



Towards the Safety Verification of Real-Time Systems with the Coq Proof Assistant

Olga Tveretina

Institute for Computing and Information Sciences,
Radboud University Nijmegen,
The Netherlands
olga@cs.ru.nl

Abstract. Hybrid systems are systems involving the interaction of discrete and continuous dynamics. Hybrid systems have been used as a mathematical model for many safety critical applications. One of the most important analysis problems of hybrid systems is the reachability problem. Approaches based on predicate abstraction are widely used for the reachability analysis. They are not efficient enough because of introducing additional transitivity along the series of abstract states. In this paper we give an approach to solve this problem for some classes of hybrid systems. A verification example formalized within the Coq proof assistant is provided.

Keywords: Hybrid systems, reachability, verification, discrete abstraction

1 Introduction

Real-time embedded systems have become very important in our everyday life. Programs such as device drivers and embedded controllers must run on real-time constraints. Demand placed on the embedded systems functionality, complexity and critical nature are increasing [5].

Hybrid systems are systems when there is a significant interaction between the continuous and discrete parts and high performance specifications are to be met by the system. For example, hybrid systems arise from the interaction of discrete planning algorithms and continuous processes. The study of hybrid systems is essential in designing intelligent control systems. Examples of such systems include intelligent highway systems, air traffic management systems, computer and communication networks, smart houses and others. Another challenging application is a construction of mathematical models of biological processes.

Many of the above mentioned applications are *safety critical* and require the guarantee of safe operation. The problem of safety verification seeks for an answer to the question: is there a potentially unsafe state reachable from an initial state? Therefore, formal verifying safety properties of a hybrid system consists of building a set of reachable states and checking whether this set intersects with a set of unsafe sets. That is why, one of the most central problems in

the analysis of hybrid automata is the *reachability* problem. For systems with continuous dynamics, it is very difficult to compute the set of all states reachable from an initial set.

Abstraction is one of the complexity reduction techniques [4]. It reduces the state space of a system by mapping it to an abstract set of states that preserve the actual behavior of the system. Predicate abstraction is the most widely used form of the abstraction. Given a concrete infinite state system and a set of abstraction predicates, a finite state abstraction is produced. It is an abstraction in terms of that for every execution in the concrete system there is a corresponding execution in the abstract system. A predicate abstraction approach for the verification of hybrid systems is represented in [9]. However, the computational cost of predicate abstraction can be too high, and for large systems practically infeasible. That is why we start from a method that decomposes the state space of a system according a rectangular grid [11]. We show applicability of our method with the verification example that we have formalized within the `Coq` proof assistant. It models the gate controller of a railroad crossing.

Related work. The problem of computing reachable states of hybrid systems received a lot of attention during past several years and different reachability tools have been developed. There are two main approaches for reachability analysis: a) the methods "overapproximating" the reachable set; b) the methods based on computing "convergent approximations" to reachable sets [12]. "Overapproximating" methods compute an overapproximation of a set of reachable states. The tools such as *d/dt* [2] and *Checkmate* [10] represent sets as convex polyhedra and propagate these polyhedra under continuous dynamics. A second group of methods is based on computing solutions to static Hamilton-Jacobi equations and on techniques from viability theory and set valued analysis [12]. Overapproximating approaches are mostly hard for systems with non-linear dynamics. The methods in the second group are exponential in n and thus not practical for problems with large dimensions.

2 Problem description

Basically, a hybrid system is a graph, whose vertices represent continuous changes and whose edges represent discrete transitions. A system trajectory is a sequence of continuous flows and discrete jumps. A hybrid automaton combines discrete-state and continuous-state dynamics to model systems which evolve both continuously and according discrete jumps. Therefore, we use a hybrid automaton as a mathematical formalism for describing hybrid systems.

Now we formally define the notion of hybrid automaton. In the following $P(Q)$ denotes the power set (the set of all subsets) of Q , and \mathbb{R} denotes the set of real numbers.

Definition 1. *A Hybrid automaton is a tuple $\mathcal{H} = (\mathcal{L}, n, \mathcal{S}_0, \mathcal{I}, F, \mathcal{G}, \mathcal{R})$ with the following components:*

- \mathcal{L} is a finite set of discrete locations; $n \geq 0$ is called the dimension of \mathcal{H} . The state space of \mathcal{H} is $\mathcal{S} = \mathcal{L} \times \mathbb{R}^n$. Each state thus has the form (l, x) , where $l \in \mathcal{L}$ is a discrete part, and $x \in \mathbb{R}^n$ is continuous;
- $\mathcal{S}_0 \subseteq \mathcal{S}$ is a set of initial states;
- $\mathcal{I} : \mathcal{L} \rightarrow P(\mathbb{R}^n)$ assigns to each location l an invariant set $\mathcal{I}(l) \subseteq \mathbb{R}^n$, which constrains the value of the continuous part while the discrete part is l , i.e. continuous evaluation can go on as long as x remains in $\mathcal{I}(l)$.
- $F : \mathcal{S} \rightarrow \mathbb{R}^n$ assigns to each discrete state a continuous vector field, i.e. starting from an initial state $(l_0, x_0) \in \mathcal{S}_0$ the continuous state x flows according to the differential equation $\dot{x} = F(l_0, x)$ and $x(0) = x_0$. We denote by \dot{x} the first derivative of x with respect to time, i.e. $\dot{x} = dx/dt$.
- $\mathcal{G} : \mathcal{L} \times \mathcal{L} \rightarrow P(\mathbb{R}^n)$ describes a guard condition, i.e. if a system remains in a discrete location l_1 and a continuous state x reaches the guard $\mathcal{G}(l_1, l_2)$ then the discrete state may change its value to l_2 .
- $\mathcal{R} : \mathcal{L} \times \mathcal{L} \times \mathbb{R}^n \rightarrow P(\mathbb{R}^n)$ is a reset function. It means that when a discrete state changes its value from l_1 to l_2 , a continuous state gets reset to some value in $\mathcal{R}(l_1, l_2, x) \subseteq \mathbb{R}^n$.

For simplicity, we assume that a number of discrete locations is finite and for all $l \in \mathcal{L}$, the vector field $F(l, \cdot)$ is Lipschitz continuous. It guarantees that the solutions of the differential equation $\dot{x} = F(l, x)$ are well-defined.

Example 1. (Temperature control) The temperature T of a room is controlled by a thermostat, which continuously checks the temperature and turns the heater on and off if some threshold values T_{min} and T_{max} are reached. When the heater is on, the temperature increases according to the flow $\dot{T} = kt$, when the heater is off, the temperature decreases according to the flow $\dot{T} = -kt$. The resulting hybrid system is shown in Figure 1. A variable c is used to ensure an upper bound spent in each node.

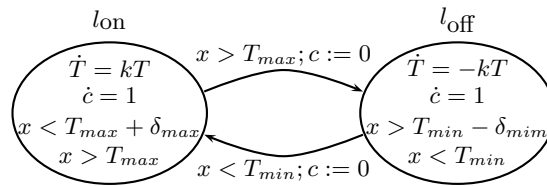


Fig. 1. Graphical representation of the thermostat hybrid automaton.

Let $\mathbb{R}_{\geq 0}$ be the set of non-negative real numbers. We denote by Φ the set of functions $f : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}^n$ and define the flow of a system in a discrete location l as $\phi_l(x, f, t)$ with an initial condition $\phi_l(x_0, f, 0) = x_0$. A system state can change in two ways: either by time evaluation or by discrete transitions. Hence, there are two kinds of transitions: a continuous transition, denoted as \rightarrow_c , describes

evolution of a system in a given location, and a transition, denoted as \rightarrow_d , describes moving from one location to another location and, possibly, changing the continuous variables according to the function \mathcal{R} . Then the semantics of a hybrid automaton is given by the corresponding transition system [1].

Definition 2. *Given a hybrid automaton $\mathcal{H} = (\mathcal{L}, n, \mathcal{S}_0, \mathcal{I}, F, \mathcal{G}, \mathcal{R})$, the transition system $\mathcal{T} = \{\mathcal{S}, \rightarrow, \mathcal{S}_0\}$ of \mathcal{H} consists of*

- set of states \mathcal{S} ;
- the set of initial states \mathcal{S}_0 ;
- the transition relation $\rightarrow \subseteq \mathcal{S} \times \mathcal{S}$ between states, and it is defined as an union of two relations \rightarrow_c and \rightarrow_d , where $\rightarrow_c \subseteq \mathcal{S} \times \mathcal{S}$ the relation describing a transition due to continuous flow and $\rightarrow_d \subseteq \mathcal{S} \times \mathcal{S}$ the relation describing a transition due to a discrete jump, i.e.

$$(l, x) \rightarrow_c(l, y) \Leftrightarrow \exists t \geq 0, f \in \Phi : \phi_l(x, f, t) = y \wedge \forall t' \in [0, t], \phi_l(x, f, t') \in \mathcal{I}(l)$$

$$(l, x) \rightarrow_d(l', y) \Leftrightarrow x \in \mathcal{G}(l, l') \wedge y = \mathcal{R}(l, l', x)$$

When it is necessary, we will use $(l, x) \rightarrow_c^t(l, y)$ instead of $(l, x) \rightarrow_c(l, y)$.

Trajectories of a hybrid automaton are sequences of continuous flow and discrete steps and start from initial states.

Definition 3. (*Trajectory*) *Given a hybrid automaton $\mathcal{H} = (\mathcal{L}, n, \mathcal{S}_0, \mathcal{I}, F, \mathcal{G}, \mathcal{R})$, we say that $\tau_n : s_0 \rightarrow \dots \rightarrow s_n$ is a trajectory of \mathcal{H} if $s_0 \in \mathcal{S}_0$, and $s_i \in \mathcal{S}, 1 \leq i \leq n$, where n is the length of the trajectory.*

Given a hybrid automaton \mathcal{H} and a set of unsafe states \mathcal{U} , the safety verification problem concerns with proving that all trajectories of \mathcal{H} cannot enter \mathcal{U} . Formally it can be defined as follows.

Definition 4. (*Safety*) *We say that a hybrid automaton is safe if for all $n \geq 0$ there is no trajectory $s_0 \rightarrow \dots \rightarrow s_n$ such that $s_i \cap \mathcal{U} \neq \emptyset$ for some $0 \leq i \leq n$.*

Our goal is to design an algorithm that, given a hybrid automaton and a set of unsafe states, decides whether the corresponding system is safe. However, this problem is undecidable in general [6]. That is why we are interested in algorithms that terminate efficiently for problems of practical importance.

3 Reachability analysis

In this section we describe an algorithm for verifying safety of hybrid systems. Basically, it is a modification of traditional methods based on discrete abstraction. The fact that the approaches based on a discrete abstraction of a state space produce too many spurious transitions, motivated us to find another approach.

3.1 Abstract transition system

Our method is based on the approach that decomposes the continuous state space according to a n -dimensional rectangular grid as, for example, in Figure 2. Such abstractions are mostly performed in a manual manner. In general, a state space can be represented by polyhedra [1]. This is a more flexible approach but it requires an algorithm for dealing with these polyhedra. A rectangular grid is less flexible but it is simpler to implement the corresponding operations.

We assume that we have an algorithm that can produce a decomposition of a state space, as for example in [8]. We denote by χ an abstract state and by S^a a set of all abstract states.

In order to define a discrete abstraction of a given system, we have to describe the transitions between the abstract states, the set of initial abstract states S_0^a , and the set of abstract unsafe states \mathcal{U}^a . Given a hybrid automaton \mathcal{H} and a set of abstract states, the set of initial abstract states can be computed.

Definition 5. A hybrid automaton $\mathcal{H} = (\mathcal{L}, n, \mathcal{S}_0, \mathcal{I}, F, \mathcal{G}, \mathcal{R})$ and an abstract state space S^a generate the abstract transition system $T^a = \{S^a, \rightsquigarrow, S_0^a\}$, where

- the set of initial abstract states S_0^a : an abstract state $(l, \chi) \in S_0^a$ if there is $(l, x) \in \mathcal{S}_0$ such that $x \in \chi$;
- the abstract transition relation $\rightsquigarrow \subseteq S^a \times S^a$ is a union of two relations: $\rightsquigarrow_c \subseteq S^a \times S^a$ is a relation describing a transition due to continuous flow and $\rightsquigarrow_d \subseteq S^a \times S^a$ is a relation describing a transition due to a discrete jump, i.e.

$$(l, \chi) \rightsquigarrow_c (l', \chi') \Leftrightarrow \exists t \geq 0, x \in \chi, x' \in \chi', f \in \Phi, l' \in \mathcal{L} : \phi_l(x, f, t) = x' \wedge x' \in \mathcal{G}(l, l') \wedge \forall t' \in [0, t], \phi_l(x, f, t') \in \mathcal{I}(l)$$

$$(l, \chi) \rightsquigarrow_d (l', \chi') \Leftrightarrow \exists x \in \chi, x' \in \chi', : x \in \mathcal{G}(l, l') \wedge x' = \mathcal{R}(l, l', x).$$

When it is necessary, we will use $(l, \chi) \rightsquigarrow_c^t (l, \chi')$ instead of $(l, \chi) \rightsquigarrow_c (l, \chi')$.

Due to finiteness of the abstraction, we can check it's safety by computing all possible transitions.

The abstract transition system described here differs from the abstraction transition system as given by [1] in terms of the formulation of the relation \rightsquigarrow_c defining a continuous step.

Example 2. The grid for the thermostat example is depicted in Figure 2. Each rectangular box represents an abstract state. We assume that $T_{min}^* < T_{min} < T_{max} < T_{max}^*$, where $T_{min}^* = T_{min} - \delta_{min}$ and $T_{max}^* = T_{max} + \delta_{max}$. While staying in a discrete location l_{on} , an abstract continuous step from a state (l_{on}, χ_0) to a state (l_{on}, χ_1) is possible.

Definition 6. An abstract trajectory is defined as a sequence

$$s_{(0,0)}^a \rightsquigarrow_c s_{(0,1)}^a \rightsquigarrow_d s_{(1,0)}^a \rightsquigarrow_c \dots \rightsquigarrow_d s_{(i,0)}^a \rightsquigarrow_c s_{(i,1)}^a \rightsquigarrow_d \dots,$$

where $s_{(0,0)}^a \in S_0^a$.

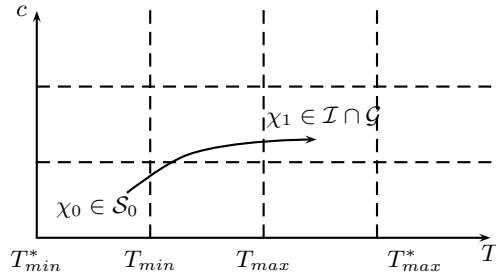


Fig. 2. An abstract continuous step from a state (l_{on}, χ_0) to a state (l_{on}, χ_1) .

An abstract trajectory is closed if there are i and j such that $i < j$ and

$$s_{(0,0)}^a \rightsquigarrow_c s_{(0,1)}^a \rightsquigarrow_d s_{(1,0)}^a \rightsquigarrow_c \dots \rightsquigarrow_d s_{(i,0)}^a \rightsquigarrow_c \dots \rightsquigarrow_d s_{(j,0)}^a,$$

where $s_{(i,0)}^a = s_{(j,0)}^a$. We denote by $Tr(s^a)$ a closed abstract trajectory starting from an abstract state s^a .

Note that in general an abstract trajectory starting from some initial abstract state is not unique. Existence of a closed abstract trajectory follows from finiteness of the continuous state space partition.

Standard approaches based on predicate abstraction are less efficient than they could be, because of introducing additional transitivity along the series of abstract states, i.e. they produce too many spurious abstract transitions. It means that if for abstract states χ_1, χ_2 and χ_3 holds $\chi_1 \rightsquigarrow_c \chi_2$ and $\chi_2 \rightsquigarrow_c \chi_3$ than χ_3 is proclaimed to be reachable from χ_1 , although it is not always the case. In contrast to the standard methods, our framework allows computing only transitive transitions.

3.2 Algorithm

Given a hybrid system, the reachability problem is to check whether all trajectories starting from an initial state avoid unsafe regions. If the reachability is verified for the abstraction then it holds in the concrete system. In general, this problem is undecidable. So we have aimed to have an algorithm that given a hybrid system and an abstraction of the state space the following holds: a) it terminates; b) if it terminates with an answer 'safe', then the original system is safe.

We define an "auxiliary" abstract continuous step as follows.

$$(l, \chi) \rightsquigarrow^{aux} (l, \chi') \Leftrightarrow \exists x \in \chi, x' \in \chi' : (l, x) \rightarrow_c (l, x')$$

The above definition is similar to the definition of an abstract continuous step of [1] but it serves a different purpose, i.e. defining the "auxiliary" set of successors $Succ_c^{aux}$ we avoid computing spurious abstract transitions.

$$Succ_c^{aux}((l, \chi), (l, \chi')) = \{(l, \chi'') \in \mathcal{S}^a \mid \exists t \geq 0 : (l, \chi) \rightsquigarrow_c^t (l, \chi') \wedge \forall t' \in]0, t[, (l, \chi) \rightsquigarrow^{aux} (l, \chi'')\}.$$

Given an abstract trajectory

$$\tau^a : s_{(0,0)}^a \rightsquigarrow_c s_{(0,1)}^a \rightsquigarrow_d s_{(1,0)}^a \rightsquigarrow_c s_{(1,1)}^a \rightsquigarrow_d \dots \rightsquigarrow_d s_{(n,0)}^a$$

the set $\text{Reach}(\tau)$ is define as follows.

$$\text{Reach}(\tau) = \bigcup_{i=0}^n s_{(i,0)}^a \cup \bigcup_{i=0}^{n-1} \text{Succ}_c^{aux}(s_{(i,0)}^a, s_{(i,1)}^a)$$

Lemma 1. *Given a hybrid automaton $\mathcal{H} = (\mathcal{L}, n, \mathcal{S}_0, \mathcal{I}, F, \mathcal{G}, \mathcal{R})$, for each trajectory τ there is a corresponding abstract trajectory τ^a such that for each $s \in \tau$ there is $s^a \in \text{Reach}(\tau^a)$ such that $s \in s^a$.*

Proof. We give a sketch of a proof. Suppose

$$\tau : s_{(0,0)} \rightarrow_c \dots \rightarrow_c s_{(0,i_0)} \rightarrow_d s_{(1,0)} \rightarrow_c \dots \rightarrow_d s_{(n,0)} \rightarrow_c \dots \rightarrow_c s_{(n,i_n)}$$

Let us construct the following abstract trajectory

$$\tau^a : s_{(0,0)}^a \rightsquigarrow_c s_{(0,1)}^a \rightsquigarrow_d s_{(1,0)}^a \rightsquigarrow_c s_{(1,1)}^a \rightsquigarrow_d \dots \rightarrow_d s_{(n,0)}^a,$$

where $s_{(k,0)} \in s_{(k,0)}^a$, $0 \leq k \leq n$ and $s_{(k,i_k)} \in s_{(k,i_k)}^a$, $0 \leq k \leq n - 1$.

By definition, $s_{(k,j)} \in \text{Succ}_c^{aux}(s_{(k,0)}^a, s_{(k,1)}^a)$, $0 \leq k \leq n - 1$, $1 \leq j \leq i_k$.

We conclude that for each $s \in \tau$ there is $s^a \in \text{Reach}(\tau^a)$ such that $s \in s^a$.

Given an abstraction of a state space and a set of unsafe states, the set of unsafe abstract states \mathcal{U}^a can be computed.

The methods based on predicate abstraction are often less efficient than they could be because of the accuracy to which the abstract reachable states are computed. The extra accuracy requires much more calls of validity checker than it is necessary. Often an *over-approximation* of abstract reachable states is sufficient to prove the verification condition. In cases when the continuous flow is represented by a monotone function in a discrete location l , we can compute $\text{Succ}^{aux}((l, \chi), (l, \chi'))$ as a convex hull of χ and χ' .

We define the set $\text{Succ}(Q)$ of successors of a set $Q \in \mathcal{S}^a$ and an 'auxiliary' set of reachable states $\text{Reach}^{aux}(Q)$ as follows.

$$\text{Succ}(Q) = \{s \in \mathcal{S}^a \mid \exists q \in Q, s' \in \mathcal{S}^a : q \rightsquigarrow_c s' \wedge s' \rightsquigarrow_d s\}$$

$$\text{Reach}^{aux}(Q) = \{s \in \mathcal{S}^a \mid \exists q \in Q, p \in \mathcal{S}^a : (q \rightsquigarrow_c p) \wedge (s \in \text{Succ}^{aux}(q, p) \cup p)\}.$$

Theorem 1. *Given a hybrid automaton $\mathcal{H} = (\mathcal{L}, n, \mathcal{S}_0, \mathcal{I}, F, \mathcal{G}, \mathcal{R})$, a finite set of abstract states \mathcal{S}^a , and a set of abstract unsafe states \mathcal{U}^a , the safety of the abstract system implies the safety of \mathcal{H} .*

Now we define hybrid systems for which this approach can be applied efficiently. We need to define some restrictions on functions describing continuous behavior.

Algorithm 1 REACHABILITY ALGORITHM

```

1:  $i := 0$ ;
2:  $\text{Reach}_i := \mathcal{S}_0$ ;
3:  $\text{Succ}_i := \mathcal{S}_0$ ;
4:  $\text{Succ}_{i+1} := \text{Succ}(\mathcal{S}_0)$ ;
5:  $\text{Reach}_{i+1} := \mathcal{S}_0 \cup \text{Succ}_i \cup \text{Reach}^{aux}(\mathcal{S}_0)$ ;
6: while ( $\text{Reach}_i \cap \mathcal{U}^a = \emptyset \wedge \text{Reach}_{i+1} \not\subseteq \text{Reach}_i$ ) do
7:   if  $\text{Reach}_{i+1} \cap \mathcal{U}^a \neq \emptyset$  return "unsafe";
8:   if  $\text{Reach}_{i+1} \subseteq \text{Reach}_i$  return "safe";
9:    $i := i + 1$ ;
10:   $\text{Succ}_i := \text{Succ}(\text{Succ}_{i-1})$ ;
11:   $\text{Reach}_{i+1} := \text{Reach}_i \cup \text{Succ}_i \cup \text{Reach}^{aux}(\text{Succ}_{i-1})$ ;
12: end while

```

Definition 7. (*Admissible function*) We say that a function $f : \mathbb{R}_0 \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ for some $m, n \in \mathbb{N}$ is admissible if for all $x_1, x_2, x_3 \in \mathbb{R}^m$ and $t \in \mathbb{R}_0$ there are $\alpha_1, \alpha_2 \in [0, 1]$ such that $x_3 = \alpha_1 x_1 + (1 - \alpha_1)x_2$ and $f(t, x_3) = \alpha_2 f(t, x_1) + (1 - \alpha_2)f(t, x_2)$.

The above definition means that while time elapses, each segment of a straight line is mapped to a segment of a straight line. In the following we consider only admissible functions.

4 Modeling of hybrid systems with Coq

We give here only a very brief description of the Coq proof assistant, and refer for more details to [3]. Coq is a proof assistant for higher order typed λ -calculus, i.e. the proof in Coq is equivalent to the construction of a λ -term.

Coq is an interactive proof assistant, composed of a specification language called Galina and a proof engine, and it allows formal defining mathematical objects and helps the user with proving the properties of these objects. Basically, the use of Coq follows three steps: a) define the objects and axioms used, b) state a theorem, c) provide proof steps, until the proof is complete.

In Coq, data types are presented as inductive types. For example, natural numbers are defined as follows.

```
Inductive nat: Set := | 0 : nat | S : nat -> nat.
```

This definition introduces a type *nat* which is itself of type *Set*.

Formalization in Coq. We show here how an abstraction of a hybrid automaton can be formalized in Coq. A partition P of a continuous state space is defined as a list of lists of real numbers, an abstract continuous state is defined as a list of naturals and a set of all abstract states as a list of lists of naturals.

```
Parameter P: list(list R).
```

```
Definition AbsState:= list nat.
```

Definition SetAbsStates:= list(list nat).
 Definition Time := R.
 Definition DiscState:= Set.
 Definition ContState:= list R.
 Parameter InitialStates: DiscState → SetAbsStates.
 Parameter Invariant: DiscState → list(list nat).
 Parameter Guard: DiscState → DiscreteState → list(list nat).
 Parameter Flow: Time → State → State.
 Parameter Jump: DiscState → DiscState → SetAbsStates.
 Parameter Reset: DiscState → DiscState → SetAbsStates
 → SetAbsStates.

Verification example: the gate controller of a railroad crossing. As an example we consider the gate controller of a railroad crossing from [7]. The model consists of two subsystems: a train and the gate controller. The train is required to send a signal *app* at least two minutes before it enters the crossing. The train sends a signal *out* when it leaves the crossing and it must happen no later than 5 minutes after the *app* signal (this is expressed by the guard $2 < x \leq 5$). The gate must be closed in at least 1 minute after the *app* signal is received and not later than in 2 minutes (the guard is $1 \leq y < 2$). The gate responds by opening within 1 minute after receiving the *out* signal. We have defined a product of timed automata that combines the train process and the controller process, and the resulting system is depicted in Figure 3. It has the following discrete locations: Loc1: a train is far from the gate and the gate is open; Loc2: the train is approaching the crossing and the gate is open; Loc3: the train is approaching the crossing and the gate is closed; Loc4: the train has left the crossing and the gate is closed. We want to verify that the gate is never closed more than 5 minutes.

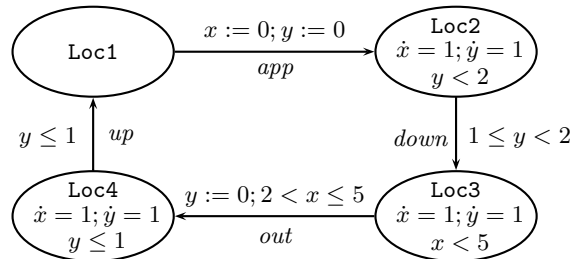


Fig. 3. A model of a train crossing problem.

We define the set of discrete states as:

Inductive DS: Set:= | Loc1:DS | Loc2:DS | Loc3:DS | Loc4:DS, and the partition of the continuous state space as:

Definition P := (0::1::2::3::4::5::6)::(0::1::2::3::4::5::6).

We do not have enough of space to present all formalizations in Coq. We just conclude that we have been able to verify with our approach that the gate is never closed more than 5 minutes.

5 Conclusion

The results presented in the paper are twofold. On one hand, we have given a framework for reachability analysis which allows to avoid spurious transitions for some classes of hybrid systems. On the other hand, the results presented in the paper were ‘mechanically’ checked in `Coq` by formalizing a train crossing example. As for future work, we intend to investigate the structure of special classes, for example linear hybrid systems, for which our method is efficient, to generalize this approach within `Coq` and to apply it to larger case studies.

6 Acknowledgments

We are grateful for the support by Milad Niqui in the field of `Coq`, and for the support by Herman Geuvers and Dan Synek for the examples of hybrid systems formalizations in `Coq`.

References

1. Rajeev Alur, Thao Dang, and Franjo Ivancic. Reachability analysis of hybrid systems via predicate abstraction. *ACM transactions on embedded computing systems (TECS)*, 2004.
2. E. Asarin, T. Dang, and O. Maler. The `d/dt` tool for verification of hybrid systems. In *Computer Aided Verification: 14th International Conference*, pages 365–370. LNCS, 2002.
3. Yves Bertot and Pierre Casteran. *Interactive Theorem proving and Program Development*. Springer, 1998.
4. E. M. Clarke and O. Grumberg. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 1992.
5. A. R. Girard, A. S. Howell, and J. K. Hedrick. Model-driven hybrid and embedded software for automotive applications. In *Proceedings of the 2nd RTAS Workshop on Model-Driven Embedded Systems*, 2004.
6. T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What’s decidable about hybrid automata? *Journal of Computer and System Sciences*, 57(1):94 – 124, August 1998.
7. T. A. Henzinger, Xavier Nicollin, Joseph Sifakis, and Sergio Yovine. Symbolic model checking for real-time systems. In *7th. Symposium of Logics in Computer Science*, 1992.
8. T. M. A. Colon and E. Uribe. Generating finite-state abstractions of reactive systems using decision procedures. In *Computer Aided Verification*, 1998.
9. A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. In *Proceedings of the sixth International Conference on Computer-Aided Verification (CAV)*, 1994.
10. B. I. Silva and B. H. Krogh. Formal verification of hybrid systems using checkmate: a case study. In *American Control Conference*, pages 1679 – 1683, 2000.
11. O. Stursberg, S. Kowalewski, I. Hoffmann, and J. Preussig. Comparing timed and hybrid automata as approximations of continuous systems. In *Hybrid Systems*, pages 361 – 377, 1997.
12. C. J. Tomplin, I. Mitchell, A. B. Bayen, and M. Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7), 2003.