^n, \text{max}^n\} \)
  \- \( \ast \text{ min} = \text{if} (\text{min} < 0 < \text{max}) \text{ then } 0 \)
  \- else \( \text{ min} = \min\{\text{min}^n, \text{max}^n\} \)
\- Case \( g = g_1 + g_2 \):
  \- \((\text{min}, \text{max}) = \text{MinMax}(g_1, [a,b]), i \in \{1, 2\}\)
  \- \text{Return} \((\text{min}_1 + \text{min}_2, \text{max}_1 + \text{max}_2)\)
\- Case \( g = g_1 \ast g_2 \):
  \- \((\text{min}, \text{max}) = \text{MinMax}(g_1, [a,b]), i \in \{1, 2\}\)
  \- \text{min} = \text{min}\{\text{min}_1 \ast \text{min}_2, \text{min}_1 \ast \text{max}_2, \text{min}_2 \ast \text{max}_1 \ast \text{max}_2\}
  \- \text{max} = \text{max}\{\text{min}_1 \ast \text{min}_2, \text{min}_1 \ast \text{max}_2, \text{min}_1 \ast \text{min}_2, \text{max}_1 \ast \text{max}_2\}
  \- \text{Return} \((\text{min}, \text{max})\)

Figure 5: Algorithm MinMax

the result will not be affected negatively. Independent guards will be processed at their own speed, while dependencies between guards are used to directly get finer interval interpolations.

4.4. Discussion

Our simulation method proposed above provides several advantages compared to simulations based on the numeric solving of ODEs.

The first and most important advantage is that the zero-crossing detection works provably correct w.r.t. a predetermined numeric inaccuracy \( \epsilon \) independent of guard sets, which are possibly difficult to deal with. That means, one can guarantee, that no zero-crossing is missed throughout the simulation as it may happen by the use of numeric ODE solvers due to unfortunate step-sizes. Furthermore, it is reported, when the zero-crossing detection only finds a candidate for roots, i.e., when the function values of the guards only are in the \( \epsilon \)-neighborhood, but not provably beyond 0.

The algorithm also is robust w.r.t. the boolean combination of guard functions. A further advantage is the small number of simulation parameters, which do not depend on the underlying continuous dynamics.

Of course, our approach cannot be applied to all classes of hybrid systems. However, important classes of hybrid systems such as linear hybrid automata satisfy the necessary requirements, thus the applicability is given for a broad range of real-world examples.
5. Example

The method of operation of our zero-crossing detection is best presented as a graphical illustration of the (recursive) steps of the algorithm zero-crossing. To this end, we consider again the moving object example introduced in Section 2. Figure 7 illustrates the first three (recursive) function calls of zero-crossing. The rhomboids depict the conservative estimation of possible function evaluations within the interval of interest. The computed intervals of interest are denoted via the points $t_l^i$, $t_r^i$, and $t_m^i$, respectively, where $l$ stands for 'left', $r$ for 'right' and $m$ for the intermediate point acquired via bisection.

6. Conclusions and Future Work

In this paper, we presented a new approach for predicting discrete events for the simulation of hybrid systems. Given a predefined numeric inaccuracy for function evaluations, we can predict discrete events with a constant inaccuracy, which is independent on the duration of the continuous evolutions. The resulting algorithm will be implemented in the simulator for the system description language Quartz, which has been recently extended to deal with hybrid systems [6].

References


