How to Capture Hybrid Systems Evolution Into Slices of Parallel Hyperplanes

Tomas Dzetkulic * **Stefan Ratschan** **

* *Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodarenskou vezi 2, 182 07 Prague 8, Czech Republic (e-mail: dzetkulic@cs.cas.cz).*
** *Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodarenskou vezi 2, 182 07 Prague 8, Czech Republic (e-mail: ratschan@cs.cas.cz).*

Abstract: In this paper we make a step towards an algorithm for the verification of hybrid systems that, on the one hand allows very general inputs (e.g., with non-linear ordinary differential equations), but on the other hand exploits the structure of those parts of the input that represent special cases (e.g., clocks). We show how to compute slices of parallel hyperplanes separating reachable from unreachable parts of the state space for a given abstraction of the input system, and demonstrate the usefulness of such slices within an abstraction refinement algorithm based on hyper-rectangles.

Keywords: verification, hybrid systems, timed automata

1. INTRODUCTION

Current algorithms for the verification of hybrid systems range from one extreme of techniques that are quite efficient but allow as input only a restricted problem class (e.g., timed automata (Bengtsson and Yi, 2004)), to the other extreme of algorithms that do allow as input a very general problem class, but can hardly verify any examples of the size occurring in practical applications (Ratschan and She, 2007).

The key to arrive at algorithms that combine the advantages of both approaches, lies in the exploitation of the structure of parts of the input that represent special cases.

In this paper we use the observation that differential equalities of the form $\dot{x} = c$ (i.e., clocks of arbitrary speed) and affine switching conditions often result in reach sets that are to a certain extent bounded by hyperplanes. Hence we use pairs of affine inequalities of the form $b_{lo} \leq \boldsymbol{a}x \leq b_{hi}$ (which we call *slices*) to separate the reachable from unreachable parts of the state space. However, in order to allow general applicability, we design an algorithm for generating such slices for hybrid systems that contain non-linear ordinary differential equations, non-linear jumps, etc.

The algorithm computes such slices for a given abstraction of a hybrid system. It sets up a constraint that formalizes reachability of states within an abstract state and over-approximates the solution set of this constraint in the form of a slice.

The presentation in the paper uses abstractions in the form of hyper-rectangles (*boxes*), but also discusses the case where polyhedra are used instead. Moreover, we study the use of such slices in safety verification methods based on abstract refinement, especially constraint propagation based abstraction refinement (Ratschan and She, 2007). Extensive computational experiments show that in fact the efficiency of these methods becomes much more robust when using slices. Especially, the method can now verify the up to now unsolved heating benchmark (Fehnker and Ivančić, 2004).

In contrast to many algorithms in hybrid systems verification, the correctness of our algorithms cannot be hampered by floating-point rounding errors, since we do conservative rounding throughout.

Concerning related work, Sankaranarayanan et al. (2008) describe how to over-approximate the reach set of hybrid systems using hyper-planes with *given* coefficients. Their approach proved successful in *forward* computation of the reach set, repeating flow pipe constructions with a fixed time step. In contrast to that, in our approach we deduce not only the constant of the hyper-plane but also their coefficients. Moreover, we aim at improving *abstractions*, and not at forward computation. Hence we do not employ a potentially unbounded number of flow pipe constructions with a fixed time step, but try to infer single slices as fast as possible, allowing for efficient abstraction refinement.

Gulwani and Tiwari (2008) synthesize inductive invariants of hybrid systems. Whenever such an inductive invariant can be found, this verifies safety of the system. However, the resulting search for an invariant can be expensive: Gulwani and Tiwari (2008) use a translation to Boolean satisfiability modulo bit vectors here. In contrast to that, we concentrate on improvements of abstractions that can be computed in negligible time.

⋆ The work on this paper has been supported by GAČR grants 201/08/J020 and 201/09/H057, and by the institutional research plan AV0Z100300504 of the Czech Republic.

The content of the paper is as follows: In Section 2 we describe the problem of hybrid systems verification, in Section 3 we describe how to formulate constraints that over-approximate the reachable states, in Section 4 we describe our over-all approach of computing slices from the reachability constraint already used for computing abstractions and we estimate complexity of such computation, in Section 5 we evaluate the method using some computation experiments, and in Section 6 we conclude the paper.

2. SAFETY VERIFICATION OF HYBRID SYSTEMS

In this section, we briefly recall our formalism for modeling hybrid systems. It captures many relevant classes of hybrid systems, and many other formalisms for hybrid systems in the literature are special cases of it. We use a set S to denote the modes of a hybrid system, where S is finite and nonempty. $I_1, \ldots, I_k \subseteq \mathbb{R}$ are compact intervals over which the continuous variables of a hybrid system range. Φ denotes the state space of a hybrid system, i.e., $\Phi = S \times I_1 \times \cdots \times I_k$.

Definition 1. A *hybrid system* H is a tuple $(\texttt{Flow}, \texttt{Jump}, \texttt{Init}, \texttt{Unsafe})$, where $\texttt{Flow} \subseteq \Phi \times \mathbb{R}^k$, $\texttt{Jump} \subseteq \Phi \times \Phi$, $\texttt{Init} \subseteq \Phi$, and $\texttt{Unsafe} \subseteq \Phi$.

Informally speaking, the predicate `Init` specifies the initial states of a hybrid system and `Unsafe` the set of unsafe states that should not be reachable from an initial state. The relation `Flow` specifies the possible continuous flow of the system by relating states with corresponding derivatives, and `Jump` specifies the possible discontinuous jumps by relating each state to a successor state. Formally, the behavior of H is defined as follows:

Definition 2. A *flow* of length $l \geq 0$ in a mode $s \in S$ is a function $r : [0, l] \to \Phi$ such that the projection of r to its continuous part is differentiable and for all $t \in [0, l]$, the mode of $r(t)$ is s. A *trajectory* of H is a sequence of flows $r_0, \ldots, r_p$ of lengths $l_0, \ldots, l_p$ such that for all $i \in \{0, \ldots, p\}$,

(i) if $i > 0$ then $(r_{i-1}(l_{i-1}), r_i(0)) \in \texttt{Jump}$, and
(ii) if $l_i > 0$ then $(r_i(t), \dot{r}_i(t)) \in \texttt{Flow}$, for all $t \in [0, l_i]$, where $\dot{r}_i$ is the derivative of the projection of r_i to its continuous component.

A (concrete) *counterexample* of a hybrid system H is a trajectory $r_0, \ldots, r_p$ of H such that $r_0(0) \in \texttt{Init}$ and $r_p(l) \in \texttt{Unsafe}$, where l is the length of r_p. H is *safe* if it does not have a counterexample.

We use the following constraint language to describe hybrid systems and corresponding safety verification problems. The variable s ranges over S and the tuple of variables $\mathbf{x} = (\mathsf{x}_1, \ldots, \mathsf{x}_k)$ ranges over $I_1 \times \cdots \times I_k$, respectively. In addition, to denote the derivatives of $\mathsf{x}_1, \ldots, \mathsf{x}_k$ we use the tuple of variables $\dot{\mathbf{x}} = (\dot{\mathsf{x}}_1, \ldots, \dot{\mathsf{x}}_k)$ that ranges over $\mathbb{R}^k$,[1] and to denote the targets of jumps, we use the variable $\widehat{s}$ and the tuple of variables $\widehat{\mathbf{x}} = (\widehat{\mathsf{x}_1}, \ldots, \widehat{\mathsf{x}_k})$ that range over S and $I_1 \times \cdots \times I_k$, respectively. Constraints are arbitrary Boolean combinations of equalities and inequalities over terms that may contain function symbols, like $+$, $\times$, exp, sin, and cos.

[1] The dot does not have any special meaning here; it is only used to distinguish dotted from undotted variables.

We assume in the remainder of the text that a hybrid system is described by our constraint language. That means, the flows of a hybrid system are given by a constraint $\texttt{Flow}(\mathsf{s}, \mathbf{x}, \dot{\mathbf{x}})$, the jumps are given by a constraint $\texttt{Jump}(\mathsf{s}, \mathbf{x}, \widehat{\mathsf{s}}, \widehat{\mathbf{x}})$, and the initial and unsafe states are given by constraints $\texttt{Init}(\mathsf{s}, \mathbf{x})$ and $\texttt{Unsafe}(\mathsf{s}, \mathbf{x})$, respectively. To simplify notation, we do not distinguish between a constraint and the set it represents.

Example 1. For illustrating the above definitions, consider the following simple hybrid system with the modes m_1, m_2 and the continuous variables x_1, x_2, where x_1 ranges over the interval $[0, 2]$ and x_2 over $[0, 1]$, i.e, $\Phi = \{m_1, m_2\} \times [0, 2] \times [0, 1]$.

The set of initial states are given by the constraint $\texttt{Init}(s, (x_1, x_2)) \equiv (s = m_1 \wedge x_1 = 0 \wedge x_2 = 0)$. The constraint $\texttt{Unsafe}(s, (x_1, x_2)) \equiv (x_1 > 1 \wedge x_2 = 1)$ describes the set of unsafe states. The hybrid system can switch modes from m_1 to m_2 if $x_2 \geq 1$, i.e., $\texttt{Jump}(s, (x_1, x_2), s', (x_1', x_2')) \equiv (s = m_1 \wedge x_2 \geq 1 \to s' = m_2 \wedge x_1' = x_1 \wedge x_2' = x_2)$, The continuous behavior is very simple: In mode m_1, the values of the variables x_1, x_2 change with slope 1; in mode m_2, the slope of variable x_1 is 1 and variable x_2 has slope -1. For a flow in mode m_1, the constraint $0 \leq x_1 \leq 1$ must hold and in mode m_2, $1 \leq x_1 \leq 2$ must hold. The corresponding flow constraint is $\texttt{Flow}(s, (x_1, x_2), (\dot{x}_1, \dot{x}_2)) \equiv (s = m_1 \to \dot{x}_1 = 1 \wedge \dot{x}_2 = 1 \wedge 0 \leq x_1 \leq 1) \wedge (s = m_2 \to \dot{x}_1 = 1 \wedge \dot{x}_2 = -1 \wedge 1 \leq x_1 \leq 2)$. Note that the constraint $0 \leq x_1 \leq 1$ in `Flow` forces a jump from mode m_1 to m_2 if x_1 becomes 1. In general, an invariant that has to hold in a mode can be modeled by formulating a flow constraint that does not allow a continuous behavior in certain regions.

Obviously, this hybrid system is safe.

A *box abstraction* of a hybrid system H is a set of non-overlapping mode/box pairs (which we call *abstract states*) with transitions between them, such that:

- every point on a counterexample of H is an element of an abstract state
- whenever a counterexample moves from an abstract state a_1 to an abstract state a_2 then there is a corresponding transition $a_1 \to a_2$ from a_1 to a_2 (it is an easy, but rather technical exercise to formally define "moves from").

We assume that we have a method for computing such a box abstraction (Ratschan and She, 2007; Stursberg and Kowalewski, 2000).

3. REACHABILITY CONSTRAINTS

Now we assume that we have a box abstraction for a given hybrid system H, and present a constraint formalizing the fact that a given point in the state space of H may lie on a counterexample.

Observe that a point lies on a counterexample iff it is both reachable from an initial state *and* leads to an unsafe state. We will only formalize the first part of this condition, the second part is dual. Details can be found in an earlier