Preprints of the 2nd IFAC Conf. on Analysis and Design of Hybrid Systems (Alghero, Italy), 7-9 June 2006

VERIFICATION-INTEGRATED FALSIFICATION OF NON-DETERMINISTIC HYBRID SYSTEMS

Stefan Ratschan * **Jan-Georg Smaus** **

* *Max-Planck-Institut für Informatik, Saarbrücken, Germany*
** *Albert-Ludwigs-Universität Freiburg, Germany*

Abstract: This paper provides a method for coupling safety verification algorithms for non-deterministic (and, in general, non-linear) hybrid systems with the ability of finding concrete counterexamples, i.e., with falsification. Such a tight integration of verification with falsification has the advantage that verification attempts guide the search for concrete counterexamples, and endless attempts to verify unsafe systems or to find counterexamples in safe systems can often be avoided.

Keywords: hybrid systems, verification, falsification, simulation, testing

1. INTRODUCTION

For debugging designs of embedded systems, it is essential to be able to falsify hybrid systems, i.e., to find concrete counterexamples to a given property (e.g., safety). This is usually done by checking the results of simulation runs. However, such an approach has the disadvantage that one might continue looking for counterexamples although there are none, since the system fulfills the sought property. Also, simulation usually requires considerable user intelligence for avoiding redundant runs, determining the necessary time horizon, and canceling obviously useless simulation runs. Moreover, for systems with non-deterministic evolution, even from a given starting point there are uncountably many different trajectories that might be a counterexample. Hence it usually does not suffice to simulate with a fixed instantiation of the non-deterministic values in the specification of the given hybrid system, since these values might change over time (e.g., due to changes of the system input or changes in the environment).

In this paper, we consider non-deterministic systems, and take a step towards avoiding these problems by an approach that is integrated into verification, and that solves for counterexamples instead of blindly searching for them in the space of all possibilities opened by non-determinism.

The verification we have in mind is based on a finite abstraction of the infinite state space of a hybrid system. The safety of the abstract system implies the safety of the original system, and by successively refining the abstraction, it is often (but not always, due to resource and decidability limitations) possible to find an abstraction that *is* safe, provided the original system is safe. Thus, by integrating the method presented here into such an abstraction refinement loop, on the one hand we avoid endless refinement for systems that have an unsafe trajectory, on the other hand we avoid endless search for an unsafe trajectory for safe systems. Moreover, by using information computed by the verification engine, the approach

[1] This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See `www.avacs.org` for more information.

can often avoid trying to find counterexamples in parts of the state space that do not lead to an unsafe state. Furthermore, by following the state-space partition of the verification engine instead of the usual time partitioning of simulation, the approach avoids redundant recomputations for the same part of the state space, and is not restricted to a certain time horizon.

By *non-determinism*, we refer to systems where the flow of the real variables within one control mode is not uniquely determined, but rather, this flow has a whole set of evolution possibilities at any time point. In this case, there is hope that a flow described by piecewise polynomial functions fulfills the (in general, non-linear) flow conditions. This observation is the basis of our method. We formalize constraints that imply the existence of a trajectory leading from an initial to an unsafe state, and solve these constraints, arriving at connection points between the pieces.

The limitation of our method to non-deterministic systems can also be interpreted as follows: given a system that is deterministic (the flows are uniquely determined), one could relax the flow constraints to a certain extent to make the system non-deterministic, and then apply our method. If an error path is then found, one can argue that the original system is very close to an unsafe one. From a practical perspective, similar conclusions arise in both cases: one could either say, to be on the safe side, one should assume that systems behave non-deterministically within a certain range, or, to be on the safe side, one should reject systems that are close to an unsafe one.

2. METHOD

We assume that the hybrid system is given using a tuple of families of constraints

$$(\mathit{Init}_{m\in M}, \mathit{Flow}_{m\in M}, \mathit{Jump}_{m,m'\in M}, \mathit{Unsafe}_{m\in M}),$$

where M denotes the set of discrete modes, Init_m specifies the set of initial states in mode m, the constraint Flow_m the possible continuous flows in mode m, the constraint $\mathit{Jump}_{m,m'}$ the possible discontinuous jumps from mode m to mode m' and Unsafe_m the set of unsafe states in mode m. These constraints may contain elements of a finite set X of continuous state variables. The flow constraints may in addition contain dotted versions of the continuous state variables that denote their differentiation, and the jump constraints may in addition contain primed versions of these variables that denote their value after the jump. As examples, consider the flow constraint $\sin x - 1 \leq \dot{x} \wedge \dot{x} \leq \sin x + 1$, and the jump constraint $x' = x + 1$ which denotes the fact that the jump increases the variable x by one.

The systems we consider are *non-deterministic* in the sense that the flow constraints are typically inequalities, as in the example just given, rather than equations. We do not need to define precisely when a system is (non-)deterministic or give some measure for the degree of non-determinism. It suffices to note that the more non-deterministic the system is, the more likely it is that our method will find an unsafe trajectory.

Concerning the verification, we assume an abstraction refinement algorithm (Clarke *et al.* 2003, Alur *et al.* 2003, Ratschan and She 2005) for verifying safety properties of hybrid systems. Such methods decompose the state space into pieces, and based on that, construct a finite system (the *abstraction*) whose safety implies the safety of the original (*concrete*) system. The abstraction consists of *abstract states*, which are pairs consisting of a mode (i.e., an element of M) and a subset of the continuous state space (i.e., a subset of $\mathbb{R}^{|X|}$). How these sets are represented depends on the verification technique, e.g., they could be polyhedra or hyper-rectangles—however, the material of this section is independent of the chosen representation.

Given an abstract counterexample consisting of a sequence of abstract states $(m_0, B_0), \ldots, (m_k, B_k)$ such that (m_0, B_0) is initial and (m_k, B_k) is unsafe in the abstraction, we try to find points $(m_0, \boldsymbol{x}_0), \ldots, (m_k, \boldsymbol{x}_k)$ such that $\boldsymbol{x}_i \in B_i$ for all $i \in \{0, \ldots, k\}$, and such that there is a piecewise polynomial trajectory of the hybrid system that follows $(m_0, \boldsymbol{x}_0), \ldots, (m_k, \boldsymbol{x}_k)$, with $(m_0, \boldsymbol{x}_0)$ initial, and $(m_k, \boldsymbol{x}_k)$ unsafe. By *piecewise polynomial*, we mean that the trajectory within one mode is a continuous curve composed of (in general) several polynomial pieces.

So we try to find a solution to the constraint

$$\begin{aligned}\Big(\bigwedge_{i\in\{1,\ldots,k\}} \boldsymbol{x}_i \in B_i\Big) \;\wedge\; \mathit{Init}_{m_0}(\boldsymbol{x}_0) \;\wedge\\ \bigwedge_{i\in\{1,\ldots,k\}} \big(\mathit{Jump}_{m_{i-1},m_i}(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) \vee\\ (m_{i-1} = m_i \wedge \mathit{flow}_{m_i}(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i))\big) \;\wedge\\ \mathit{Unsafe}_{m_k}(\boldsymbol{x}_k) \quad (1)\end{aligned}$$

which we call the *concretization constraint*. This uses the constraints defining the given hybrid system (the ones with uppercase first letter), except the constraint defining the flow, since it contains the differentiation operator, which we would like to avoid. Since we assume that our input system is non-deterministic, the set of trajectories from $\boldsymbol{x}_{i-1}$ to $\boldsymbol{x}_i$ fulfilling Flow_{m_i} will—in general—consist of a central trajectory together with a certain neighborhood around it. Hence, one can find a polynomial approximation of the central trajectory that also fulfills Flow_{m_i}. If the neighborhood

is sufficiently large or $\boldsymbol{x}_{i-1}$ and $\boldsymbol{x}_i$ sufficiently close, the polynomial trajectory might even be linear.

So we assume a polynomial function $\phi_{\boldsymbol{a}}$ that gives us for a given time point in an interval $[0, \tau]$ the corresponding value of the trajectory. Here, $\boldsymbol{a}$ is a vector of parameters (real numbers) that specifies $\phi_{\boldsymbol{a}}$ in some way that we deliberately leave open at this point. We will later be more specific.

Hence we define $\mathit{flow}_m(\boldsymbol{x}, \boldsymbol{y}) \equiv$

$$\exists \boldsymbol{a} \exists \tau > 0 \left[\mathit{traj}(\boldsymbol{x}, \boldsymbol{y}, \phi_{\boldsymbol{a}}, \tau) \wedge \mathit{constr}_m(\phi_{\boldsymbol{a}}, \tau) \right]. \tag{2}$$

Here $\mathit{traj}(\boldsymbol{x}, \boldsymbol{y}, \phi_{\boldsymbol{a}}, \tau)$ models the fact that function $\phi_{\boldsymbol{a}}$ leads to a trajectory of length τ from $\boldsymbol{x}$ to $\boldsymbol{y}$, and $\mathit{constr}_m(\phi_{\boldsymbol{a}}, \tau)$ models the fact that the trajectory $\phi_{\boldsymbol{a}}$ fulfills the differential constraint Flow_m valid in mode m in the time interval $[0, \tau]$.

So we define $\mathit{traj}(\boldsymbol{x}, \boldsymbol{y}, \phi_{\boldsymbol{a}}, \tau) \equiv$

$$\phi_{\boldsymbol{a}}(0) = \boldsymbol{x} \wedge \phi_{\boldsymbol{a}}(\tau) = \boldsymbol{y},$$

and $\mathit{constr}_m(\phi_{\boldsymbol{a}}, \tau) \equiv$

$$\forall t \in [0, \tau] \left[\mathit{Flow}_m(\phi_{\boldsymbol{a}}(t), \dot{\phi}_{\boldsymbol{a}}(t)) \right], \tag{3}$$

where Flow_m is the constraint defining the flow of the given hybrid system in mode m.

Now observe that we have to solve the constraint $\mathit{flow}_m(\boldsymbol{x}, \boldsymbol{y})$ many times for a given input system, since it occurs k times in (1), and since (1) has to be solved over and over again in the verification/falsification loop sketched in the introduction. Hence we would like to arrive at a formulation of $\mathit{flow}_m(\boldsymbol{x}, \boldsymbol{y})$ for which it is as easy as possible to check whether it has a solution.

Observe that $\mathit{flow}_m(\boldsymbol{x}, \boldsymbol{y})$ contains several quantifiers. Assume that we have a given, fixed input system containing only polynomial constraints with a certain constraint Flow_m defining the possible flow in mode m, and a class of functions $\phi_{\boldsymbol{a}}$ containing only polynomials, in which we search for concrete trajectories. Then, due to the classical result (Tarski 1951) that the first-order theory of the real numbers with addition and multiplication admits quantifier elimination, it is always effectively possible to find an equivalent formula without quantifiers.

In spite of the fact that quantifier-free constraints are easier to check than constraints with quantifiers, we do not opt for automatic quantifier elimination since it can be very costly and lead to a big increase in formula size. Instead, we will try to identify problem classes corresponding to special cases for these constraints for which manual quantifier elimination will yield simple and useful results.

An especially important class is the class where $\phi_{\boldsymbol{a}}$ is linear, i.e., $\phi_{\boldsymbol{a}_1, \boldsymbol{a}_0}(t) = \boldsymbol{a}_1 t + \boldsymbol{a}_0$, where $\boldsymbol{a}_1 \in \mathbb{R}^{|X|}$, $\boldsymbol{a}_0 \in \mathbb{R}^{|X|}$.

So we have: $\mathit{traj}(\boldsymbol{x}, \boldsymbol{y}, \phi_{\boldsymbol{a}_1, \boldsymbol{a}_0}, \tau) \equiv$

$$\begin{aligned} \phi_{\boldsymbol{a}_1, \boldsymbol{a}_0}(0) = \boldsymbol{x} \wedge \phi_{\boldsymbol{a}_1, \boldsymbol{a}_0}(\tau) = \boldsymbol{y} \equiv \\ \boldsymbol{a}_0 = \boldsymbol{x} \wedge \boldsymbol{a}_1 \tau + \boldsymbol{a}_0 = \boldsymbol{y}, \end{aligned} \tag{4}$$

and $\mathit{constr}_m(\phi_{\boldsymbol{a}_1, \boldsymbol{a}_0}, \tau) \equiv$

$$\forall t \in [0, \tau] \left[\mathit{Flow}_m(\boldsymbol{a}_1 t + \boldsymbol{a}_0, \boldsymbol{a}_1) \right] \tag{5}$$

Now we can solve (4) for $\boldsymbol{a}_0$ and $\boldsymbol{a}_1$, and substitute the resulting terms in (5) which thus becomes

$$\forall t \in [0, \tau] \left[\mathit{Flow}_m(\frac{\boldsymbol{y} - \boldsymbol{x}}{\tau} t + \boldsymbol{x}, \frac{\boldsymbol{y} - \boldsymbol{x}}{\tau}) \right],$$

and (2) becomes

$$\exists \tau > 0 \forall t \in [0, \tau] \left[\mathit{Flow}_m(\frac{\boldsymbol{y} - \boldsymbol{x}}{\tau} t + \boldsymbol{x}, \frac{\boldsymbol{y} - \boldsymbol{x}}{\tau}) \right]. \tag{6}$$

We call *Flow convex* if for every vector $\boldsymbol{c} \in \mathbb{R}^{2|X|}$, and vector $\boldsymbol{d} \in \mathbb{R}^{2|X|}$, if $\mathit{Flow}(\boldsymbol{c})$ and $\mathit{Flow}(\boldsymbol{d})$ hold, then for every $\lambda \in [0, 1]$, also $\mathit{Flow}(\lambda \boldsymbol{c} + (1 - \lambda)\boldsymbol{d})$ holds.

In general, flow constraints of the form $\mathit{Flow}(\boldsymbol{x}, \dot{\boldsymbol{x}}) \equiv \underline{A}\boldsymbol{x} + \underline{\boldsymbol{b}} \leq \dot{\boldsymbol{x}} \leq \overline{A}\boldsymbol{x} + \overline{\boldsymbol{b}}$, where $\underline{A}, \overline{A}$ are a $|X| \times |X|$-matrices, and $\underline{b}, \overline{b}$ are vectors in $\mathbb{R}^{|X|}$, are convex. This follows from the trivial arithmetic facts that $e \leq e'$ implies $\lambda e \leq \lambda e'$, for every $\lambda \geq 0$, and that $e_1 \leq e_1' \wedge e_2 \leq e_2'$ implies $e_1 + e_2 \leq e_1' + e_2'$.

Now if $\phi_{\boldsymbol{a}}$ is a linear function and Flow_m is convex, then (3) is equivalent to

$$\mathit{Flow}_m(\phi_{\boldsymbol{a}}(0), \dot{\phi}_{\boldsymbol{a}}(0)) \wedge \mathit{Flow}_m(\phi_{\boldsymbol{a}}(\tau), \dot{\phi}_{\boldsymbol{a}}(\tau)),$$

i.e., the universal quantification can be replaced by a conjunction referring to the endpoints of the quantified interval. Note that just convexity of Flow_m or linearity of $\phi_{\boldsymbol{a}}$ alone are not sufficient.

Thus (6) can be simplified to

$$\exists \tau > 0 \left[\mathit{Flow}_m(\boldsymbol{x}, \frac{\boldsymbol{y} - \boldsymbol{x}}{\tau}) \wedge \mathit{Flow}_m(\boldsymbol{y}, \frac{\boldsymbol{y} - \boldsymbol{x}}{\tau}) \right] \tag{7}$$

Now we assume, in addition to the linearity of $\phi_{\boldsymbol{a}}$, that Flow_m is of the form $\underline{F}(\boldsymbol{x}) \leq \dot{\boldsymbol{x}} \leq \overline{F}(\boldsymbol{x})$. So (7), i.e. (2), becomes

$$\underline{F}(\boldsymbol{x}) \leq \frac{\boldsymbol{y} - \boldsymbol{x}}{\tau} \leq \overline{F}(\boldsymbol{x}) \wedge \underline{F}(\boldsymbol{y}) \leq \frac{\boldsymbol{y} - \boldsymbol{x}}{\tau} \leq \overline{F}(\boldsymbol{y})$$

that is

$$\tau \underline{F}(\boldsymbol{x}) \leq \boldsymbol{y} - \boldsymbol{x} \leq \tau \overline{F}(\boldsymbol{x}) \wedge \tau \underline{F}(\boldsymbol{y}) \leq \boldsymbol{y} - \boldsymbol{x} \leq \tau \overline{F}(\boldsymbol{y})$$

If we again take the case $\underline{F}(\boldsymbol{x}) = \underline{A}\boldsymbol{x} + \underline{\boldsymbol{b}}$, $\overline{F}(\boldsymbol{x}) = \overline{A}\boldsymbol{x} + \overline{\boldsymbol{b}}$, we get

$$\tau \underline{A}\boldsymbol{x} + \underline{\boldsymbol{b}} \leq \boldsymbol{y} - \boldsymbol{x} \leq \tau \overline{A}\boldsymbol{x} + \overline{\boldsymbol{b}} \wedge \tau \underline{A}\boldsymbol{y} + \underline{\boldsymbol{b}} \leq \boldsymbol{y} - \boldsymbol{x} \leq \tau \overline{A}\boldsymbol{y} + \overline{\boldsymbol{b}}$$

that is

$$\begin{aligned} \boldsymbol{x} + \tau \underline{A}\boldsymbol{x} + \underline{\boldsymbol{b}} \leq \boldsymbol{y} \leq \boldsymbol{x} + \tau \overline{A}\boldsymbol{x} + \overline{\boldsymbol{b}} \wedge \\ \boldsymbol{y} - \tau \overline{A}\boldsymbol{y} + \underline{\boldsymbol{b}} \leq \boldsymbol{x} \leq \boldsymbol{y} - \tau \underline{A}\boldsymbol{y} + \overline{\boldsymbol{b}} \end{aligned}$$

and equivalently

$$(I+\tau\underline{A})\boldsymbol{x}+\underline{\boldsymbol{b}} \leq \boldsymbol{y} \leq (I+\tau\overline{A})\boldsymbol{x}+\overline{\boldsymbol{b}} \wedge$$
$$(I-\tau\overline{A})\boldsymbol{y}+\underline{\boldsymbol{b}} \leq \boldsymbol{x} \leq (I-\tau\underline{A})\boldsymbol{y}+\overline{\boldsymbol{b}}$$

One might want to derive special methods for this specific constraint, however this is beyond the scope of the current paper. Considering the concretization constraint (1) and its special forms shown above, we know: For a given abstract counterexample, if there is a solution to the corresponding concretization constraint, then there is a trajectory from an initial to an unsafe state following this abstract counterexample, i.e., a concrete counterexample. We will show how to solve the concretization constraint in the next section.

3. CONSTRAINT SOLVING

We only consider cases where the universal quantifier of (3) has been eliminated (due to convexity), and where $\phi_{\boldsymbol{a}}$ is linear (although $Flow_m$, while convex, might still be non-linear). Then we are left with (7) whose quantifiers are only existential. For finding solutions, it suffices to drop the existential quantifiers, and finding a solution in all variables, including the ones that were existentially quantified before.

One could solve the concretization constraint using numerical local optimization techniques. However, these need good starting points to be able to find a solution. For this we use interval constraint propagation techniques (Davis 1987, Cleary 1987). These can, given an interval for each variable in a constraint, prune these intervals to smaller ones without losing any solution of the constraint. After constraint propagation we can use the midpoints of the resulting intervals as starting points for local optimization.

The most basic interval constraint propagation method decomposes all atomic constraints (i.e., constraints of the form $t \geq 0$ or $t = 0$, where t is a term) into conjunctions of so-called primitive constraints (i.e., constraints such as $x + y = z$, $xy = z$, $z \in [\underline{a}, \overline{a}]$, or $z \geq 0$, where x, y, z are variables) by introducing additional auxiliary variables (e.g., decomposing $x + \sin y \geq 0$ to $\sin y = v_1 \wedge x + v_1 = v_2 \wedge v_2 \geq 0$). Then it applies interval arithmetic based algorithms (Hickey *et al.* 1998) for pruning the intervals corresponding to the variables occurring in the primitive constraints to smaller ones that still contain the solution set of these primitive constraints. This is done until a fixpoint is reached.

Usually constraint propagation can only compute an overapproximation of the exact bounds on the solution set of constraints (this is also known as the *dependency problem* of interval arithmetic). For example, in the case $x + y = 0 \wedge x - y = 0$ and a starting interval $[-1, 1]$ for both x and y, these intervals cannot be pruned, although there is only one solution—the origin. Still, for certain cases (e.g., for constraints in which no variable occurs more than once), the result of constraint propagation will be tight. Hence, in this case, if constraint propagation yields non-empty intervals, the constraint has a solution, and we do not need numerical local optimization at all.

However, also in our application interval constraint propagation is not necessarily tight: Assume a 2-dimensional system with just one mode m, whose flow constraint $Flow_m$ is given by $-x_1 - 1 \leq \dot{x_1} \leq -x_1 + 1 \wedge \dot{x_2} = 1$, and assume interval vectors $\boldsymbol{x} = ([0,0],[0,0])$ and $\boldsymbol{y} = ([-5,5],[5,5])$. Then propagation of (7) wrt. the second conjunct of $Flow_m$ will result in the interval $\tau \in [5,5]$. Further propagation will yield the interval $[-1.25, 1.25]$ for the first component of $\boldsymbol{y}$. However, computation of the explicit solution of the borderline cases of $Flow_m$ shows that this is an overestimation of approximately 25%. For example, for $\boldsymbol{y} = (1.25, 5)$, the trajectory would have the slope $(0.25, 1)$, and these values do not satisfy $Flow_m$. The problem comes from the fact that the slope $(0.25, 1)$ satisfies $Flow_m$ in combination with another point within the computed bounds, namely the point $\boldsymbol{y} = (-1.25, 5)$. However, due to the enclosure of variables in intervals, the precise depency of specific values for $\boldsymbol{y}$ and a specific slope of the trajectory is lost.

4. IMPLEMENTATION AND EXAMPLES

We integrated a prototype implementation of the method into the verification engine HSolver (Ratschan and She 2005, Ratschan and She 2004), which does abstraction refinement by incrementally refining a decomposition of the state space into hyper-rectangles, i.e., by bisecting hyper-rectangles. After each refinement step we apply our method to the current abstraction.

However, this abstraction might have a huge number of abstract counterexamples to check. For avoiding this, we observe that many of them share the same sub-sequence (a *fragment*); if there is no trajectory through a given fragment, then there is also no trajectory through any of the abstract counterexamples containing the fragment. Hence we also check fragments, using a version of the concretization constraint that does not contain the first line (specifying an initial state) or last line (specifying an unsafe state), if the corresponding sequence of abstract states does not start with an initial state, or end in an unsafe state, respectively.

In the current version, we produce fragments by recursively transversing the abstraction from ini-

tial to unsafe states and checking the corresponding fragments. If the given fragment cannot be concretized we backtrack, refine the abstraction, and avoid checking the abstract counterexamples containing this fragment.

The current prototype does not yet consider jumps, and instead of doing numerical local optimization (which would be difficult to integrate due to the constraint representation used by HSolver) to find a solution to the concretization constraint, it simply iterates taking the midpoint of some interval enclosing the solution set and doing interval constraint propagation using the solver RSolver (`http://rsolver.sourceforge.net`). In lucky cases this already results in a solution. Since constraint propagation does not enclose the solution set closely, also the unlucky case that this midpoint is not a solution happens quite often. This would make numerical optimization necessary. However, the fact that already such a prototype can solve several examples shows the strength of the method.

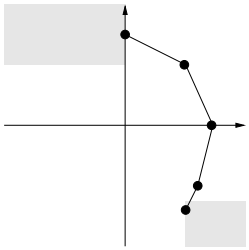
We tried the following examples:

Example 1: A very simple example where trajectories follow circles around the origin:

$$x_2 - 1 \leq \dot{x_1} \leq x_2 + 1$$
$$-x_1 - 1 \leq \dot{x_2} \leq -x_1 + 1$$

The set of initial states is given by $x_1 \leq 0, x_2 \geq 2$, the set of unsafe states by $x_1 \geq 2, x_2 \leq -2.5$, and the state space $[-4, 4] \times [-4, 4]$. The fact that a counterexample has to flow around a half-circle makes this example non-trivial.

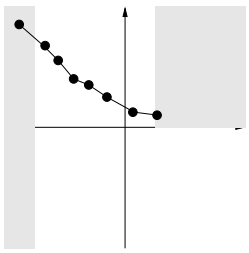
After 10 bisections of the state space, in negligible time, the method finds the counterexample $(0, 3), (2, 2), (2, 2), (2.959, 0), (2.322, -2), (2, -2.57)$. The repeated point stems from two different, neighboring abstract states.



Example 2: A non-deterministic version of an example from (Kapinski *et al.* 2003):

$$x_1 + 3x_2 - 1 \leq \dot{x_1} \leq x_1 + 3x_2 + 1$$
$$x_1 - x_2 - 1 \leq \dot{x_2} \leq x_1 - x_2 + 1$$

The set of initial states is given by $x_1 \leq -3$, the set of unsafe states by $x_1 \geq 1, x_2 \geq 0$, and the state space $[-4, 4] \times [-4, 4]$. After 75 bisections, in a few seconds, the method finds the counterexample $(-3.5, 3.5)$, $(-2.75, 2.75)$, $(-2.25, 2.288)$, $(-1.75, 1.686)$, $(-1.25, 1.355)$, $(-0.5, 1)$, $(-0.5, 1)$, $(-0.5, 1)$, $(0.25, 0.464)$, $(1, 0.440)$.



Example 3: A non-deterministic version of another example from (Kapinski *et al.* 2003):

$$x_2 - 1.5 \leq \dot{x_1} \leq x_2 + 1.5$$
$$-8x_1 - 1.5 \leq \dot{x_2} \leq -8x_1 + 1.5$$
$$x_4 - 1.5 \leq \dot{x_3} \leq x_4 + 1.5$$
$$-4x_3 - 1.5 \leq x_4 \leq -4x_3 + 1.5$$

The set of initial states is given by $x_1^2 + x_2^2 + x_3^2 + x_4^2 \leq 4$ the set of unsafe states by $x_1^2 + x_2^2 + x_3^2 + x_4^2 \geq 9$, and the state space by $[-4, 4] \times [-4, 4] \times [-4, 4] \times [-4, 4]$. After 16 bisections, but a few minutes of computation, the method finds the counterexample $(2, 0, 0, 0)$, $(2, 0, 0, 0)$, $(1.792, -2.64, -0.155, -0.155)$.

5. RELATED WORK

Apart from manual simulation using according tools (Lee and Zheng 2005, and references therein), the main approach for falsifying safety of dynamical systems is bounded model checking (BMC). This approach is well developed for discrete systems (Biere *et al.* 1999), but only lately receives attention in the hybrid systems cases (Fränzle and Herde 2004, Ábrahám *et al.* 2005). However, the current approaches are limited to dynamics formulated by linear inequalities (i.e., the solution of constant differential inclusions), and there is no systematic way of using bounded model checking for obtaining concrete counterexamples for hybrid systems whose dynamics is given by more interesting (e.g., linear) differential (in)equations. Moreover, BMC might redundantly compute similar information for the same part of the start space, and explores branches that might never lead to an unsafe state. We avoid this by taking advantage of the information generated in the verification algorithm.

For linear discrete-time systems with input, another approach (Kapinski *et al.* 2003) exhaustively simulates the system using a time and state discretization of the input, and tries to avoid redundant computation by merging nearby trajectories. Our approach avoids time discretization by searching for a trajectory between two given points in the state space that may be of unbounded length in time.

Bhatia and Frazzoli (2004) adapt techniques from robotic motion planning to compute an under-

approximation of the set of trajectories of a given hybrid system. Prajna and Rantzer (2005) show that—in analogy to Lyanpunov functions—the existence of certain functions implies the reachability of given sets in (non-linear) ODEs. Such functions can be computed using sum-of-squares (SOS) tools (S. Prajna 2002).

Tools for counterexample guided abstraction refinement based on flow pipe computation (Clarke *et al.* 2003) may terminate with a concrete counterexample. They can deal with non-determinism only in the form of parametric systems, where different values for these parameters are tried to search for trajectories that happen to be counterexamples. This fixes the parameter to a certain value between switches. In contrast to that, our approach the trajectory can take *any* solution of the differential constraint, also solutions that might correspond to inputs or disturbances that change between switches.

6. CONCLUSION

We provided a first version of a method for coupling hybrid systems verification algorithms with the ability to find concrete counterexamples for non-deterministic hybrid systems, i.e., with *falsification*. The advantage of the method is that it uses information from system abstractions computed by the verification algorithm to guide the search for a counterexample.

In future work we will provide an efficient implementation of the method based on local optimization, we will try to come up with special, efficient constraint solving algorithms for the special cases of the concretization constraint that we have identified in this paper, we will avoid a full re-check of all the counterexamples after each refinement step by re-using information from earlier checks, and we will provide efficient strategies for the order in which the abstract counterexamples (and their fragments) should be checked. Moreover, we will work on similar methods for hybrid systems without non-determinism.

REFERENCES

Ábrahám, E., B. Becker, F. Klaedtke and M. Steffen (2005). Optimizing bounded model checking for linear hybrid systems. In: *VMCAI* (R. Cousot, Ed.). Vol. 396–412 of *LNCS*. Springer. pp. 396–412.

Alur, R., T. Dang and F. Ivančić (2003). Counter-example guided predicate abstraction of hybrid systems. In: *TACAS* (H. Garavel and J. Hatcliff, Eds.). Vol. 2619 of *LNCS*. Springer. pp. 208–223.

Bhatia, A. and E. Frazzoli (2004). Incremental search methods for reachability analysis of continuous and hybrid systems. In: *HSCC'04* (Rajeev Alur and George J. Pappas, Eds.). number 2993 In: *LNCS*. Springer.

Biere, A., A. Cimatti, E. M. Clarke and Y. Zhu (1999). Symbolic model checking without BDDs. In: *TACAS '99*. Springer. pp. 193–207.

Clarke, E., A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg and M. Theobald (2003). Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. Journal of Foundations of Comp. Science* **14**(4), 583–604.

Cleary, J. G. (1987). Logical arithmetic. *Future Computing Systems* **2**(2), 125–149.

Davis, E. (1987). Constraint propagation with interval labels. *Artif. Intell.* **32**(3), 281–331.

Fränzle, M. and C. Herde (2004). Efficient proof engines for bounded model checking of hybrid systems. In: *FMICS 04, Electr. Notes in Theor. Comp. Sc. (ENTCS)*. Elsevier.

Hickey, T. J., M. H. van Emden and H. Wu (1998). A unified framework for interval constraint and interval arithmetic. In: *CP'98* (M. Maher and J.F. Puget, Eds.). number 1520 In: *LNCS*. Springer. pp. 250–264.

Kapinski, J., B. H. Krogh, O. Maler and O. Stursberg (2003). On systematic simulation of open continuous systems. In: *HSCC'03* (Oded Maler and Amir Pnueli, Eds.). Vol. 2623 of *LNCS*. Springer.

Lee, E. A. and H. Zheng (2005). Operational semantics of hybrid systems. In: Morari and Thiele (2005).

Morari, M. and Thiele, L., Eds. (2005). *Hybrid Systems: Computation and Control*. Vol. 3414 of *LNCS*. Springer.

Prajna, S. and A. Rantzer (2005). Primal-dual tests for safety and reachability. In: Morari and Thiele (2005).

Ratschan, Stefan and Zhikun She (2004). HSolver. `http://hsolver.sourceforge.net`. Software package.

Ratschan, Stefan and Zhikun She (2005). Safety verification of hybrid systems by constraint propagation based abstraction refinement. In: Morari and Thiele (2005).

S. Prajna, A. Papachristodoulou, P. A. Parrilo. (2002). Introducing SOSTOOLS: A general purpose sum of squares programming solver.. In: *CDC'02*.

Tarski, A. (1951). *A Decision Method for Elementary Algebra and Geometry*. Univ. of California Press. Berkeley.