European Journal of Control (2001)7:1–16
© 2001 EUCA

European Journal of Control

Symbolic Verification of Hybrid Systems: An Algebraic Approach

Martin v. Mohrenschildt

Department of Computing and Software, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada L8S 4K1

In this paper we present a new symbolic, computer algebra based approach to hybrid systems. Hybrid systems are systems containing both, continuous and discrete changing quantities. As is commonly done, we model hybrid systems using hybrid automata. Hybrid automata extend the classical notion of finite state machines by combining differential equations to model the dynamic behavior of systems with a finite control. In contrast to other approaches we consider hybrid automata as a generalization of differential equations and develop the notion of an "explicit symbolic solution" of a hybrid automaton. An explicit symbolic solution is an expression which gives the value of the quantities in question, the state variables, as a function of the design parameters and time. These solutions allow us to perform the verification of design properties. We present an algorithm leading to an implementation which computes these explicit symbolic solutions. We were able to detect design constraints on control systems that other methods fail to detect. This paper gives the basic definitions, algorithms, and three examples to demonstrate the advantage of the proposed approach.

Keywords: Computer algebra; Differential equations; Hybrid systems; Hybrid system design; Symbolic solutions; System verification

Correspondence and offprint requests to: Martin v. Mohrenschildt, Department of Computing and Software, Faculty of Engineering, McMaster University, Hamilton, Ontario, Canada L8S 4K1. E-mail: mohrens@mcmaster.ca.

1. Introduction

Many engineered systems contain both traditional analog components and modern digital (finite state) components. Such mixed continuous-discrete systems are often called *Hybrid Systems*. Hybrid systems are studied in the control theory community as well as in the computer science community. It is essential to develop precise models for hybrid systems in order to develop and verify control software (hardware) for systems such as nuclear reactors, petrol chemical plants, or car engines. As one models systems using hybrid systems, it is essential that one is able to verify the properties of such systems, especially as hybrid systems are part of safety critical systems. The verification of safety critical properties of systems has to be exact, meaning formal, using logic or algebra; a numerical simulation of hybrid systems is not sufficient. A detailed overview and references of different applications of hybrid systems can be found in [21]. In literature one finds a variety of approaches to the verification of hybrid systems including model checking techniques and various approaches using logic and symbolic methods [2,18,20]. Even if many verification methods are quite successful and some can be automated we found that it is essential to develop symbolic methods that can be automated to assist system developers in the verification of safety properties of hybrid systems.

In this paper we present a symbolic, algebraic approach to hybrid systems deriving an algorithm which allows us to compute symbolic solutions of

Received 11 April 2001; Accepted ...;
Recommended by ...

hybrid systems. As commonly done in literature [1,2,4,12,14–16,18,19,21], we model hybrid systems using the so called *hybrid automata* which are essentially differential equations combined with a finite state control. Hybrid automata combine the regions of continuous state changes into modes and model them using differential equations. The discrete state changes are modeled using transition between these modes. The type of hybrid automata defined and used in this paper are a variation of those found in literature e.g. [1], and similar to the so called transition systems as defined in [18] but allowing more general differential equations. To be able to prove a uniqueness theorem we restrict ourselves to deterministic automata.

We have developed a theory which defines and allows us to compute expressions called *explicit symbolic solutions* of a hybrid automaton. An explicit symbolic solution of a hybrid automaton is defined as an explicit expression which, when evaluated, meaning interpreted as a function, determines the value of each of the state variables as the system evolves in time. The theory of explicit solutions of hybrid automata is in some sense an extension to the theory of unique solutions of initial value problems. As a mode is entered in a specific state the system is described by an initial value problem that has a unique solution. Obviously, the initial conditions to each mode are not given explicitly. Our idea is to derive a recurrence relation which, solved, gives the initial conditions to the differential equations in each mode. Using this information we are able to construct expressions which give the value of all state variables as a function of time. We give an algorithm which computes these explicit solutions. The presented algorithm is implemented into the computer algebra system Maple. We perform all computations exactly, not using floating-point approximations, hence all results are exact: "There are no mode switches due to rounding errors". To be able to represent these piecewise and periodic functions as algebraic expressions we have to extend basic algebraic structures to contain elements representing "discrete switches" [10,11] and periodic functions.

Our algorithm will not succeed for any hybrid automaton. Even for simple hybrid systems reachability, the question if a system will reach some specific set of states, is undecidable [14,19]. It is not our goal to define a class of decidable hybrid automata, such as in [14,19,21], but to develop a general algorithm that can be applied to many systems. One should keep in mind that many hybrid systems are designed with a specific behavior in mind. The designer intends a specific trace, the sequence of modes the system is in as time passes, and his goal is to verify that the system actually has the designed behavior.

The hybrid systems and hence the solutions of hybrid automata often contain design parameters, parameters that can be chosen in order to optimize the behavior of the system by minimizing or maximizing some *cost-function* as in control theory. The solutions as defined here, being symbolic expressions, allow us to *verify* design constraints and to optimize design parameters. Our approach to verification does in contrast to most other approaches not only verify given properties by proving, but allows us to derive properties of a given system. Further, in contrast to numerical solutions symbolic solutions are exact, they do not contain any rounding errors; rounding errors could trigger an unwanted discrete state change. The proposed approach differs from the logic based approach which normally use fix-point computation in order to derive formulas which hold in each mode the system is in. Our approach allows the verification of systems which do not converge, the state variables convert to a point in the state space. Fix-point computation methods cannot be applied to such systems.

This paper first gives a definition of hybrid automaton and then defines their semantics. Then the possible solutions are examined and a uniqueness theorem is given. In Section 4, we develop the algorithm used to compute the explicit solutions. We present three "classic" examples found in literature to demonstrate the presented approach.

2. Differential Equations with Piecewise Continuous Coefficients

Normally, the definition of a solution of the differential equation

$$y^{(n)} = f(y^{(n-1)}, \ldots, y, t)$$

assumes that f is Lipschitz continuous. But as we model real systems, especially engineering systems containing a discrete control, the differential equation modeling the behavior of the system can change in a discretely manner. These discrete changes can, in general, be triggered not only by conditions on time [9–11] but also by conditions on the solution itself. Also, the solutions of classical ordinary differential equations are *local*, meaning that the behavior of the system is defined by the shape of the vector field at the point of interest. Real systems can have a "finite memory", the vector field (given by the coefficients of the equations) can change based on a finite number of events from the past (e.g. hysteresis). We could even stop the system in any state (point of time) and then continue with some variables (initial states) changed in a discretely manner. Even the order of some differential equations could change (as in the example given later in the paper). Hybrid systems can perform actions; assignments to the state variables. New initial conditions can be given triggered by some conditions on the state of the system. The above shows that, in order to model hybrid systems we have to extend the notion of differential equations to hybrid differential equations, which are differential equations that can change their coefficients (form) obeying a finite state machine. We have to extend the classical theory of differential equations to a theory of differential equations containing elements that allow finite state control. (See [9], even though we call such differential equations piecewise continuous differential equations).

Defining and computing the solution of a hybrid differential equation is much more difficult than that of a continuous differential equation. The variety of possible behaviors is much larger. The classical uniqueness theorems for initial value problems require Lipschitz continuity of the equations. Hybrid systems are not Lipschitz continuous.

We approach hybrid differential equations by "transforming" [9] them into semantically equivalent (having the same solution) hybrid automata. These will be formally defined in the next section. Usually we do not even give differential equations such as $y'' + y = \text{signum}(y - y')$ but immediately give the corresponding hybrid automaton. A hybrid automaton splits the system into modes; in each mode the differential equation is continuous and all discrete state changes are performed as the mode is changed.

3. Hybrid Automata

We study systems containing several (continuous) variables changing continuously or discretely, in time. A state of a system is given as each of the variables has a specific value. Clearly, if we consider time as an integral component of our systems, (non-autonomous) the system will never be two times in the same state. The values of the variables of the systems can either change continuously in time, or instantly (discrete), to new values. The continuous changes of the variables are described by differential equations, the discontinuous changes are described by assignments. As is commonly done, we model such systems using hybrid automata. Hybrid automata are build using a finite set of modes and a finite set of transitions. Each mode contains a system of differential equations. The transitions between the modes are guarded with conditions and can contain assignments (actions).

Formally, a hybrid automaton is defined as follows:

Definition 3.1 (*Hybrid automaton*). Given are:

- A finite set of state variables $V = \{y_i\}$, $i = 1, \ldots, m$. All variables depend on time. We write $y_i(t)$ to denote the value of a variable y_i at time point t.
- A finite set of *modes* $S = \{s_i\}$, $i = 1, \ldots, n$. The mode s_i contains the system of m autonomous differential equations $y_j' = \Phi_{i,j}(y_1, y_2, \ldots, y_m, t)$, $j = 1, \ldots, m$. The differential equations $\Phi_{i,j}$ are Lipschitz continuous in the y_i. Without loss of generality, if needed, we assume that these are coupled first-order systems.[1] Derivatives are always derivatives by the time t. If a mode does not contain a differential equation in the variable y_i then we assume that the mode implicitly contains the equation $y_i' = 0$, the value of the state variable y_i is not changed.
- A set of *transitions* $T = \{t_{i,j}\}$, $1 \leq i,j \leq n$. The transition $t_{i,j}$, leads from mode s_i to mode s_j. A transition $t_{i,j}$ is a pair $(c_{i,j}, a_{i,j})$ of:
  - *Condition* $c_{i,j} \cdot c_{i,j}$ a boolean expression which can contain predicates depending on the variables y_i and time t. The conditions $c_{i,j}$ have to be closed, meaning that the sets

    $$C_{i,j} = \{(t, y_1, \ldots, y_m) | t, y_k \in \Re^{m+1}, \\ c_{i,j}(t, y_1, \ldots, y_m) = \text{true}\}$$

    have to be closed but not necessarily bounded.
  - *Actions* $a_{i,j}$. An action $a_{i,j} = \{y_{l_1} := c_{l_1}, \ldots, y_{l_k} := c_{l_k}\}$ is a set of assignments to variables of V, where the c_{l_1} are values. Actions are used to "trigger-events" or to give new initial conditions to the differential equations of the next mode.
- An unique initial transition $t_{0,i} = (\text{true}, a_{0,i})$ with an identical true condition and the action $a_{0,i} = \{y_1 := \alpha_1, y_1 := \alpha_1, \ldots, y_m := \alpha_m\}$, that initializes *all* variables of V.

A hybrid automaton is then given by the four-tuple:

$$H = (V, S, T, t_{0,i}).$$

Some aspects of the hybrid systems as presented here are kept general, e.g. we do not specify the form of the transition conditions or restrict the type of differential equations, since it is not our intent to

[1] Although we sometimes use higher order equations for convenience, the formal definition is based on first-order systems. Higher order ODEs can be converted into systems of first-order.

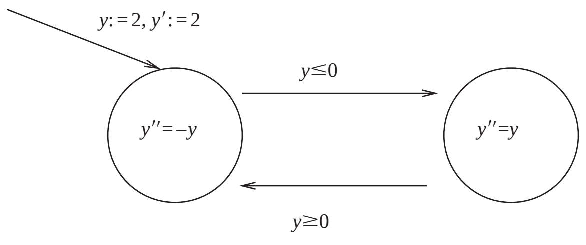


Fig. 1. Sample system.

define a class of decidable hybrid systems, as in [14,19], but to develop a general algorithm, knowing that the presented methods will fail for some types of hybrid systems. In Section 5.2.1 we present some necessary properties of hybrid systems such that our methods will succeed.

Often a graphical representation of hybrid systems is given such as the one in Fig. 1.

The hybrid automata defined here are a variation of those found in literature [1,4,7]. The main differences can be observed in the definition of the semantics, discussed in more detail later.

Definition 3.2 (*State of hybrid automata*). The state of a hybrid automaton is given by a time point t and the value of all state variables y_i at this time point t. We write $(t, y_1, \dots, y_m)$ to denote the state a time point t.

3.1. Semantics of Hybrid Automata

In order to give a deterministic semantics and, later, to be able to prove a uniqueness theorem for solutions of hybrid automata we restrict the general definition of hybrid automata.

Definition 3.3 (*Well-defined*). A hybrid automaton is well-defined if

- *Deterministic* In any mode s_i only one of the conditions $c_{i,j}$ can be true at any instance of time. This means that $C_{i,j} \wedge C_{i,k} = \{\}$, $i \neq j$, the conditions partition the state space in each mode.
- *No-zero-time* If a condition $c_{i,j}$ is true at time point t_k and the automaton changes to mode s_j, then there is no transition condition $c_{j,l}$, leading out of s_j true at time point t_k. We require that the system remains in each mode for a time t with $t > \varepsilon$. This implies that we cannot "pass through" a mode or else we could define an automaton which would never stay in any mode, and hence never proceed in time. (Such systems are commonly called Zeno systems).
- *Unique transition* There exists at most one unique transition $t_{i,j}$ leading from mode s_i to mode s_j.

Now, we are ready to give the semantics of a hybrid automaton. Our semantics can be called an *earliest time semantics* that is, the automaton changes mode at the earliest possible point of time. Other semantics found in literature, e.g. [1,4,7] allow mode changes at any point of time a transition condition is true given that some additional state invariant remains true, possibly leading to a nondeterministic behavior. We see the possibility of generalizing our approach in future work using intervals as solutions. This means that each possible solution will be enclosed by an interval with functions as boundaries. This approach would allow us to model some non-deterministic behavior.

With t_i we denote the time point where the automaton switches mode the ith time.

Definition 3.4 (*Semantics of a well-defined hybrid automaton*). Let H be a well-defined hybrid automaton.

- The automaton starts with the initial transition $t_{0,i}$, where all state variables are assigned a specific initial value. The time starts at zero, $t_0 = 0$.
- When entering the mode s_i at time point t_k we start the "dynamic system" defined by the differential equations in this mode s_i. The initial conditions are given by the value of the variables at time point t_k. The state variables y_i change now continuously in time obeying the differential equations of mode s_i. As soon, if, as a condition $c_{i,j}$ becomes true, meaning that
$$\begin{aligned} d = \min_d c_{i,j}(t_k + d, y_1(t_k + d), \dots, y_m(t_k + d)) \\ = \text{true}, \quad d > 0 \end{aligned}$$
exists, the automaton changes mode to s_j performing the assignments given in the action $a_{i,j}$ and $t_{k+1} = t_k + d$. If this minimum does not exist we remain in that mode forever; no further discrete transition will be performed.[2] Since the automaton is "no-zero-time" the system will stay for some time $d > \varepsilon$ in mode s_i.

Also, we restrict our examination to hybrid automata, where no variable can be changed by external "events". All changes of variables are performed by the hybrid automaton itself. Such automata are called closed hybrid automata.

[2]We assume that this mode switch happens instantly. A hybrid automaton where transitions take time can always be obtained by adding extra modes modeling the time a transition would take to perform.

Definition 3.5 (*Transition–Trace, Trace*). A *transition–trace* TT of a hybrid automaton is the (finite or infinite) sequence of transitions $t_{i_k,i_{k+1}}$ that the hybrid automaton performs as time passes. The *trace* T of a system corresponding to a transition–trace is the sequence of modes s_{i_k} in which the hybrid automaton is in as time passes.

We define the run corresponding to a trace of a hybrid automaton that gives the values of all variables as time passes.

Definition 3.6 (*Run*). The run R corresponding to the trace $T = s_{i_0}, s_{i_1}, s_{i_2}, \ldots,$ of a hybrid automaton is a sequence of pairs (t_k, f_k) where t_k is the time point at which the automaton switches from mode s_{i_k} to mode $s_{i_{k+1}}$ and f_k is a vector valued function, giving values for all state variables y_l, and satisfying the differential equations in mode $s_{i_{k+1}}$ with initial values $y(t_{k+1}) = f_k(t_k) \cdot f_0(t_0)$ is the initial condition of mode s_q given by the initial transition $t_{0,q}$.

3.2. Solutions and Uniqueness

As stated, our aim is to develop a theory of solutions of hybrid automata, which is similar to the solutions of initial value problems.

Following the idea of a solution to an initial value problem for differential equation and based on the semantics given in Section 3.4 we say that a hybrid automaton with initial transition $t_{0,i}$ defines a vector valued function $\bar{f}$ of time t:

$$\bar{f}(t) = (f_{y_1}(t), f_{y_2}(t), \ldots, f_{y_m}(t)).$$

Following the theory of differential equations, we call the function $\bar{f}$ the solution of the hybrid automaton.

Now we are able to state a uniqueness theorem for solutions of closed, well-defined hybrid automata.

Theorem 3.7 (*Unique Trace*, *Run*). The trace and run of a well-defined closed hybrid automaton is unique.

Proof. We point to the uniqueness results for initial value problems of Lipschitz continuous differential equations. The system enters mode s_i from some mode s_k at time point t_l or by the initial transition, using a transition $t_{k,i}$ performing the actions $a_{k,i}$. The values of all variables $y_1, \ldots, y_m$ at time point t_l give a unique initial condition for the system of differential equations $\Phi_{i,j}$ in mode s_i. Hence there is a unique continuous behavior of the state variables defined by the differential equations in mode s_i. The hybrid automaton is well defined; the "no-zero-time" condition guarantees that there is no transition condition leading out for this mode true at the point of time t_l, the time we enter the mode, even stronger, we will remain for at least ε in this mode. If the minimum

$$\min_d c_{i,j}(t_l + d, y_1(t_l + d), \ldots, y_m(t_l + d))$$
$$= \text{true}, \quad d > 0$$

exists then d is equal to this minimum, else we remain in mode s_i from now on. If the minimum exists we also obtain a $c_{i,j}$ and define $t_{l+1} := t_l + d$. The system takes the transition $t_{i,j}$ leading to mode s_j performing the action $a_{i,j}$. This leads again to a unique initial condition for the system of differential equations in the mode s_j. Hence, the sequence of modes s_{i_l}, and the times t_l where the automaton switches mode are unique for one hybrid automaton. □

Behaviors of solutions of hybrid automata We give the following classification of behaviors of traces of hybrid automata:

- *Finite Trace* The trace is finite. This means that the automaton enters some mode s_i at some time point t_k where $c_{i,j}(y_1, \ldots, y_m, t) = \text{false}$ for all $t > t_k$, it remains in that mode from now on,

  $s_{i_0}, s_{i_1}, \ldots, s_{i_n}.$

- *Periodic Trace* The trace is infinite, but periodic. This means that after an initial sequence of modes, the automaton cycles through a finite sequence of modes:

  $s_{i_0}, s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, s_{i_{k+2}}, \ldots s_{i_{k+p}}, s_{i_{k+1}}, s_{i_{k+2}}, \ldots.$

- *General Trace* The trace is infinite and not periodic.

Note that a periodic trace does not imply that the values of the variables are periodic, only that after a finite sequence of modes, we cycle through a fixed sequence of modes.

With the methods presented here we can only solve systems with solutions falling into the first two classes of behaviors.

4. Transformations of Hybrid Automata

We present a transformation of hybrid automata which we will use later to simplify our algorithms.

Definition 4.1 (*Equivalence*). Two hybrid automata H and $\tilde{H}$ are equivalent if the functions f defined by H and $\tilde{f}$ defined by $\tilde{H}$ are the same for each possible action in the initial transition $t_{0,i}$.

Theorem 4.2 (*Single condition, action entrance*). Given a hybrid automaton H, we can construct a hybrid automaton $\tilde{H}$ where all transitions $t_{i,j}$ leading to any mode s_i have the same condition and same action:

$$\forall j, k t_{j,i}, t_{k,i} \in T c_{j,i} = c_{k,i} \wedge a_{j,i} = a_{k,i}$$

and H and $\tilde{H}$ are equivalent.

Proof. Let s_i be a mode with two transitions $t_{j,i} t_{k,i}$ with different conditions or different actions leading to it. We add a new mode $s_{\tilde{i}}$. The transition $t_{j,i}$ is changed to the transition $t_{j,\tilde{i}}$ ($t_{j,i}$ is deleted) and for each transition $t_{i,k}$ we add a new transition $t_{\tilde{i},k}$ with identical condition $c_{\tilde{i},k} = c_{i,k}$ and identical action $a_{\tilde{i},k} = a_{i,k}$. Clearly, mode s_k will have additional new transitions leading to it, but their conditions and actions are identical. We continue until all modes have only transitions with identical conditions and actions. The process will terminate, since we will not add any new "problem transitions" and the number of transitions is finite. □

Transforming a hybrid system will reduce the number of possible initial conditions in each mode. This will be very useful in the algorithm computing the solution of a hybrid automaton.

5. Computing Solutions and Verification

Our goal is to compute a closed-form expression which will represent the behavior of a hybrid automaton as a function of time. In contrast to other approaches to the verification of hybrid automata, we define and compute explicit solutions of hybrid automata. This means that we find symbolic expressions which compute the value of all state variables of the hybrid system as a function of time in a systematic manner that can be automated. The presented approach supports a symbolic verification. As design parameters are symbolic constants we are able to detect (compute) constraints on the design parameters in order to satisfy constrains put on the hybrid system. Clearly, due to undecidability results for hybrid automata we are not able to compute the solution of each hybrid automaton.

5.1. Expressions, Symbolic Solutions

Our goal is to give explicit representations of solutions. We define EXP, the set of expressions which we use to represent "solutions" of hybrid automata. In order to represent solutions of hybrid automata we have to enlarge the expressions as used in standard symbolic computation systems.

We start with the polynomials, formed using the variables y_i, formal parameters p_i, time t, rational numbers, and the function symbols $+, -, *$. Further, as needed, we add transcendent extensions such as e, ln, sin, cos. For expressions $t_1, t_2, \ldots, t_n$ and t we write $t \circ \sigma = t \circ \{y_{i_1} \to t_{i_1}, y_{i_2} \to t_{i_2}, \ldots\}$ to represent the simultaneous substitution of the variables y_{i_k} with the expressions t_l.

Next we extend EXP using three additional expression constructors.

- Piecewise Continuous Functions In [8] a decidable theory of piecewise continuous functions is presented. A piecewise continuous function is defined using piecewise expressions, expressions of the form:

  $$\texttt{piecewise}(c_1, f_1, c_2, f_2, c_3, f_3, \ldots, f_n)$$

  where the c_i are conditions, and the f_i are expressions. In some state the value of the expression f_i is selected if the condition c_i is true in that state, else, if none of the c_i is true, then the value of f_n is returned.[3] Piecewise expressions extend EXP such that we can represent discrete changes of functions.
- Periodic and Generalized Periodic Expressions In order to give expressions which represent periodic functions such as the saw-wave function we provide two additional constructs for EXP. A function f is called periodic if for some p, the period,

  $$\forall x \ \forall k \in \mathbf{Z} \quad f(x) = f(x + kp).$$

We define the following two helper functions in order to define the symbolic representations of periodic functions.

Definition 5.1 (*Integer divider and reminder*). We define $\texttt{irem} : \mathbf{R} \to \mathbf{R}^+$ and $\texttt{idiv} : \mathbf{R} \to \mathbf{Z}$

$x \ \texttt{idiv} \ p = k,$
where $k \in \mathbf{Z}$ such that $pk \leq x < p(k+1)$

$x \ \texttt{irem} \ p = y,$
where y such that $0 \leq y < p, \ y = x - p \ (x \ \texttt{idiv} \ p)$

We represent periodic expressions using the following notation $\texttt{per}_{[f,p]}$. So $\texttt{per}_{[f,p]}(t) = f(t \ \texttt{irem} \, p)$ which is commonly called the periodic extension of f.

We generalize further and even allow the period of a function to change.

Definition 5.2 (*Generalized periodic expressions*). A function $f : \mathbf{R} \to \mathbf{R}$ is represented by a *generalized*

[3]If two conditions are true at the same time, then the first in the sequence of conditions is chosen.

periodic expression with period p if there exists an expression $g : \mathbf{R} \times \mathbf{Z} \to \mathbf{R}$ with

$$f(x) = g(x \texttt{ irem } p, x \texttt{ idiv } p),$$

we then write $f = \texttt{gen\_periodic}_{[g,p]}$

Generalized periodic expressions allow us to represent a large class of "behaviors" of hybrid automata in explicit form. Such a function is, for example, $x \,\texttt{idiv}\, p$, the "stairway to heaven" function.

Now we are able to give the closed form representation of many different functions, e.g. the sawwave function with period 1: $\texttt{per}[\texttt{piecewise}(x < \frac{1}{2}, 1, 0), 1]$.

- Symbolic Solutions With a symbolic solution of a differential equation $y' = f(y, t)$ we mean an explicitly given expression y depending on t as defined in [3,17].

5.2. Computing Explicit Symbolic Solutions

We finally define what we mean with a symbolic solution of a hybrid automaton:

Definition 5.3 (*Solution of a hybrid automata*). Given a hybrid automaton H with m variables. Let $\bar{f}$ be a m-dimensional vector valued function defined by this automaton (3.7). An expression of exp of EXP with $\exp(t) = \bar{f}(t)$ is called a solution of this hybrid automaton.

Now we are ready to develop our algorithm to compute explicit solutions of hybrid automata. The key idea behind our approach is the following: to derive transformations which map the value of all state variables and time, as some mode is entered, into the value of all state variables and time as this mode is exited. These transformations then, in the case of a periodic trace, lead to a recurrence relation, which, solved allows to compute the value of the state variables as any mode in the cycle is entered depending on the number of repetitions the system passed through the cycle.

The computation of the solution is performed in four steps: (1) Solve the differential equations, determine the time spent in each mode and derive state transformations, (2) compute the possible traces of the system, (3) for periodic traces derive a recurrence relation that computes the initial conditions of the mode in the period, (4) construct the explicit solution. We describe each step of the algorithm in general. In the next section we will identify restrictions and the level of automation we can achieve with our implementation of our algorithm. Given a hybrid automaton H we first transform H to a single condition–action–entrance automaton $\tilde{H}$ as given in Theorem 4.2.

1. Solving Step As some mode s_i is entered in some state $(t, y_1, \ldots, y_m)$ at some time point t, we compute the sate of the system as this mode s_i is exited together with the time the system remained in this mode. This is done symbolically since entering time and the value of some state variables might not be known, resulting in the state transformation which maps entering states, initial conditions, to exit states. Specifically:
   - Compute the solution (as defined in Section 5.1) $f_{i,y_i}(t, y_1, \ldots, y_m)$ of the the initial value problem $y_j' = \Phi_{i,j}(t, y_1, \ldots, y_m)$ with $y_j(t_{s_i}) = y_{j,s_i}$ in each mode s_i where the starting time t_{s_i} is symbolic and all $y_j(t_{s_i})$, not fixed by the conditions or actions leading to this mode s_i, are symbolic. This means the value of $y_j(t_{s_i})$ is c if $t_{i,j}$ contains the assignment $y_j := c$, and $y_j(t_{s_i}) = y_{j,s_i}$ otherwise. This results in the substitution $\mathrm{sol}_i = \{y_i \to f_{i,y_j}\}$, meaning that if mode s_i is entered at time point t_{s_i} with initial conditions $y_j(t_{s_i})$ then, the value of the state variables after some time $t > t_{s_i}$ is given by the substitution sol_i.
   - For each mode s_i and for each transition $t_{i,j}$ leading form s_i to some other mode s_j we compute the time points trans_{s_i,s_j}, the time points at which the mode is exited, by solving the equation

     $$c_{i,j} \circ \mathrm{sol}_i = \mathrm{true}.$$

     These time points will depend on the parameters introduced as the initial conditions for this mode and time. If an equation $c_{i,j} \circ \mathrm{sol}_i = \texttt{true}$ has several solutions we determine the smallest of them which is larger then t_{s_i} so $\mathrm{trans}_{s_i,s_j} > t_{s_i}$ (earliest time semantics). We would like to point out that for some systems, containing design parameters, we detect restrictions of these parameters as we solve $c_{i,j} \circ \mathrm{sol}_i = \texttt{true}$, since often a solution only exists if certain conditions on the design parameters hold.
2. Trace Determining Step First we determine all possible loops of the hybrid automaton (a loop is a repeating sequence of modes). Note, our hybrid automaton $\tilde{H}$ is a single–action–entrance automaton. We only consider loops which enter each mode once. The idea here is that any loop can be at most as long as the number of modes the system has (analogous to the "pumping lemma" for finite state machines). We represent the possible traces as

sequences of modes:

$$s_{i_0}, s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, s_{i_{k+2}}, \ldots, s_{i_{k+p}}, s_{i_{k+1}}, s_{i_{k+2}}, \ldots$$

In the next step we will have to decide which of the possible traces are actually taken. This problem is, in general undecidable but many practical hybrid automata (e.g. control systems) are constructed with a periodic trace in mind.

3. Recurrence Step For all periodic traces found in step (2), we use the sequence of transformations in each mode computed in (1) to derive a recurrence relation. The solution of this recurrence relation gives the value of the state variables as we re-enter the first mode in the loop depending on the state the system was in as we entered the first mode the last time. Hence for the periodic trace

$$s_{i_0}, s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, s_{i_{k+2}}, \ldots, s_{i_{k+p}}, s_{i_{k+1}}, s_{i_{k+2}}, \ldots$$

we derive the coupled system of recurrence equations

$$f_{i_{k+1},y_i}(n+1) = f_{i_{k+1},y_i}(n) \circ \mathrm{sol}_{i_{k+p}} \circ \cdots \circ \mathrm{sol}_{i_{k+2}} \circ \mathrm{sol}_{i_{k+1}}$$

where $f_{i_k,y_i}(n)$ denotes the value of y_i as we enter the mode $s_{i_{k+1}}$ the nth time. Now we proceed as follows:

- Solve the recurrence equations. The solution will depend on n and $f_{i_k,y_i}(0)$.
- Compute the remaining initial conditions of the modes $s_{i_{k+j}}$, $j = 2, \ldots, p$, using the solution of the recurrence equations.
- Compute the actual period `per` of the automaton, by computing the time needed for one loop of the system.

$$\texttt{per} = \mathrm{trans}_{s_{i_{k+1}}, s_{i_{k+2}}} + \mathrm{trans}_{s_{i_{k+2}}, s_{i_{k+3}}} + \cdots + \mathrm{trans}_{s_{i_{k+p}}, s_{i_{k+1}}}$$

The period may be generalized, meaning it could depend on n.

- Compute the value of $f_{i_k,y_i}(0)$ by following the sequence of modes $s_{i_0}, s_{i_1}, \ldots, s_{i_k}$ leading to the start of the period:

$$f_{i_k,y_i}(0) = \{y_i(t_{n-1}) \to \mathrm{sol}_{n-1}(t_{n-1})\} \circ \cdots \circ \{y_i(t_0) \to y_{i,t_0}\}$$

4. Solution Construction Step Equipped with all the pieces needed to construct the solution we proceed as follows:

- Finite Trace Given a finite trace we can easily construct a piecewise defined expression which is the solution of the hybrid automaton. For the trace

$$s_{i_0}, s_{i_1}, s_{i_2}, \ldots, s_{i_n}$$

we construct the expression

$$\texttt{piecewise}(t < \mathrm{trans}_{s_{i_0}, s_{i_1}}, \mathrm{sol}_1 \circ \{y_i(t_0) \to y_{i,t_0}\},$$
$$t < \mathrm{trans}_{s_{i_0}, s_{i_1}} + \mathrm{trans}_{s_{i_1}, s_{i_2}, s_{i_1}},$$
$$\mathrm{sol}_2 \circ \{y_i(t_1) \to \mathrm{sol}_1(t_1)\} \circ \{y_i(t_0) \to y_{i,t_0}\}, \ldots,$$
$$t > \mathrm{trans}_{s_{i_0}, s_{i_1}} + \cdots + \mathrm{trans}_{s_{i_{n-1}}, s_{i_n}},$$
$$\mathrm{sol}_n \circ \{y_i(t_{n-1}) \to \mathrm{sol}_{n-1}(t_{n-1})\} \circ \cdots$$
$$\circ \{y_i(t_0) \to y_{i,t_0}\})$$

where $y_i(t_0)$ is the value of the state variables y_i at time point t_0 given by the initial transition.

- Periodic Trace For a periodic trace

$$s_{i_0}, s_{i_1}, \ldots, s_{i_k}, s_{i_{k+1}}, s_{i_{k+2}}, \ldots, s_{i_{k+p}}, s_{i_{k+1}}, s_{i_{k+2}}, \ldots$$

we first construct the expressions f_{init} which computes the value of the state variables for time init_t leading to the period $s_{i_0}, s_{i_1}, \ldots, s_{i_k}$ and the expression f_{cyc} which computes the value of the state variables for one period using the piecewise construct as described for finite traces above. Then we combine these two expressions obtaining the final solution

$$\texttt{piecewise}(t < \mathrm{init}_t, f_{\mathrm{init}},$$
$$\texttt{gen\_periodic}_{[f_{\mathrm{cyc}} \circ \{t \to \mathrm{trans}_{s_{i_1}} + \cdots + \mathrm{trans}_{s_{i_k}}\}, \mathrm{per}]})$$

where `per` is the period so the system.

We implemented these steps into the computer algebra system Maple. Step one, three and four are automated, step two needs user input to select candidates from the possible traces. The total code consists of about 700 lines of Maple code, which seems small, but Maple is powerful; we can solve an initial value problem in one line. The Maple code can be obtained from the author upon request.

5.2.1. Limitations of Algorithm and Existence of Symbolic Solutions

Obviously, our approach does not allow to compute closed-form solutions for all systems. It is not our intention to define a class of systems where our algorithm is guaranteed to succeed, but the presented algorithm is a semi algorithm, which means that if it terminated then it found a solution. In the following we discuss necessary conditions for the closed-form solution to be generated by our algorithm.

- *Differential Equations* As required in [18] we have to be able to generate closed-form differentiable expressions for the differential equations in each mode of our system.
- *Trace* We have to be able to determine the trace, finite or periodic, of the system. Even one can determine candidates for periodic traces by determining all possible loops, it in general undecidable which trace is actually taken. One could verify if some candidate is actually the periodic trace of the system by constructing the solution corresponding to this trace and if possible to verify that the times *trans* and the period per of the system are positive. For certain classes, e.g. Linear hybrid automata, as identified in [14,19] using a Bisimulation algorithm, the possible traces are decidable. We like to point out that the designer of a hybrid system designs his system with some specific behavior in mind; the trace the system is supposed to take is defined beforehand.
- *Exit times* The exit times for each mode have to be computed. As mentioned, sometimes the solver of Maple is not able to find a solution, but need additional assumptions on the parameters, e.g. the range of some parameter. Again, this is decidable for linear hybrid automata.
- *Recurrence Relation* We have to compute a closed-form solution for the recurrence relation generated for a periodic trace. Maple is able to do this for linear recurrence relations.

5.3. Verification

A closed form solution of a hybrid automaton allows to verify properties of the system. With verification we mean that given safety properties in the form of constraints on the state variables, e.g. the value of some state variable has to stay within some given range, or a predicate on the state variables, we verify that these properties remain true for all times.

Some examples of verifications are given in the water level control example (Section 6.1) and the reactor control example (Section 6.2).

6. Examples

In the following we give three examples of hybrid systems which we found in literature and apply the here presented approach. We present each step of our algorithm to demonstrate our approach even though we are able to perform these computations fully automated. The example show clearly that the computations could not (easily) be performed by hand and the advantage of using a symbolic algebra system such as Maple. All computations are done exactly, no floating-point approximation is done.

6.1. Example: Water Level Control

The water level control example can be found in several places in literature [1,12,18,19]. Although, it is a simple system, it allows us to demonstrate our methods. By verifying the safety properties of the water level control system we were able to detect constraints on the design parameters which we did not find in literature.

The system consists of a reservoir which has an intake valve and an out-flow. The intake valve can be opened or closed. Clearly valves do not open or close instantly, opening or closing takes time. The control has to be designed so that the reservoir is never empty and never overflows at any point in time.

We can easily identify four modes of the system, (1) the valve is open, (2) the valve is closing, (3) the valve is closed, and (4) the valve is opening. In each of these modes the change of the water level y is given by a differential equation. A graphical representation is given in Fig. 2.

- Mode 1 $y'(t) = a_1$ where a_1 is the rate the reservoir fills if the intake valve is open.
- Mode 2 $y''(t) = a_2$ where a_2 is the closing rate of the valve.

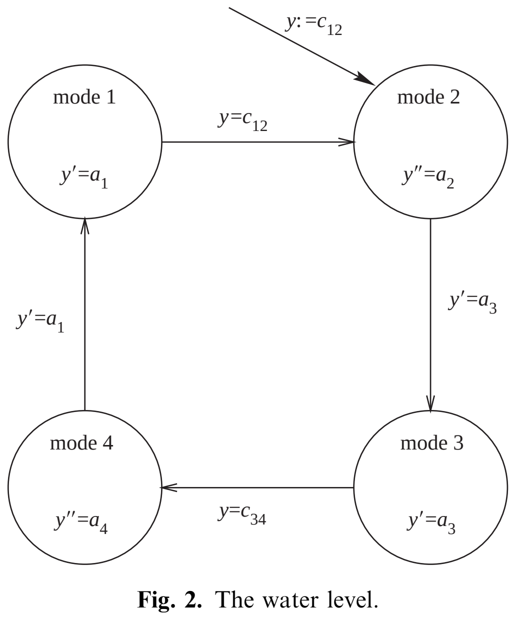


Fig. 2. The water level.

- Mode 3 $y'(t) = a_3$ where a_3 is the outflow-rate of the reservoir as the intake valve is closed.
- Mode 4 $y''(t) = a_4$ where a_4 is the opening rate of the valve.

In contrast to other approaches we actually model the dynamic phase of opening and closing the valve using a second order differential equation and do not just say it takes n seconds to open the valve. The two water levels that trigger to close the valve or to open the valve are called c_{12} and c_{34}. The water levels after opening (closing) the valves are called c_{23} and c_{41}, but they will cancel out in the calculation and are just used to set up the symbolic solutions of the differential equations. The transitions of our system are the following: $t_{0,2} = (\texttt{true}, y = c_{12})$, $t_{1,2} = (y = c_{12})$, $t_{2,3} = (y' = a_3)$, $t_{3,4} = (y = c_{34})$, $t_{4,1} = (y' = a_1)$, our automaton does not contain any actions (which would, in fact, make it simpler to solve).

Assuming that $c_{12} < c_{34}$, $a_3 < 0 < a_1$ and $a_2 < 0 < a_4$ as first design constraints we observe that the trace is periodic, cycling through modes 1, 2, 3, 4. We actually do not have to determine this trace, since the entire system is designed with this trace in mind. The conditions on the design parameters are made so that the system will actually have this trace.

"Safety properties" of the system

1. The reservoir is never empty, $\forall t\ y(t) > 0$.
2. The reservoir will never overflow $\forall t\ y(t) < \max$, where max is the maximal allowed water level in the reservoir.

We can immediately observe the following design constrains: (1) $0 \le c_{12} \le \max$ and (2) $0 \le c_{34} \le \max$, the sensors which cause the valve to open and close have to lie within the range of the reservoir.

Now, given some a_1, a_2, a_3, a_4 (flow rates and valve closing speeds) we have to determine the conditions on c_{12}, c_{24} and max such that the safety requirements are met.

6.1.1. Solve Differential Equations

We start in mode one, and solve the differential equations with the initial conditions $y'(t) = a_1 y(s_1) = c_{41}$ resulting in

$$f_1 = a_1 t - a_1 s_1 + c_{41}.$$

To switch from mode one to mode two, the water level has to pass the c_{12} mark. Solving, we find that this happens at time point

$$\mathrm{trans}_{1,2} = \frac{a_1 s_1 - c_{41} + c_{12}}{a_1}.$$

Now we are in mode two. The water level is c_{12}. The initial flow rate of the water is a_1 since we are coming from mode one. We solve the differential equation $y''(t) = a_2$ under the initial conditions $y(s_2) = c_{12}$, $y'(s_2) = a_1$ obtaining

$$f_2 = 1/2 a_2 t^2 + (-a_2 s_2 + a_1)t + 1/2 a_2 s_2^2 - s_2 a_1 + c_{12}.$$

We have to determine the time point at which the valve is closed. This is the case if $y'(t)$ is a_3. Computing the derivative and solving the resulting equation we receive

$$\mathrm{trans}_{2,3} = -\frac{-a_2 s_2 + a_1 - a_3}{a_2}.$$

Now, in mode three, we have to solve the differential equation $y'(t) = a_3$, we know the water level as we enter this mode by using the previously computed solution and time point.

$$f_3 = 1/2 a_2 t^2 + (-a_2 s_2 + a_1)t + 1/2 a_2 s_2^2 - s_2 a_1 + c_{12}.$$

We will switch to mode 4 as the water level reaches c_{34}, we solve the equation to determine this time point.

$$\mathrm{trans}_{3,4} = 1/2 \frac{2 a_3 s_3 a_2 + a_1^2 - a_3^2 - 2 c_{12} a_2 + 2 c_{34} a_2}{a_2 a_3}.$$

Finally, we are in mode four. Similar to mode two we solve the equation $y''(t) = a_4$ with the initial conditions $y(s_4) = c_{34}$ and $y'(s_4) = a_3$ and determine the point of time at which $y'(t) = a_1$, at which the valve is closed again,

$$f_4(t) = 1/2 a_4 t^2 + (-a_4 s_4 + a_3)t + 1/2 a_4 s_4^2 - s_4 a_3 + c_{34},$$

and the time point where the valve is closed is

$$\mathrm{trans}_{4,1} = \frac{a_4 s_4 - a_3 + a_1}{a_4}.$$

Step one is completed, we computed the solutions of all ODS's and the times for all modes.

6.1.2. Construct Recurrence Relation

Using the previously computed solutions we are able to find a recurrence relation which gives the water level in mode 1 depending on the water level as we entered mode 1 the last time in one period.

$$
\begin{aligned}
&ft(n+1)\\
&:= \left(\frac{1}{2}a_4\left(2a_3\left(-\frac{1}{2}\frac{a_2(-2a_1ft(n)a_4-a_3^2+a_1^2+2c_{34}a_4-2c_{12}a_4)}{a_4a_1}\right.\right.\right.\\
&\qquad \left.\left. - a_1 + a_3\right) + a_1^2 - a_3^2 - 2c_{12}a_2 + 2c_{34}a_2\right)\\
&\qquad \left. \Big/ (a_2a_3) - a_3 + a_1\right) \Big/ a_4.
\end{aligned}
$$

Solving the recurrence relation and computing the water level as we enter state 2, 3 and 4 we have all the pieces and can construct the general solution.

$$
\mathtt{per}\begin{cases}
a_1 t + \frac{1}{2}\frac{-a_3^2+a_1^2+2c_{34}a_4}{a_4}, & t < -\frac{1}{2}\frac{\%1}{a_4a_1},\\
\frac{1}{2}a_2t^2 + \left(\frac{1}{2}\frac{a_2\%1}{a_4a_1} + a_1\right)t + \frac{1}{8}\frac{a_2\%1^2}{a_4^2a_1^2} + \frac{1}{2}\frac{\%1}{a_4}, & t < \frac{\%2}{a_2},\\
a_3 t + \frac{1}{2}\frac{-2a_3\%2 - a_1^2 + a_3^2 + 2c_{12}a_2}{a_2}, & t < \frac{1}{2}\frac{\%3}{a_2a_3},\\
\frac{1}{2}a_4t^2 + \left(-\frac{1}{2}\frac{a_4\%3}{a_2a_3} + a_3\right)t + \frac{1}{8}\frac{a_4\%3^2}{a_2^2a_3^2} - \frac{1}{2}\frac{\%3}{a_2}, & \text{otherwise.}
\end{cases}
$$

$$
\begin{aligned}
\%1 &:= -a_3^2 + a_1^2 + 2c_{34}a_4 - 2c_{12}a_4\\
\%2 &:= -\frac{1}{2}\frac{a_2\%1}{a_4a_1} - a_1 + a_3\\
\%3 &:= 2a_3\%2 + a_1^2 - a_3^2 - 2c_{12}a_2 + 2c_{34}a_2
\end{aligned}
$$

A plot of the water level using some sample numbers can be found in Fig. 3.

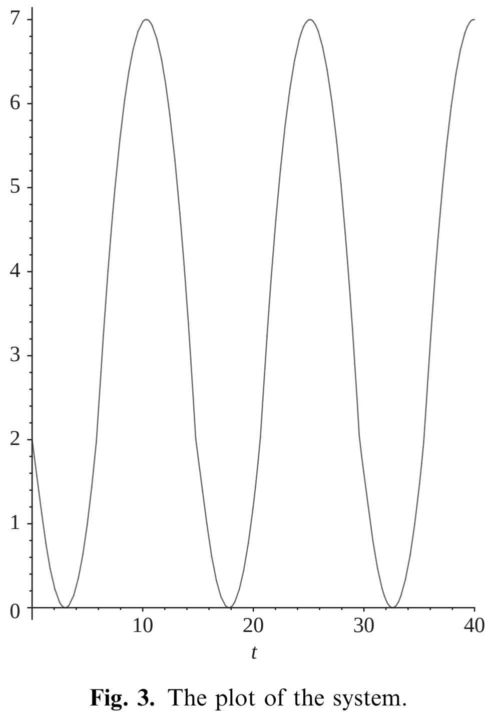


Fig. 3. The plot of the system.

6.1.3. Verification

Now we verify the safety conditions of the system with the goal to derive conditions on the design parameters of the system.

1. Overflow: We verify under which conditions the water reservoir would overflow. For this, we compute the maximum of the water level using the derivative as defined in [10], resulting in

   $$\frac{1}{2}\frac{2c_{34}a_4 - a_3^2}{a_4}.$$

   The maximal water level has to be smaller than the maximum the reservoir can hold, we solve this resulting condition for c_{34}

   $$c_{34} < \frac{1}{2}\frac{a_3^2 + 2\max a_4}{a_4}.$$

   It turns out that we cannot freely choose the size of the reservoir. Clearly the longer the valve takes to close the larger the reservoir has to be to prevent overflow.
2. Empty Reservoir: To verify that the reservoir will not be empty we compute the minimal water level, again using the derivative according to [10]:

   $$\frac{1}{2}\frac{2c_{12}a_2 - a_1^2}{a_2}.$$

   The resulting condition is that

   $$\frac{1}{2}\frac{a_1^2}{a_2} < c_{12}.$$

   The minimum possible value for c_{12} is restricted by the closing speed of the valve but further we know that $c_{12} < c_{34} < \max$.

6.2. Reactor Temperature Control

The goal is to keep the temperature of a reactor in a certain range using two cooling rods. This example is proposed in [1,7,18]. The cooling rods can only be used for a certain amount of time and then have to regenerate, hence we use them in alternation. We have to make sure that the reactor does not heat up to its critical temperature. The problem is usually presented as an automaton having four modes, shown in Fig. 4a. The "Shutdown" mode is special, it is not a mode, but is used to express the required safety property of the system, namely that we have to shutdown the reactor if no cooling rod is available but the temperature of the reactor is critical. In our modeling we do not use this mode but verify the property directly.

For easy readability and space reasons, we demonstrate the computation using sample numbers, the parameterized solutions are too large to be presented here. In the verification step we use the fully parameterized systems, meaning that even reactor heating, regenerating and usage times for each individual rod are symbolic.

In this example we used the method presented in Section 4.2 to construct a hybrid automaton with only a single entrance to each mode. The original automaton, having $3+1$ modes, is extended to an automaton with 4 modes as shown in Fig. 4b.

We use the following variables and constants:

- x is the timer for cooling rod one, y is the timer for cooling rod two, z the temperature of the reactor. If no cooling rod is inserted the temperature of the reactor is modeled using the differential equation $z' = (z - z_{\text{heat}})/10$. If a cooling rod is inserted having the cooling effect *cool* then the temperature behaves like $z' = (z - \text{heat} - \text{cool})/10$. We normalize the constants *heat* and *cool* resulting in the equation is $z' = 1/10 * z - z_{\text{heat}} - z_{\text{ref}_i}$. The cooling effect of the rods are z_{ref_1}, z_{ref_2}, both assumed to be 10. The heating rate of the reactor without any rod inserted is $z_{\text{heat}} = 50$.
- The critical and normal temperature t_{crit}, t_{nor}, we use $t_{\text{crit}} = 550$, $t_{\text{nor}} = 510$.
- The regeneration time of the rods are *txr* and *tyr*, assumed to be 3 and 4.

As stated, we are able to perform the entire computation just using symbols, e.g. parameterizing the cooling effect and the regenerating times of each individual rod as shown in Fig. 4. The differential equations and the known initial conditions in each mode are as follows:

Mode M1: Wait until cooling rod one is regenerated, the reactor is not cooled.

$$x' = 1, \quad y' = 1, \quad z' = \tfrac{1}{10}z - 50, \quad x(s_1) = x_1, \quad y(s_1) = 0, \quad z(s_1) = t_{\text{nor}}.$$

Mode M2: Rod one is used to cool the reactor.

$$x' = 1, \quad y' = 1, \quad z' = \tfrac{1}{10}z - 60, \quad x(s_2) = x_2, \quad y(s_2) = y_2, \quad z(s_2) = t_{\text{crit}}.$$

Mode M3: Wait until cooling rod two is regenerated, the reactor is not cooled.

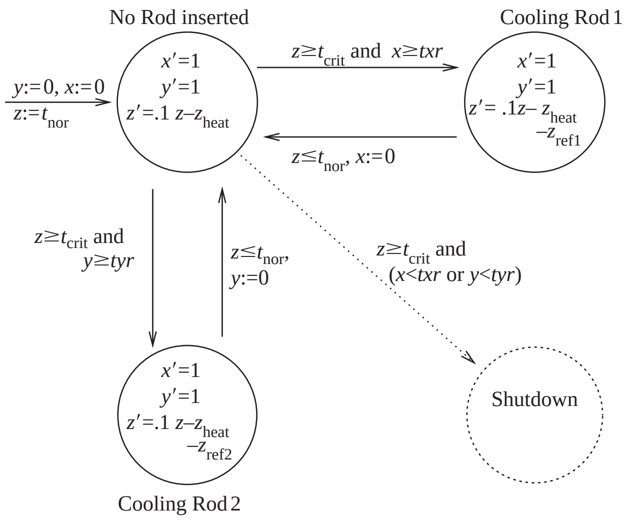
$$x' = 1, \quad y' = 1, \quad z' = \tfrac{1}{10}z - 50, \quad x(s_3) = 0, \quad y(s_3) = y_3, \quad z(s_3) = t_{\text{nor}}.$$

Mode M4: Cooling rod two is inserted into the reactor.

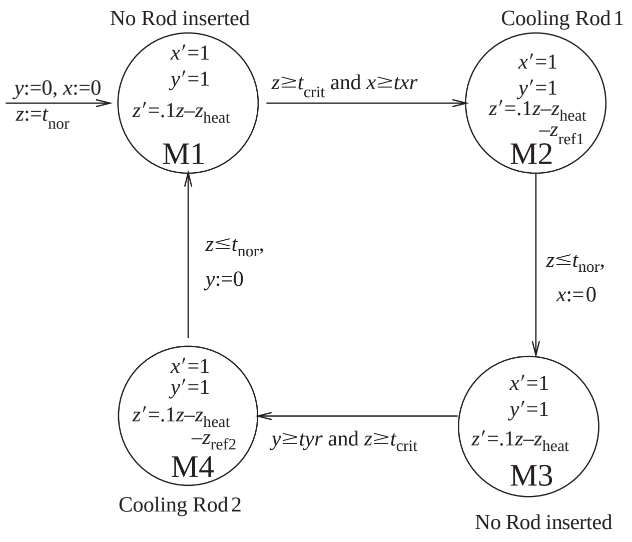
$$x' = 1, \quad y' = 1, \quad z' = \tfrac{1}{10}z - 60, \quad x(s_4) = x_4, \quad (s_4) = y_4, \quad z(s_4) = t_{\text{crit}}.$$

6.2.1. Solving the Equations

We solve the differential equations in each mode and determine how long the system remains in this mode and how the state variables change during this time.



No Rod inserted
Cooling Rod 1
Cooling Rod 2
Shutdown
$y:=0, x:=0$
$z:=t_{\text{nor}}$
$x'=1$
$y'=1$
$z'=.1\,z-z_{\text{heat}}$
$z'=.1z-z_{\text{heat}}-z_{\text{ref1}}$
$z'=.1\,z-z_{\text{heat}}-z_{\text{ref2}}$
$z\geq t_{\text{crit}}$ and $x\geq txr$
$z\leq t_{\text{nor}}, x:=0$
$z\geq t_{\text{crit}}$ and $y\geq tyr$
$z\leq t_{\text{nor}}, y:=0$
$z\geq t_{\text{crit}}$ and $(x<txr$ or $y<tyr)$



No Rod inserted
Cooling Rod 1
Cooling Rod 2
No Rod inserted
M1
M2
M3
M4
$y:=0, x:=0$
$z:=t_{\text{nor}}$
$x'=1$
$y'=1$
$z'=.1z-z_{\text{heat}}$
$z'=.1z-z_{\text{heat}}-z_{\text{ref1}}$
$z'=.1z-z_{\text{heat}}-z_{\text{ref2}}$
$z\geq t_{\text{crit}}$ and $x\geq txr$
$z\leq t_{\text{nor}}, x:=0$
$y\geq tyr$ and $z\geq t_{\text{crit}}$
$z\leq t_{\text{nor}}, y:=0$

Fig. 4. Reactor temperature control. (a) Original system and (b) Modified system.

Mode M1 Solving the first differential equation results in:

$$\mathrm{sol}_1 := y(t) = -s_1 + y_1 + t,\quad x(t) = -s_1 + x_1 + t,$$
$$z(t) = 10\frac{\mathrm{e}^{(1/10t)}}{\mathrm{e}^{(1/10s_1)}} + 500$$

We determine at which time point we have to switch to mode M2, by computing the time the reactor reaches the critical temperature, this is $z(t) = t_{\mathrm{crit}}$. Solving this equation we obtain $\mathrm{e}x_1$ the time spent in mode M1 depending on s_1, the time point in which mode M1 is entered, and the resulting value of the other state variables.

$$\mathrm{e}x_1 := 10\ln(5) + s_1,\quad v_1x := x_1 + 10\ln(5),$$
$$v_1y := y_1 + 10\ln(5),\quad v_1z := 10\frac{\mathrm{e}^{(\ln(5)+1/10s_1)}}{\mathrm{e}^{(1/10s_1)}} + 500$$

Mode M2 Again we solve the differential equations resulting in:

$$\mathrm{sol}_2 := z(t) = -50\frac{\mathrm{e}^{(1/10t)}}{\mathrm{e}^{(1/10s_2)}} + 600,$$
$$y(t) = -s_2 + y_2 + t,\quad x(t) = -s_2 + t$$

We determine when the reactor temperature is normal, $z(t) = t_{\mathrm{nor}}$. Solving this equation we obtain $\mathrm{e}x_2$ the time spent in mode M2 depending on s_2, the time point in which mode M2 is entered, and the resulting value of the other state variables.

$$\mathrm{e}x_2 := 10\ln\left(\frac{9}{5}\right) + s_2,\quad v_2x := 10\ln\left(\frac{9}{5}\right),$$
$$v_2y := y_2 + 10\ln\left(\frac{9}{5}\right),\quad v_2z := -50\frac{\mathrm{e}^{(\ln(9/5)+1/10s_2)}}{\mathrm{e}^{(1/10s_2)}} + 600$$

Mode M3 The solution of the differential equation is:

$$\mathrm{sol}_3 := z(t) = 10\frac{\mathrm{e}^{(1/10t)}}{\mathrm{e}^{(1/10s_3)}} + 500,$$
$$x(t) = -s_3 + x_3 + t,\quad y(t) = -s_3 + y_3 + t$$

Again we wait until the reactor temperature is critical, $z(t) = t_{\mathrm{crit}}$. Again we solve to determine the time and the resulting value of all state variables.

$$\mathrm{e}x_3 := 10\ln(5) + s_3,\quad v_3x := x_3 + 10\ln(5),$$
$$v_3y := y_3 + 10\ln(5),\quad v_3z := 10\frac{\mathrm{e}^{(\ln(5)+1/10s_3)}}{\mathrm{e}^{(1/10s_3)}} + 500,$$

Mode M4 Finally mode M4, the solution of the equation is:

$$\mathrm{sol}_4 := z(t) = -50\frac{\mathrm{e}^{(1/10t)}}{\mathrm{e}^{(1/10s_4)}} + 600,$$
$$x(t) = -s_4 + x_4 + t,\quad y(t) = -s_4 + t$$

Rod two is used until the reactor temperature is normal again, $z(t) = t_{\mathrm{nor}}$. And for the last time we determine the time send in the mode and the resulting change of the state variables.

$$\mathrm{e}x_4 := 10\ln\left(\frac{9}{5}\right) + s_4,\quad v_4x := x_4 + 10\ln\left(\frac{9}{5}\right),$$
$$v_4y := 10\ln\left(\frac{9}{5}\right),\quad v_4z := -50\frac{\mathrm{e}^{(\ln(9/5)+1/10s_4)}}{\mathrm{e}^{(1/10s_4)}} + 600.$$

6.2.2. Solving the Recurrence Relations

Since we already know the trace we can easily derive the recurrence relations by following the transitions of one period in the trace. We do not present these steps here. We directly state the solution of the resulting recurrence relations:

$$ft(n) = 40\ln(3)n,\quad fx(n) = 20\ln\left(\frac{9}{5}\right) + 10\ln(5),$$
$$fy(n) = 10\ln\left(\frac{9}{5}\right),\quad fz(n) = -50\frac{\mathrm{e}^{(\ln(9/5)+1/10s_4)}}{\mathrm{e}^{(1/10s_4)}} + 600$$

The solutions are simple, since we know most of the values of the state variables as we enter the modes, hence fx, fy, fz are always the same as we exit mode M1.

Using our equations we can compute the initial conditions of each mode in one period, now depending on n. This is done mechanically, using our implementation and the results are not shown here.

The period of the system is easily computed to be $\mathtt{per} = 40\ln(3) = 43.944$. Finally, we are ready to construct the solutions:

$$sfx := \mathrm{periodic}\begin{cases} -40\ln(3) + 20\ln(9/5) + 10\ln(5) + t & t < 10\ln(5) + 40\ln(3)_{,40\ln(3),t}(t) \\ -10\ln(5) - 40\ln(3) + t & \text{otherwise} \end{cases}$$

$$sfy := \mathrm{periodic}\begin{cases} -40\ln(3) + 10\ln(9/5) + t & t < 10\ln(5) + 40\ln(3)_{,40\ln(3),t}(t) \\ -20\ln(5) - 10\ln(9/5) - 40\ln(3) + t & \text{otherwise} \end{cases}$$

$$sfz := \text{periodic}\begin{cases} \frac{10}{81}e^{(1/10t)} + 500 & t < 10\ln(5) + 40\ln(3) \\ -50\frac{e^{(1/10t)}}{e^{(\ln(5)+4\ln(3))}} + 600 & t < 10\ln(9/5) + 10\ln(5) + 40\ln(3)_{,40\ln(3),t}(t) \\ 10\frac{e^{(1/10t)}}{e^{(\ln(9/5)+\ln(5)+4\ln(3))}} + 500 & t < 20\ln(5) + 10\ln(9/5) + 40\ln(3) \\ -50\frac{e^{(1/10t)}}{e^{(2\ln(5)+\ln(9/5)+4\ln(3))}} + 600 & \text{otherwise} \end{cases}$$

The plot in Fig. 5 shows the temperature $\tilde{sfz}(t)$, using $\tilde{sfz}(t) = sfz(t) - 500$ to compress the picture, and the timers $sfx(t)$ and $sfy(t)$.

6.2.3. Verification

For the verification we performed the entire computation using the parameters. All parameters are assumed to have positive values. The condition that in mode M1 rod one is regenerated and ready to be used as soon as the reactor reaches its critical temperature is

$$10\ln\left(\frac{10z_{\text{heat}} - t_{\text{crit}}}{10z_{\text{heat}} - t_{\text{nor}}}\right) + 10\ln\left(\frac{10z_{\text{heat}} + 10z_{\text{ref}_2} - t_{\text{nor}}}{10z_{\text{heat}} + 10z_{\text{ref}_2} - t_{\text{crit}}}\right) < txr.$$

The condition that in mode M3 rod two is regenerated and ready to be used as soon as the reactor reaches its critical temperature is

$$10\ln\left(\frac{10z_{\text{heat}} - t_{\text{crit}}}{10z_{\text{heat}} - t_{\text{nor}}}\right) + 10\ln\left(\frac{10z_{\text{heat}} + 10z_{\text{ref}_1} - t_{\text{nor}}}{10z_{\text{heat}} + 10z_{\text{ref}_1} - t_{\text{crit}}}\right) < tyr.$$

One should notice another implicit additional condition. The arguments of the ln have to be positive which implies that $t_{\text{crit}}/10 < z_{\text{heat}} + z_{\text{ref}_i}, i = 1, 2$, assuming that $t_{\text{nor}} < t_{\text{crit}}$. These two conditions mean, examining the differential equation for $z(t)$, that inserting the cooling rods actually cools the reactor.

6.3. The Cat and Mouse Game

The cat–mouse game is a simple example found in literature [7,15]. We present it here since it is an example of a non-periodic system. Our implementation can compute the explicit solution of this system fully automated. In contrast to other approaches which verify

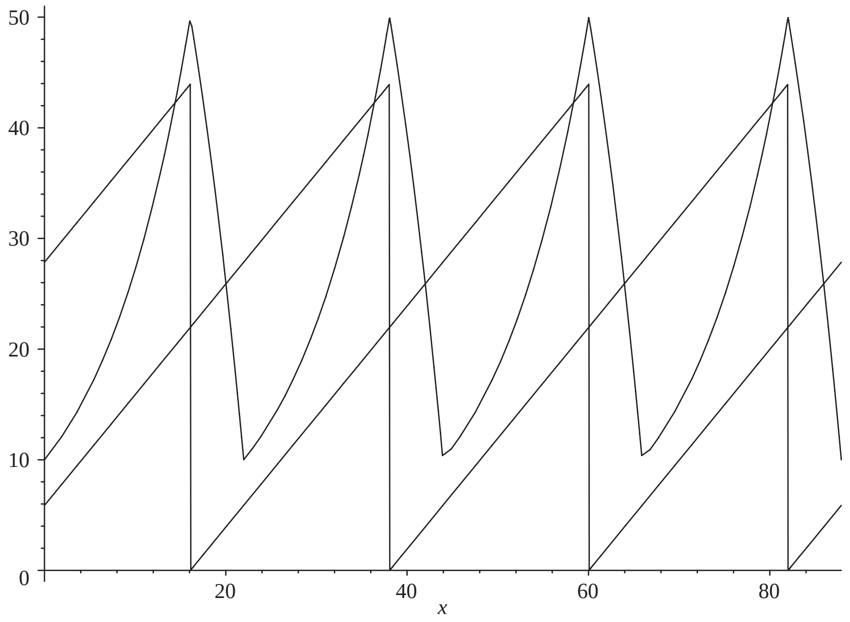


Fig. 5. Plot of solution.

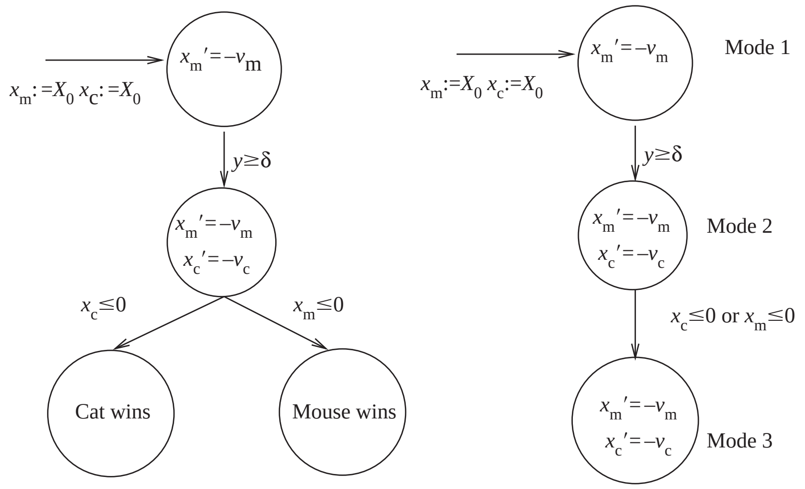


Fig. 6. Cat-Mouse system. (a) Original system (b) Modified system.

given properties we are actually able to derive the properties of the system. The game works as follows: The mouse and the cat are at position X_0, the mouse starts running with speed v_m, the cat waits for δ seconds and the starts running with speed v_c. If the mouse reaches home before the cat can catch up the mouse wins, else the cat wins. We use the following variables and symbols:

- $X_0 > 0$ initial position of cat and mouse,
- $x_m > 0$ position of mouse, v_m speed of mouse,
- $x_c > 0$ position of cat, $v_c > 0$ speed of cat, $y > 0$ timer used by cat, $\delta > 0$ time the cat waits until running.

The game is described by the hybrid automaton given in Fig. 6a, Again, for the actual verification we do not use the automaton presented in 6a, but we modify the automaton as given in Fig. 6b to obtain a deterministic 3 mode automaton and verify the property that $x_m(t) < x_c(t)$ in mode 3, meaning we verify under which condition the mouse wins.

The computations are again performed using the computer algebra system Maple.

6.3.1. *Differential Equations with Initial Conditions*

Mode 1: Mouse runs, Cat waits until timer y is δ

$$\mathrm{deq}_1 := \left\{\frac{\partial}{\partial t}x_m(t) = -v_m, \frac{\partial}{\partial t}x_c(t) = 0, \frac{\partial}{\partial t}y(t) = 1, x_m(0) = X_0, x_c(0) = X_0, y(0) = 0\right\}$$

Mode 2: Mouse and cat run

$$\mathrm{deq}_2 := \left\{\frac{\partial}{\partial t}x_m(t) = -v_m, \frac{\partial}{\partial t}x_c(t) = -v_c, x_m(st_2) = x_2, x_c(st_2) = X_0\right\}$$

Solve Mode 1 Solving differential equation 1 we get

$$\mathrm{sol}_1 := \{x_m(t) = X_0 - v_m t, x_c(t) = X_0, y(t) = t\}.$$

Next we determine the time spend in mode 1 by solving the equation $y(t) = \delta$, which happens after a period of δ. A time point δ the value of x_m is

$$x_m(\delta) = X_0 - v_m\delta.$$

Solve Mode 2 Solving differential equation 2, we obtain

$$\mathrm{sol}_2 := \{x_m(t) = v_m st_2 + x_2 - v_m t, \quad x_c(t) = v_c st_2 + X_0 - v_c t\}.$$

Mode 3 The solution in mode 3 is the same as the one in mode 2. We only have to decide at which time points we enter mode 3. This is accomplished using parameters st_2 and x_2:

$$x_m := X_0 - v_m t.$$
$$x_c := v_c\delta + X_0 - v_c t.$$

We now can decide in which sate the system will end up.

The time point when Mouse is home obtained by solving $x_m = 0$

$$c_m := \frac{X_0}{v_m}.$$

Next we compute the time point at which the car arrives at home, obtained by solving $x_c = 0$

$$c_c := \delta + \frac{X_0}{v_c}.$$

We reached mode 3 and would like to determine if the mouse reached home first, by deciding which of the quantities p_m or p_c is first to be zero. The can easily be computed by solving $p_m = 0$ and $p_c = 0$ resulting in:

$$\frac{X_0}{v_m} < \delta + \frac{X_0}{v_c}.$$

The solution (computed using our Maple implementation) is

$$x_m(t) = \begin{cases} X_0 - v_m t & t < \delta, \\ X_0 - v_m t & \text{otherwise} \end{cases}$$

$$x_c(t) = \begin{cases} X_0 & t < \delta, \\ v_c\delta + X_0 - v_c t & \text{otherwise} \end{cases}$$

$$y(t) = \begin{cases} t & t < \delta. \\ t - \delta & \text{otherwise} \end{cases}$$

Note, the solution has only two conditionals, the solutions in mode 2 and 3 are equal.

7. Conclusions

The presented approach to the verification of hybrid automata differs from other approaches in that it considers hybrid automata as a generalization of differential equations, states a uniqueness theorem, defines solutions and gives an algorithm to compute explicit symbolic solution of an hybrid automaton. These explicit solutions allow then to verify dynamic and static properties of hybrid systems such as computing the maximum or minimum value of certain state variables. We were able, using the presented methods, to automatically derive design constraints of systems which other methods found in literature fail to detect.

Acknowledgements

The author would like to thank the anonymous referees for their constructive comments.

References

1. Alur R, Coucoubetis C, Halbwachs N, Henzinger TA, Ho PH, Nicolin X, Olivero A, Sifakis J, Yovine S. "The Algorithmic Analysis of Hybrid Systems", Theor Comput Sci 1995; 138: 3–34
2. Alur R, Henzinger TA, Ho PH. Automatic symbolic verification of embedded systems. In: Proceedings of the 14th Annual IEEE real-time systems symposium (RTSS) 1993, pp 2–11
3. Bronstein M. The transcendental risch differential equation. J Symbol Computation, 1990; 9: 49–60
4. Henzinger TA. The theory of hybrid automata. In: 11th Annual IEEE symposium on logic in computer science (LICS) 96, 278–292
5. Filipoph P. Differential equations with discontinuous right-hand side. Mathematics and its Applications. Kluwer, ISBN 90-277-2699-X
6. Branicky MS. A unified framework for hybrid control: model and optimal control theory. IEEE Trans Auto Control 1998; 43(1)
7. Nicollin X, Olivero A, Sifakis J, Yovine S. An approach to the description and analysis of hybrid systems, Lecture Notes in Computer Science 1993; 736: 149–178
8. Mohrenschildt Mv. A normal form for rings of piecewise functions. J Symbolic Computation 1998; 26: 607–619
9. Mohrenschildt Mv. Using piecewise to solve classes of control theory problems. MapleTech 1997; 4(3): 33–37
10. Mohrenschildt Mv. Solving discontinuous differential equations. In: (eds). The Combinatory Program, Engeler E, Birkhauser, pp 98–116
11. Mohrenschildt Mv. Solving discontinuous ordinary differential equations with a computer algebra system, submitted to J Symbolic Computation
12. Hybrid Systems, Lecture Notes in Computer Science (736)
13. Clarke FH. et al. Non-smooth analysis and control theory. Springer, ISBN 0-387-98336-8
14. Kesten Y, Pnueli A, Sifakis J, Yovine S. Decidable integration graphs: a class of decidable hybrid systems. Information and Computation, 1999; 150: 209–243
15. Kurki-Suonio R. Hybrid models with fairness and distributed clocks, Lecture Notes in Computer Science 1993, 736, pp 103–120
16. Nicollin X, Sifakis J, Yovine S. Compiling real-time specifications into extended automata. IEEE TSE Real-Time Systems 1992; 18(9): 794–804
17. Singer MF. Formal solutions of differential equations. J Symbolic Computation 1990; 10: 59–94
18. Kapur D, Shyamasundar RK. Synthesizing controllers for hybrid systems. Hybrid and Real Time Systems, HART'97, Lecture Notes in Computer Science 1201, Springer-Verlag, 1997; 361–375
19. Pappas G, Lafferriere G, Yovine S. A new class of decidable hybrid systems, Hybrid Systems: Computation and Control, Lecture Notes in Computer Science 1569, HS'99, Nijmegen, The Netherlands, March 1999, 137–152
20. Manna Z, Sipma H. Deductive verification of hybrid systems using STeP, HSCC 98, Lecture Notes in Computer Science 1386, 1998; 305–318
21. Alur R, Henzinger A, Lafferriere G, Pappas GJ. Discrete Abstraction of Hybrid Systems. Proceedings of the IEEE, 2000; 88(2): 971–984