Constraints for Continuous Reachability in the Verification of Hybrid Systems ⋆

Stefan Ratschan[1] and Zhikun She[2]

[1] Institute of Computer Science, Czech Academy of Sciences, Prague, Czech Republic
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. The method for verification of hybrid systems by constraint propagation based abstraction refinement that we introduced in an earlier paper is based on an over-approximation of continuous reachability information of ordinary differential equations using constraints that do not contain differentiation symbols. The method uses an interval constraint propagation based solver to solve these constraints. This has the advantage that—without complicated algorithmic changes—the method can be improved by just changing these constraints. In this paper, we discuss various possibilities of such changes, we prove some properties about the amount of over-approximations introduced by the new constraints, and provide some timings that document the resulting improvement.

1 Introduction

A hybrid system is a dynamical system that involves both continuous and discrete state and evolution. This can, for example, be used for modeling the behavior of an embedded (digital) computing device influencing its (continuous) environment. An important task is to verify that a given hybrid system is safe, that is, every trajectory starting from an initial state never reaches an unsafe state.

In this paper we study constraints that can be used for modeling the continuous flow in the safety verification of hybrid systems by constraint propagation based abstraction refinement [16, 14]. Especially, we exploit the fact that the underlying solver, which is based on interval constraint propagation as introduced within the field of artificial intelligence [5, 3, 9, 13], allows the use of a rich language of constraints that includes function symbols such as sin, cos, exp. These symbols arise naturally as solutions of linear differential equations.

More specifically, in this paper we study two types of constraints that model the reachability problem: one, for linear differential equations, is related to the explicit solution of such equations; and the second, for general differential constraints, is based on the mean-value theorem. Both constraints are quite simple

⋆ 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.

to derive and similar ones have been used before in the literature. However, here we study in detail, how they behave when using interval based constraint propagation techniques for solving them, and how this behavior reflects in our method of verification of hybrid systems using constraint propagation based abstraction refinement. Our verification software HSolver [15], allows the user to experiment with these, and additional, user-defined, constraints.

Regarding additional related work, the approach by Hickey and Wittenberg [7] puts the level of modelling even higher, by employing a constraint logic programming language [8] that directly can deal with differential equations. Internally it solves constraints by transforming them into polynomial constraints using Taylor expansion, and then solves these using a similar solver as ours. The approach does not provide a comparison of different formulations of these constraints, and does not employ abstraction refinement to concentrate on refining the solution which is relevant for a given safety verification problem.

Tiwari [19] derives simple polynomial constraints from linear differential equations by manually doing an over-approximating quantifier elimination on a similar constraint as one of the constraints employed here. In contrast to that, since our solver can handle function symbols such as sin, cos, and exp, we can directly work on the original constraint, and—using an abstraction refinement scheme—approximate it arbitrarily closely.

Anai and Weispfenning [1] provide a classification of the cases when the time variable can be symbolically eliminated from the solution of linear differential equations (which may contain transcendental function symbols).

Similar constraints as employed here, which are based on the mean-value theorem or Taylor expansion, are ubiquitous in the integration of ODE's.

The content of the paper is as follows: In Section 2 we review our method of verification using abstraction refinement; in Section 3 we discuss constraints for modeling reachability of differential equations; in Section 4 we discuss how we solve these constraints; in Section 5 we compare the constraints theoretically; in Section 6 we study empirically, how the constraints behave within our method; and in Section 7 we conclude the paper.

2 Constraint Propagation Based Abstraction Refinement

In this section, we review our previous approach [16, 14] for verifying safety using constraint propagation based abstraction refinement. We fix a variable s ranging over a finite set of discrete modes $\{\mathtt{s_1}, \ldots, \mathtt{s_n}\}$ and variables $x_1, \ldots, x_k$ ranging over closed real intervals $I_1, \ldots, I_k$. We denote by Φ the resulting state space $\{\mathtt{s_1}, \ldots, \mathtt{s_n}\} \times I_1 \times \cdots \times I_k$. In addition, for denoting the derivatives of $x_1, \ldots, x_k$ we assume variables $\dot{x}_1, \ldots, \dot{x}_k$, ranging over $\mathbb{R}$ each,and for denoting the targets of jumps, variables $s', x'_1, \ldots, x'_k$ ranging over $\{\mathtt{s_1}, \ldots, \mathtt{s_n}\}$ and $I_1, \ldots, I_k$, correspondingly. In the following we will use boldface to denote vectors of variables.

For describing hybrid systems, we use a flow constraint $Flow(s, \boldsymbol{x}, \dot{\boldsymbol{x}})$, a jump constraint $Jump(s, \boldsymbol{x}, s', \boldsymbol{x}')$, an initial constraint $Init(s, \boldsymbol{x})$ and an unsafety constraint $UnSafe(s, \boldsymbol{x})$. Now, assuming a given hybrid system $H =$

$(Flow, Jump, Init, Unsafe)$, our safety verification problem is to verify that H is safe, that is, there is no piecewise continuous trajectory of H that starts from an initial state (i.e., a state satisfying the constraint $Init(s, \boldsymbol{x})$ and reaches an unsafe state (i.e., a state satisfying the constraint $UnSafe(s, \boldsymbol{x})$). Here the continuous parts of the trajectory are required to fulfill the flow constraint $Flow(s, \boldsymbol{x}, \dot{\boldsymbol{x}})$, and the discontinuous jumps are required to fulfill $Jump(s, \boldsymbol{x}, s', \boldsymbol{x}')$, where $(s, \boldsymbol{x})$ refers to the state before the jump and $(s', \boldsymbol{x}')$ refers to the state directly after the jump. For formal definitions refer to our previous publication [16].

Our approach decomposes the state space Φ using hyper-rectangles (*boxes*) into finitely many mode/box pairs and then computes a finite over-approximation of the hybrid systems (an *abstraction*). In detail, for each pair of boxes, it sets an abstract transition, only if it cannot prove the absence of trajectories between them; also, it marks boxes as initial (unsafe), if it cannot prove the absence of an initial (unsafe) element in the box. If the resulting finite abstraction is safe, the hybrid system is also safe, since the abstraction over-approximates the hybrid system. If it is not safe, we refine the abstraction by splitting boxes into pieces and recomputing the affected information in the abstraction.

Moreover, we have a mechanism for removing unreachable elements from boxes. For this observe that a point in a box B is reachable only if it is reachable either from the initial set via a flow in B, from a jump via a flow in B, or from a neighboring box via a flow in B. So we formulate constraints corresponding to each of these conditions and then remove points from boxes that do not fulfill at least one of these constraints.

The approach can be used with any constraint describing that $\boldsymbol{y}$ can be reachable from $\boldsymbol{x}$ via a flow in B and mode s, for example, the one introduced in our previous publication and the new ones that will be introduced in the latter sections. We denote the used constraint by $Reachable_B(s, \boldsymbol{x}, \boldsymbol{y})$.

Thus the above three possibilities of reachability allow us to formulate the following theorem:

Theorem 1. *For a set of abstract states $\mathcal{B}$ such that all boxes corresponding to the same mode are non-overlapping, a pair $(s', B') \in \mathcal{B}$ and a point $z \in B'$, if $(s', \boldsymbol{z})$ is reachable, then*

$$\begin{aligned} initflow_{B'}(s', \boldsymbol{z}) \lor & \bigvee_{(s,B)\in\mathcal{B}} jumpflow_{B,B'}(s, s', \boldsymbol{z}) \\ & \lor \bigvee_{(s,B)\in\mathcal{B}, s=s', B\neq B'} boundaryflow_{B,B'}(s', \boldsymbol{z}) \end{aligned}$$

where $initflow_{B'}(s', \boldsymbol{z})$, $jumpflow_{B,B'}(s, s', \boldsymbol{z})$ and $boundaryflow_{B,B'}(s', \boldsymbol{z})$ denote the following three constraints, respectively:

- $\exists \boldsymbol{y} \in B' \, [Init(s', \boldsymbol{y}) \land Reachable_{B'}(s', \boldsymbol{y}, \boldsymbol{z})]$,
- $\exists \boldsymbol{x} \in B \exists \boldsymbol{x}' \in B' \, [Jump(s, \boldsymbol{x}, s', \boldsymbol{x}') \land Reachable_{B'}(s', \boldsymbol{x}', \boldsymbol{z})]$,
- $\exists \boldsymbol{x} \in B \cap B' \, \big[[\forall faces\ F\ of\ B[\boldsymbol{x} \in F \Rightarrow out^F_{s',B,B'}(\boldsymbol{x})]] \land Reachable_{B'}(s', \boldsymbol{x}, \boldsymbol{z})\big]$.

Here, $out^F_{s',B,B'}(\boldsymbol{x})$ is one of the following constraints:

– $\exists \dot{x}_j [Flow(s', \boldsymbol{x}, (\dot{x}_1, \dots, \dot{x}_k)) \wedge \dot{x}_j \leq 0]$, *if* F *is the* j*-th lower face of* B, *and*
– $\exists \dot{x}_j [Flow(s', \boldsymbol{x}, (\dot{x}_1, \dots, \dot{x}_k)) \wedge \dot{x}_j \geq 0]$, *if* F *is the* j*-th upper face of* B.

We denote the main constraint (i.e., the disjunction) by $reachable_{B'}(s', \boldsymbol{z})$. If we can prove that a certain point does not fulfill this constraint, we know that it is not reachable. In Section 4 we describe pruning algorithm that takes such a constraint and an abstract state (s', B'), and returns a sub-box of B' that still contains all the solutions of the constraint in B'.

Since the constraint $reachable_{B'}(s', \boldsymbol{z})$ depends on all current abstract states, a change of B' might allow further pruning of other abstract states. So we can repeat pruning until a fixpoint is reached. This terminates since we use floating point computation here and there are only finitely many floating point numbers.

We remove the initial mark of an abstract state (s', B') if we can disprove $initflow_{B'}(s', \boldsymbol{z})$ in Theorem 1, and we remove the unsafe mark of an abstract state (s', B') if we can disprove $\exists \boldsymbol{x} \in B'\ UnSafe(s', \boldsymbol{x})$. Moreover, we remove a transition from (s, B) to (s', B') if we can disprove both $boundaryflow_{B,B'}(s', \boldsymbol{z})$ and $jumpflow_{B,B'}(s, s', \boldsymbol{z})$ from Theorem 1. It is easy to show that the resulting system is an abstraction of the original hybrid system.

3 Constraints for Reachability

Assume that the flow constraint contains a differential equation of the form $\dot{\boldsymbol{x}} = A\boldsymbol{x}$, where $A \in \mathbb{Q}^{k \times k}$. Differential equations of the form $\dot{\boldsymbol{x}} = A\boldsymbol{x} + B$ can be reduced to that form by shifting the equilibrium, provided that the equation $A\boldsymbol{x} + B = 0$ has solutions. Given an initial set $Init$, we have the exact solution $\boldsymbol{x}(t) = e^{At}\boldsymbol{x}_0$, where $\boldsymbol{x}_0 \in Init$ and e^{At} is defined by $\sum_{k=0}^{\infty} \frac{t^k}{k!} A^k$. Thus, if $\boldsymbol{x}$ is reachable, then the constraint $\exists t \in \mathbb{R}_{\geq 0} \exists \boldsymbol{x}_0 \in \mathbb{R}^k [\boldsymbol{x}_0 \in Init \wedge \boldsymbol{x} = e^{At}\boldsymbol{x}_0]$ holds.

Since the matrix A appears in an exponent, it is difficult to directly solve this constraint. We use another constraint introduced by Tiwari [19], that over-approximates the reach set, that can be easily computed from the matrix A, and that does not contain matrix exponentiation. For this we re-express the real and complex eigenvalues of A^T (the transpose of A) using the following two sets:

$$\Lambda_1 = \left\{ \lambda \in \mathbb{R} \mid \exists c \in \mathbb{R}^k \left[c \neq 0 \wedge A^T c = \lambda c \right] \right\};$$
$$\Lambda_2 = \left\{ (a, b) \in \mathbb{R} \times \mathbb{R}_{>0} \mid \exists c \in \mathbb{R}^k \left[c \neq 0 \wedge ((A^T)^2 - 2aA^T + (a^2 + b^2)I)c = 0 \right] \right\}.$$

For every $\lambda \in \Lambda_1$, let $c(1, \lambda)$ be an orthonormal basis of $\{c : A^T c = \lambda c\}$; for every $(a, b) \in \Lambda_2$, let $c(2, (a, b))$ be an orthonormal basis of $\{c : ((A^T)^2 - 2aA^T + (a^2 + b^2)I)c = 0\}$. Then we can describe an over-approximation of the set of reachable states as follows.

Lemma 1. *For a differential equation* $\dot{\boldsymbol{x}} = A\boldsymbol{x}$ *and a box* $B \subseteq \mathbb{R}^k$, *if there is a trajectory in* B *from a point* $\boldsymbol{x} = (x_1, \dots, x_k)^T \in B$ *to a point* $\boldsymbol{y} = (y_1, \dots, y_k)^T \in B$ *on which* $\dot{\boldsymbol{x}} = A\boldsymbol{x}$ *holds, then*

$$\exists t \in \mathbb{R}_{\geq 0} [eigen^*_{A,B}(t, \boldsymbol{x}, \boldsymbol{y})], \tag{1}$$

where $eigen^*_{A,B}(t, \boldsymbol{x}, \boldsymbol{y})$ *denotes*

$$\Big[\bigwedge_{\lambda\in\Lambda_1}\Big[\bigwedge_{c\in c(1,\lambda)} c^T\boldsymbol{y} = c^T\boldsymbol{x}e^{\lambda t}\Big]\Big] \wedge \Big[\bigwedge_{(a,b)\in\Lambda_2} \exists\dot{\boldsymbol{x}}\in\mathbb{R}^k$$

$$\Big[\dot{\boldsymbol{x}} = A\boldsymbol{x} \wedge \bigwedge_{c\in c(2,(a,b))} c^T\boldsymbol{y} = e^{at}c^T\left(\boldsymbol{x}\cos(bt) + \frac{\dot{\boldsymbol{x}} - \boldsymbol{x}a}{b}\sin(bt)\right)\Big]\Big]$$

This expression is a formula in the first-order predicate language over the real numbers—it does *not* contain higher-order expressions such as derivatives ($\dot{\boldsymbol{x}}$ does not denote the derivative of $\boldsymbol{x}$ but simply a new variable). However, the corresponding restriction to the first-order theory of the reals is still undecidable, since we have to deal with function symbols like sin. However, there are over-approximating constraint solvers, that can be used (see Section 4).

We denote the above Constraint 1 by $eigen_{A,B}(\boldsymbol{x}, \boldsymbol{y})$. Note that if A has k different real eigenvalues and $B = \mathbb{R}^k$, then this constraint describes the exact solution of the differential equation.

Now we will describe a constraint describing the reachability not only for linear differential equations, but for much more general descriptions of continuous evolution. We assume that the continuous dynamics is defined by a differential constraint $D(\boldsymbol{x}, \dot{\boldsymbol{x}})$ (or short, D) which can be an arbitrary first-order formula in the theory of the reals over the tuples of variables $\boldsymbol{x}$ and $\dot{\boldsymbol{x}}$. This includes explicit and implicit differential equations and inequalities, and even differential-algebraic equations and inequalities.

Earlier [16] we used a constraint $flow_{D,B}(\boldsymbol{x}, \boldsymbol{y})$ describing the reachability in boxes as follows:

Lemma 2. *For a differential constraint $D(\boldsymbol{x}, \dot{\boldsymbol{x}})$ and a box $B \subseteq \mathbb{R}^k$, if there is a trajectory in B from a point $\boldsymbol{x} = (x_1, \ldots, x_k)^T \in B$ to a point $\boldsymbol{y} = (y_1, \ldots, y_k)^T \in B$ such that for every point $\boldsymbol{u}$ on the trajectory and its derivative $\dot{\boldsymbol{u}}$, the pair $(\boldsymbol{u}, \dot{\boldsymbol{u}})$ satisfies $D(\boldsymbol{x}, \dot{\boldsymbol{x}})$, then*

$$\bigwedge_{1\leq m<n\leq k} \exists a_1, \ldots, a_k, \dot{a}_1, \ldots, \dot{a}_k[(a_1, \ldots, a_k) \in B \wedge$$

$$D((a_1, \ldots, a_k), (\dot{a}_1, \ldots, \dot{a}_k)) \wedge \dot{a}_n \cdot (y_m - x_m) = \dot{a}_m \cdot (y_n - x_n)]$$

Observe that whenever a given pair of points $(\boldsymbol{x}, \boldsymbol{y})$ fulfills the above constraint $flow_{D,B}(\boldsymbol{x}, \boldsymbol{y})$—indicating that there is a possible flow from $\boldsymbol{x}$ to $\boldsymbol{y}$—then also the flipped pair $(\boldsymbol{y}, \boldsymbol{x})$ fulfills the constraint. That is, the constraint does not distinguish time flowing forward, and time flowing backward. In order to avoid this loss of information we use the mean value theorem to formulate the following constraint:

Lemma 3. *For a differential constraint $D(\boldsymbol{x}, \dot{\boldsymbol{x}})$ and a box $B \subseteq \mathbb{R}^k$, if there is a trajectory in B from a point $\boldsymbol{x} = (x_1, \ldots, x_k)^T \in B$ to a point $\boldsymbol{y} =$*

$(y_1, \ldots, y_k)^T \in B$ *such that for every point* $\boldsymbol{u}$ *on the trajectory and its derivative* $\dot{\boldsymbol{u}}$*, the pair* $(\boldsymbol{u}, \dot{\boldsymbol{u}})$ *satisfies* $D(\boldsymbol{x}, \dot{\boldsymbol{x}})$*, then*

$$\exists t \in \mathbb{R}_{\geq 0}[flow^*_{D,B}(t, \boldsymbol{x}, \boldsymbol{y})], \tag{2}$$

where $flow^*_{D,B}(t, \boldsymbol{x}, \boldsymbol{y})$ *denotes*

$$\bigwedge_{1 \leq i \leq k} \exists a_1, \ldots, a_k, \dot{a}_1, \ldots, \dot{a}_k[(a_1, \ldots, a_k) \in B \\ \wedge D((a_1, \ldots, a_k), (\dot{a}_1, \ldots, \dot{a}_k)) \wedge y_i = x_i + \dot{a}_i \cdot t]$$

We denote the above Constraint 2 by $flow'_{D,B}(\boldsymbol{x}, \boldsymbol{y})$. Although both the constraint $flow_{D,B}(\boldsymbol{x}, \boldsymbol{y})$ and the constraint $flow'_{D,B}(\boldsymbol{x}, \boldsymbol{y})$ are quite simple, it is not at all clear, whether $flow'_{D,B}(\boldsymbol{x}, \boldsymbol{y})$ really allows to derive tighter reach set information than $flow_{D,B}(\boldsymbol{x}, \boldsymbol{y})$.

4 Solving the Constraints

We solve the constraints using our constraint solver RSolver [12], which implements interval constraint propagation techniques [5, 3, 9, 13]. These techniques can, for a given constraint and intervals for all its variables, contract these intervals to smaller ones, without losing any solutions. We illustrate the idea on an example: Given the constraint $x^2 - 1 \leq 0$, and the interval $[-2, 2]$ for x, the method first decomposes this constraint into a conjunction of so-called primitive constraints, arriving at $x^2 = t_0 \wedge t_0 - 1 = t_1 \wedge t_1 \leq 0$. Here t_0 and t_1 are new, auxiliary variables. Then it takes the interval $[-\infty, +\infty]$ for all auxiliary variables and tries to contract all intervals wrt. the primitive constraints: using $x^2 = t_0$ we can contract the interval $[-\infty, \infty]$ for t_0 to $[0, 4]$, using $t_0 - 1 = t_1$ contract $[-\infty, \infty]$ for t_1 to $[-1, 3]$, using $t_1 \leq 0$ contract $[-1, 3]$ for t_1 to $[-1, 0]$, using $t_0 - 1 = t_1$ contract $[0, 4]$ for t_0 to $[0, 1]$, and using $x^2 = t_0$, contract $[-2, 2]$ for x to $[-1, 1]$. This process continues until a fixpoint is reached, which will always happen eventually, due to the finiteness of floating point numbers. We call the resulting algorithm a *pruning function* and, given a constraint ϕ and a box B, we denote the result of applying this function to ϕ and B by $Prune(\phi, B)$. In computer implementations the resulting intervals will enclose the solution sets of the primitive constraints up to rounding to the next floating point number. For the theoretical analysis in this paper we will ignore this rounding, and assume that these intervals are the tightest possible enclosures using real-number endpoints.

Definition 1. *Given a constraint* ϕ*, pruning is* optimal *for* ϕ *iff for all boxes* B*,* $Prune(\phi, B)$ *is the smallest box containing all solutions of* ϕ *in* B*.*

Although the pruning function will contract optimally for primitive constraints, this will in general not be the case for more complex constraints. However, due to a classical result of interval arithmetic, we have (ignoring floating-point rounding):

Property 1. For every constraint ϕ, that contains every variable just once, pruning is optimal.

There are special techniques for handling disjunctions and quantifiers [13]. Moreover, there are various optimizations and extensions of the techniques discussed above. Most of them spend additional time to deal with the problem that pruning is in general not optimal if some variables occur more than once. Since our constraints usually only have few occurrences of the same variables, and since in our abstraction refinement approach (see Section 2) we will do thousands of prunings, it does not seem promising to use such optimizations here.

5 Theoretical Evaluation of the Constraints

In this section we will do a theoretical comparison of the constraints $flow_{D,B}(\boldsymbol{x},\boldsymbol{y})$, $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$ and $eigen_{A,B}(\boldsymbol{x},\boldsymbol{y})$. Moreover, based on the gained insight, we will introduce a new constraint that combines their advantages.

We start with comparing $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$ with $flow_{D,B}(\boldsymbol{x},\boldsymbol{y})$. Let us first discuss the size of the produced constraints. For k dimensions, $flow_{D,B}(\boldsymbol{x},\boldsymbol{y})$ has $k(k-1)/2$ conjuncts, whereas $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$ has just k. So, for dimensions larger than 2, $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$ is smaller, and its size increases only linearly instead of quadratically. Hence pruning will take less time on $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$, especially for high dimensions.

Let us now compare the effectiveness of the two constraints. For the one-dimensional case, $flow_{D,B}(\boldsymbol{x},\boldsymbol{y})$ reduces to a conjunction with zero conjuncts, that is, to a constraint that is trivially true. So in that case, $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$ is definitely better. For higher dimensions, the relationship between the two constraints is more complicated. Therefore, we will first study the relationship between $flow_{D,B}(\boldsymbol{x},\boldsymbol{y})$ and $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$ themselves, and then between the result of applying the pruning function to them.

Here we will use the following notation: Given two constraints $\phi_{S,B}(\boldsymbol{x},\boldsymbol{y})$ and $\psi_{S,B}(\boldsymbol{x},\boldsymbol{y})$, where S is a differential constraint D or a matrix A, and B is a box, we will write $\phi_{S,B} \preceq \psi_{S,B}$ ($\phi_{S,B} \equiv \psi_{S,B}$) iff for all S and all B, the solution set of $\phi_{S,B}$ in $B \times B$ is a subset of (equal to) the solution set of $\psi_{S,B}$ in $B \times B$. Analogously, we will write $\phi_{S,B} \preceq_P \psi_{S,B}$ ($\phi_{S,B} \equiv_P \psi_{S,B}$) iff for all S, all B and all B_0, where B_0 is a sub-box of B, $Prune(\phi_{S,B}(\boldsymbol{x},\boldsymbol{y}), B_0 \times B)$ is a subset of (equal to) $Prune(\psi_{S,B}(\boldsymbol{x},\boldsymbol{y}), B_0 \times B)$. Note that here we restrict B_0 to be a subset of B because we use the constraints always in such a context.

There is no clearcut relationship between $flow_{D,B}(\boldsymbol{x},\boldsymbol{y})$ and $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$:

Property 2. Neither $flow_{D,B} \preceq flow'_{D,B}$ nor $flow'_{D,B} \preceq flow_{D,B}$.

Proof. For showing the first part, we use a differential constraint $\dot{x}_1 = 0 \wedge \dot{x}_2 = 0$ and a box $B = [0,2] \times [0,2]$. Obviously, $((1,1),(2,2)) \in \{(\boldsymbol{x},\boldsymbol{y}) : flow_{D,B}(\boldsymbol{x},\boldsymbol{y})\}$, but $((1,1),(2,2)) \notin \{(\boldsymbol{x},\boldsymbol{y}) : flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})\}$. The reason lies in the fact that the derivatives are zero for this example, and in such a case, the equality in $flow_{D,B}(\boldsymbol{x},\boldsymbol{y})$ reduces to the trivial equality $0 = 0$ that is true for all $\boldsymbol{x},\boldsymbol{y}$.

For showing the second part, we use a differential constraint $\dot{x}_1 = x_1 + x_2 + 1 \wedge \dot{x}_2 = x_1 + x_2 + 1$ and a box $B = [0,2] \times [0,2]$. Obviously, $((0,0),(1,\frac{1}{5})) \in \{(\boldsymbol{x},\boldsymbol{y}) : flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})\}$, but $((0,0),(1,\frac{1}{5})) \notin \{(\boldsymbol{x},\boldsymbol{y}) : flow_{D,B}(\boldsymbol{x},\boldsymbol{y})\}$. This is because we need $\dot{a}_m$ and $\dot{a}_n$ to be derivatives of a_m and a_n at the same point in $flow_{D,B}(\boldsymbol{x},\boldsymbol{y})$, but in $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$, $\dot{a}_i$, can be the derivative of a different point for every i. □

However, in our method, instead of computing exact solutions to these constraints, we only use over-approximations computed by the pruning function, and a tighter constraint does not necessarily give rise to a tighter pruning result.

So let us compare these over-approximations. We want to discuss the relation between Prune$(flow_{D,B}(\boldsymbol{x},\boldsymbol{y}), B_0 \times B)$ and Prune$(flow'_{D,B}(\boldsymbol{x},\boldsymbol{y}), B_0 \times B)$.

Theorem 2. $flow'_{D,B} \preceq_P flow_{D,B}$.

Proof. Given a set of variables V and a box B for which each component corresponds to a certain variable, we denote by $\pi_V(B)$ the projection of B to the components corresponding to the variables in V.

We transform the constraints into a conjunction without existential quantifiers as follows: Rename the variables $a_1, \dots, a_k$ and $\dot{a}_1, \dots, \dot{a}_k$ to a different tuple of variables in each branch, and then drop all corresponding existential quantifiers. As a result, in addition to $\boldsymbol{x}$ and $\boldsymbol{y}$, $flow'_{D,B}$ has $2k^2+1$ free variables and $flow_{D,B}$ has $k^2(k-1)$ free variables. Obviously, we have to prove

$$\pi_{\boldsymbol{x},\boldsymbol{y}}(Prune(flow'_{D,B}(\boldsymbol{x},\boldsymbol{y}), B_0 \times B \times \mathbb{R}_{\geq 0} \times \mathbb{R}^{2k^2})) \subseteq \pi_{\boldsymbol{x},\boldsymbol{y}}(Prune(flow_{D,B}(\boldsymbol{x},\boldsymbol{y}), B_0 \times B \times \mathbb{R}^{k^2(k-1)})).$$

Let $\phi'_B \doteq \bigwedge_{1\leq i\leq k} y_i = x_i + \dot{a}_i t$, and $\phi_B \doteq \bigwedge_{1\leq m<n\leq k} \dot{a}_n(y_m - x_m) = \dot{a}_m(y_n - x_n)$. Let $\dot{A}$ be an arbitrary, but fixed, k-dimensional box. Let P'_s be the exact solution set of ϕ'_B in $B_0 \times B \times \mathbb{R}_{\geq 0} \times \dot{A}$ and $P' = Prune(\phi'_B, B_0 \times B \times \mathbb{R}_{\geq 0} \times \dot{A})$. Also, let P_s be the exact solution set of ϕ_B in $B_0 \times B \times \dot{A}$ and $P = Prune(\phi_B, B_0 \times B \times \dot{A})$. Since $(a_1, \dots, a_k) \in B \wedge D((a_1, \dots, a_k), (\dot{a}_1, \dots, \dot{a}_k))$ is shared by both $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$ and $flow_{D,B}(\boldsymbol{x},\boldsymbol{y})$, it suffices to prove that $\pi_{\boldsymbol{x},\boldsymbol{y}}(P') \subseteq \pi_{\boldsymbol{x},\boldsymbol{y}}(P)$. We will proceed by first proving that $\pi_{\boldsymbol{x},\boldsymbol{y}}(P'_s) \subseteq \pi_{\boldsymbol{x},\boldsymbol{y}}(P_s)$ and then lifting this to $\pi_{\boldsymbol{x},\boldsymbol{y}}(P') \subseteq \pi_{\boldsymbol{x},\boldsymbol{y}}(P)$.

So let $(\boldsymbol{x},\boldsymbol{y}) \in \pi_{\boldsymbol{x},\boldsymbol{y}}(P'_s)$ be arbitrary, but fixed. We will prove that $(\boldsymbol{x},\boldsymbol{y})$ is also in $\pi_{\boldsymbol{x},\boldsymbol{y}}(P_s)$. Since $(\boldsymbol{x},\boldsymbol{y})$ is in the projection of P'_s we know that there are $(t, \dot{a}_1, \dots, \dot{a}_k)$ such that $(\boldsymbol{x},\boldsymbol{y}, t, \dot{a}_1, \dots, \dot{a}_k)$ satisfies the constraint ϕ'_B. Choose a t^*, $\dot{a}^*_1, \dots, \dot{a}^*_k$ with that property.

For proving that $(\boldsymbol{x},\boldsymbol{y}) \in \pi_{\boldsymbol{x},\boldsymbol{y}}(P_s)$, it suffices to prove that $(\boldsymbol{x},\boldsymbol{y}, \dot{a}^*_1, \dots, \dot{a}^*_k)$ satisfies $\bigwedge_{1\leq m<n\leq k} \dot{a}^*_n(y_m - x_m) = \dot{a}^*_m(y_n - x_n)$. Letting m, n be arbitrary, but fixed, such that $1 \leq m < n \leq k$, we prove that $(\boldsymbol{x},\boldsymbol{y}, \dot{a}^*_1, \dots, \dot{a}^*_k)$ is in the solution set of the corresponding conjunct. Here we have three cases:

- $\dot{a}^*_m \neq 0, \dot{a}^*_n \neq 0$: Then $\frac{y_m - x_m}{\dot{a}^*_m} = t^*$ and $\frac{y_n - x_n}{\dot{a}^*_n} = t^*$, so $\frac{y_m - x_m}{\dot{a}^*_m} = \frac{y_n - x_n}{\dot{a}^*_n}$, and hence $\dot{a}^*_n(y_m - x_m) = \dot{a}^*_m(y_n - x_n)$

- $\dot{a}_m^* = 0$: then $x_m = y_m$, and both sides of the equality $\dot{a}_n^*(y_m - x_m) = \dot{a}_m^*(y_n - x_n)$ are zero,
- $\dot{a}_n^* = 0$, analogous to previous case.

Hence $\pi_{\boldsymbol{x},\boldsymbol{y}}(P_s') \subseteq \pi_{\boldsymbol{x},\boldsymbol{y}}(P_s)$. For lifting this to the projected results of pruning, we first observe that each conjunct of ϕ_B' contains each variable just once. So, due to Property 1, pruning is optimal for each conjunct. Due to the fact that the conjuncts only share a single variable t also pruning of ϕ_B' is optimal [3], and since projection commutes with the smallest-box relation, $\pi_{\boldsymbol{x},\boldsymbol{y}}(P')$ is the smallest box containing $\pi_{\boldsymbol{x},\boldsymbol{y}}(P_s')$. Moreover, since projection commutes with the subset relation, not only $P_s \subseteq P$, but also $\pi_{\boldsymbol{x},\boldsymbol{y}}(P_s) \subseteq \pi_{\boldsymbol{x},\boldsymbol{y}}(P)$, and by transitivity $\pi_{\boldsymbol{x},\boldsymbol{y}}(P_s') \subseteq \pi_{\boldsymbol{x},\boldsymbol{y}}(P)$. Now, the following property implies the theorem: If B is the smallest box containing a set S and $S \subseteq S'$, then for every box B' with $S' \subseteq B'$, $B \subseteq B'$. $\square$

So, we can prune at least as tightly using $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$ as using $flow_{D,B}(\boldsymbol{x},\boldsymbol{y})$. In fact, we can prune strictly tighter!

Property 3. Not $flow'_{D,B} \equiv_P flow_{D,B}$.

Proof. Take a differential constraint $\dot{x}_1 = 1 \wedge \dot{x}_2 = 1$ and a box $B = [0,2] \times [0,2]$. Let $B_0 = \{(1,1)\}$, then $\text{Prune}(flow'_{D,B}(\boldsymbol{x},\boldsymbol{y}), B_0 \times B) = B_0 \times [1,2] \times [1,2]$ and $\text{Prune}(flow_{D,B}(\boldsymbol{x},\boldsymbol{y}), B_0 \times B) = B_0 \times B$. $\square$

To sum up, the theoretical evaluation shows that although none of the two constraints always has a smaller solution set than the other, $flow'_{D,B}(\boldsymbol{x},\boldsymbol{y})$ has a definite advantage in size and in pruning power. Hence we only use the latter from now on.

Next, we will compare $flow'_{\dot{\boldsymbol{x}}=A\boldsymbol{x},B}(\boldsymbol{x},\boldsymbol{y})$ (or short: $flow'_{A,B}(\boldsymbol{x},\boldsymbol{y})$) with the constraint $eigen_{A,B}(\boldsymbol{x},\boldsymbol{y})$ for linear differential equations $\dot{\boldsymbol{x}} = A\boldsymbol{x}$.

If A has k different real eigenvalues and $B = \mathbb{R}^k$, $eigen_{A,B}(\boldsymbol{x},\boldsymbol{y})$ describes the exact solutions of the differential equations. But, $flow'_{A,B}(\boldsymbol{x},\boldsymbol{y})$ employs the first-order Taylor expansion to over-approximate the exact solutions. Thus, $eigen_{A,B} \preceq flow'_{A,B}$? No! Only in cases where all trajectories leave the box, and do not enter it again. Otherwise, $eigen_{A,B}(\boldsymbol{x},\boldsymbol{y})$ also includes the part of the trajectory that enters the box again, but $flow'_{A,B}(x,\boldsymbol{y})$ does not necessarily.

Now we compare the two constraints wrt. pruning. For linear differential equations, pruning is optimal for $flow'_{A,B}$. This follows from an analysis of the proof of Theorem 2, and the fact that the differential equation constrains each derivative $\dot{x}$ using a constraint $\dot{\boldsymbol{x}} = A\boldsymbol{x}$, where each equation of this constraint contains only one component of $\dot{\boldsymbol{x}}$, and contains each component of $\boldsymbol{x}$ only once. However, this is in general not the case for $eigen_{A,B}$ due to multiple occurrences of variables.

There is no clearcut relationship between $Prune(flow'_{A,B}(\boldsymbol{x},\boldsymbol{y}), B_0 \times B)$ and $Prune(eigen_{A,B}(\boldsymbol{x},\boldsymbol{y}), B_0 \times B)$:

Property 4. Neither $flow'_{A,B} \preceq_P eigen_{A,B}$ nor $eigen_{A,B} \preceq_P flow'_{A,B}$.

Proof. This can be directly proven using only one example with a differential equation $(\dot{x}_1, \dot{x}_2) = (-x_1 - x_2, x_1 - x_2)$ and a box $B = [0,4] \times [0,4]$. If we set $B_0 = [2.5, 3] \times [0,0]$, then $Prune(flow'_{A,B}, B_0 \times B) = B_0 \times [0,3] \times [0,4]$ and $Prune(eigen_{A,B}, B_0 \times B) = B_0 \times [0, 3.5] \times [0,3]$. □

Moreover, there are even some cases, where pruning $flow'_{A,B}$ returns a strict subset of pruning $eigen_{A,B}$ and vice versa: The former happens for a differential equation $(\dot{x}_1, \dot{x}_2) = (x_1 - x_2, x_1 + x_2)$ and a box $B = [0,2] \times [0,4]$. If we set $B_0 = [2,2] \times [2,4]$, then $Prune(flow'_{A,B}, B_0 \times B) = B_0 \times [0,2] \times [2,4]$ and $Prune(eigen_{A,B}, B_0 \times B) = B_0 \times B$. This is because the left-hand side of $eigen_{A,B}$, $c^T\boldsymbol{y}$, evaluates to zero on some element in B. Hence every solution of $c^T(\boldsymbol{x}\cos(bt) + \frac{\dot{\boldsymbol{x}} - \boldsymbol{x}a}{b}\sin(bt)) = 0$ fulfills the constraint. There is such a solution, and since sin and cos are periodic, the solution set is not bounded for t, and interval $[0, +\infty]$ for t will not be pruned. Since a is positive, the interval derived for the term e^{at} will also stay unbounded, and no intervals will be pruned. But, $t \in [0,1]$ in $flow'_{A,B}$, which does provide some pruning.

The latter happens for a differential equation $(\dot{x}_1, \dot{x}_2) = (x_1, x_2)$ and a box $B = [0,2] \times [0,2]$. If we set $B_0 = \{(0,0)\}$, $Prune(flow'_{A,B}, B_0 \times B) = B_0 \times B$ and $Prune(eigen_{A,B}, B_0 \times B) = B_0 \times B_0$. This is because $eigen_{A,B}$ here describes the exact solution starting from the initial point. But, since $\dot{x}_1$ and $\dot{x}_2$ can be zero, pruning $flow'_{A,B}$ results in $t \in [0, \infty]$. Thus, also the intervals for y_1 and y_2 cannot be pruned.

Since there is no clearcut relationship between $Prune(flow'_{A,B}, B_0 \times B)$ and $Prune(eigen_{A,B}, B_0 \times B)$, we strengthen both constraints by combining them. Thus, by sharing the same time variable we allow timing information to be propagated between them as follows:

Lemma 4. *For a linear differential equation $\dot{\boldsymbol{x}} = A\boldsymbol{x}$ and a box B, if there is a trajectory in B from a point $\boldsymbol{x} = (x_1, \ldots, x_k)^T \in B$ to a point $\boldsymbol{y} = (y_1, \ldots, y_k)^T \in B$ on which $\dot{\boldsymbol{x}} = A\boldsymbol{x}$ holds, then*

$$\exists t \in \mathbb{R}_{\geq 0}[flow^*_{A,B}(t, \boldsymbol{x}, \boldsymbol{y}) \wedge eigen^*_{A,B}(t, \boldsymbol{x}, \boldsymbol{y})] \tag{3}$$

We denote the above new Constraint 3 by $comb_{A,B}(\boldsymbol{x}, \boldsymbol{y})$. Clearly, this constraint implies $flow'_{A,B}(\boldsymbol{x}, \boldsymbol{y})$, and also implies $eigen_{A,B}(\boldsymbol{x}, \boldsymbol{y})$. That is, $comb_{A,B} \preceq flow'_{A,B}$ and $comb_{A,B} \preceq eigen_{A,B}$. Moreover, we have:

Theorem 3. *$comb_{A,B} \preceq_P flow'_{A,B}$ and $comb_{A,B} \preceq_P eigen_{A,B}$.*

So, the combination constraint is at least as good as $flow'_{A,B}(\boldsymbol{x}, \boldsymbol{y})$ and $eigen_{A,B}(\boldsymbol{x}, \boldsymbol{y})$. But, in fact, it is better!

Property 5. Neither $comb_{A,B} \equiv_P flow'_{A,B}$ nor $comb_{A,B} \equiv_P eigen_{A,B}$.

Proof. This can be seen on an example with a differential equation $(\dot{x}_1, \dot{x}_2) = (-x_1 - x_2, x_1 - x_2)$ and a box $B = [0,4] \times [0,4]$. If we set $B_0 = [2.5, 3] \times [0,0]$, then $Prune(flow'_{A,B}, B_0 \times B) = B_0 \times [0,3] \times [0,4]$, $Prune(eigen_{A,B}, B_0 \times B) = B_0 \times [0, 3.5] \times [0,3]$ and $Prune(comb_{A,B}, B_0 \times B) = B_0 \times [0,3] \times [0,3]$. □

However, the combination constraint is bigger than both $flow'_{A,B}(x, \boldsymbol{y})$ and $eigen_B(\boldsymbol{x}, \boldsymbol{y})$. Thus, pruning will take more time on it.

6 Empirical Evaluation

In this section we evaluate the constraints empirically by using them in the verification method introduced in Section 2. That is, we replace all occurrences of $\mathit{Reachable}_B(s, \boldsymbol{x}, \boldsymbol{y})$ introduced in Theorem 1 by $flow_{D,B}$ (or, $flow_{A,B}$), $flow'_{D,B}$ (or, $flow'_{A,B}$), $eigen_{A,B}$ and $comb_{A,B}$, respectively. We illustrate the behavior of our implementation on a few benchmark problems. Note that in the literature on the verification of hybrid systems the habit prevails to test new methods only on 2-3, or even less examples. We do not follow this tradition and do more extensive benchmarking. We will publish the corresponding HSolver input files on its web-page [15].

Note that we use the following splitting strategy here: In each mode, we choose the box with the biggest side-length, and then bisect each choice along the variable along which the box has not been split the longest time (i.e., we use a round-robin strategy to choose the variable). The computations were performed on a Pentium IV, 2.60GHz with 1 GB RAM, and they were canceled when computation did not terminate before 5 hours of computation time.

We used the following benchmark problems for comparing $flow_{D,B}$ and $flow'_{D,B}$ with computation results shown in Table 1:

Example 1:
Flow: $(\dot{x}_1, \dot{x}_2) = (-x_1 - x_2, x_1 - x_2)$, empty jump relation
Init: $2.5 \leq x_1 \leq 3 \wedge x_2 = 0$, Unsafe: $x_1 > 3 \vee x_2 > 3$
The state space: $[0, 4] \times [0, 4]$

Example 2:
Flow: $(\dot{x}_1, \dot{x}_2) = (x_1 - x_2, x_1 + x_2)$, empty jump relation
Init: $2.5 \leq x_1 \leq 3 \wedge x_2 = 0$, Unsafe: $x_1 \leq 2$
The state space: $[0, 4] \times [0, 4]$

Example 3: The flow constraints are constructed by setting all the parameters in the two tanks problem [18] to 1.
Flow: $\big(s = 1 \rightarrow \binom{\dot{x}_1}{\dot{x}_2} = \binom{1-\sqrt{x_1}}{\sqrt{x_1}-\sqrt{x_2}}\big) \wedge \big(s = 2 \rightarrow \binom{\dot{x}_1}{\dot{x}_2} = \binom{1-\sqrt{x_1-x_2+1}}{\sqrt{x_1-x_2+1}-\sqrt{x_2}}\big)$
Jump: $(s = 1 \wedge 0.99 \leq x_2 \leq 1) \rightarrow (s' = 2 \wedge x'_1 = x_1 \wedge x'_2 = 1)$
Init: $s = 1 \wedge (x_1 - 5.5)^2 + (x_2 - 0.25)^2 \leq 0.0625$
Unsafe: $\big(s = 1 \wedge (x_1 - 4.25)^2 + (x_2 - 0.25)^2 < 0.0625\big)$
The state space: $(1, [4, 6] \times [0, 1]) \cup (2, [4, 6] \times [1, 2])$

Example 4: This is a predator-prey example.
Flow: $\big(s = 1 \rightarrow \binom{\dot{x}_1}{\dot{x}_2} = \binom{-x_1+x_1x_2}{x_2-x_1x_2}\big) \wedge \big(s = 2 \rightarrow \binom{\dot{x}_1}{\dot{x}_2} = \binom{-x_1+x_1x_2}{x_2-x_1x_2}\big)$
Jump: $\big((s = 1 \wedge 0.875 \leq x_2 \leq 0.9) \rightarrow (s' = 2 \wedge (x'_1 - 1.2)^2 + (x'_2 - 1.8)^2 \leq 0.01\big)$
$\vee\big((s = 2 \wedge 1.1 \leq x_2 \leq 1.125) \rightarrow (s' = 1 \wedge (x'_1 - 0.7)^2 + (x'_2 - 0.7)^2 \leq 0.01)\big)$
Init: $s = 1 \wedge (x_1 - 0.8)^2 + (x_2 - 0.2)^2 \leq 0.01$
Unsafe: $\big(s = 1 \wedge x_1 > 0.8 \wedge x_2 > 0.8 \wedge x_1 <= 0.9 \wedge x_2 \leq 0.9\big)$
State space: $(1, [0.1, 0.9] \times [0.1, 0.9]) \cup (2, [1.1, 1.9] \times [1.1, 1.9])$

Example 5: This is a simple example with a clock variable.
Flow: $(\dot{x}, \dot{y}, \dot{t}) = (-5.5y + y^2, 6x - x^2, 1)$, empty jump relation
Init: $4 \leq x \leq 4.5 \wedge y = 1 \wedge t = 0$

Unsafe: $(1 \le x < 2 \wedge 2 < y < 3 \wedge 2 \le t \le 4)$
The state space: $[1, 5] \times [1, 5] \times [0, 4]$
Example 6: A three-dimensional and nonlinear example about a simple controller that steers a car along a straight road [4]. The three continuous variables are the position x, the heading angle γ and the internal timer c. Since we cannot prove the safety property described in the original paper, in this paper the unsafe space is reset to be $x \le -4$.
Example 7: A linear collision avoidance example from a part of the car convoi control from a paper by A. Puri and P. Varaiya [11]. Let gap, v_r, v_l and a_r respectively represent the distance between the two cars ($d_{i-1} - d_i$ in the original paper), the velocity of the rear car ($\dot{d}_i$), the velocity of the leading car ($\dot{d}_{i-1}$) and the acceleration of the rear car ($\ddot{d}_i$). By using these variables and restricting v_l by $-2 \le \dot{v}_l \le -0.5$ we transformed the original higher-order differential equation into a four-dimensional differential (in)equation of order one.

We set the state space to $[0, 4] \times [0, 2] \times [0, 2] \times [-2, -0.5]$, and we want to verify that $gap > 0$ when starting from $gap = 1, v_r = 2, v_l = 2$ and $a_r = -0.5$.
Example 8: A four-dimensional and nonlinear example about a mixing-tank-system from a paper by O. Stursberg, S. Kowalewski and S. Engell [17]. In the original paper, the system is simplified to a two-dimensional system. In this paper, we keep the differential equations $(\dot{V}_1, \dot{V}_2) = (0.008, 0.015)$ in the flow constraint, where V_1 and V_2 are two inlet streams. Then, initially, $V_1(0) = 1, V_2(0) = 1$, and $(h(0), c(0)) \in [1.32, 1.5] \times [1.2, 1.32]$, where h is liquid height and c is concentration. We want to verify that the state $\{(V_1, V_2, h, c) : h \in [1.1, 1.3] \wedge c \in [1.68, 1.80]\}$ is unreachable.
Example 9: A two-dimensional and nonlinear example about a tunnel-diode oscillator circuit [6]. It models the voltage drop V and the current I. The original problem was to prove that all trajectories eventually reach a certain set and stay there. We transformed it to a reachability problem, using the state space $[-0.1, 0.6] \times [-0.002, 0.002]$ and the unsafety constraint $V < -0.04 \vee V > 0.54 \vee I < -0.0015 \vee I > 0.00175$.
Example 10: A linear, three-dimensional model of a mutant of V. *fischeri* [2]. Let x_1, x_2 and x_3 respectively represent the protein *LuxI*, the autoinducer *Ai* and the complex *Co* described in the original paper. The model has two modes with dynamics in the form $\dot{x} = Ax + b_i$, $i = 1, 2$, where $x = (x_1, x_2, x_3)^T$ and

$$A = \begin{pmatrix} -1/3600 & 0 & 0 \\ 7.5e-5 & -(1/36000 + 7.5e-9) & 1.5e-9 \\ 0 & 0.005 & -1/3600 - 0.01 \end{pmatrix}$$

and $b_1 = (0.00375, 0, 0)^T$ and $b_2 = (3.75375, 0, 0)^T$.

We set the state space to be $[0, 30000] \times [0, 60000] \times [0, 30000]$ and the switches occur when the plane $x_3 = 1000$ is reachable and $x_2 \in [1000, 45000]$. We want to verify that $x_1 \ge 27500 \vee x_2 \ge 50000 \vee x_3 \ge 25000$ cannot be reachable when starting from $[17500, 20000] \times [40000, 45000] \times [5000, 7500]$ in mode 1.

The results show that the new constraint improves the number of pruning steps for all examples except for Example 8, which we will discuss below. As

Table 1. Computation results for $flow_{D,B}$ and $flow'_{D,B}$

Example	$flow_{D,B}$			$flow'_{D,B}$		
	CPU time	Splitting steps	Pruning number	CPU time	Splitting steps	Pruning number
1	0.041s	6	79	0.020s	2	30
2	0.34s	71	1572	0.38s	58	1203
3	0.18s	11	397	0.24s	11	397
4	0.57s	43	2250	0.66s	42	1884
5	2.59s	93	3552	1.69s	81	2929
6	0.35s	1	88	0.14s	0	41
7	187.96s	369	69158	105.55s	367	68675
8	7.68s	54	3138	43.95s	294	39281
9	13.209s	165	12215	3.653s	57	4655
10	1876s	1889	366681	686s	1417	270078

expected, this also decreases the run-time of the method except for 2-dimensional examples, where $flow'_{D,B}$ has more conjuncts than $flow_{D,B}$.

We analyzed the anomaly in the behavior on Example 8 in more detail. After applying the pruning algorithm for the first time to $reachable_{B'}(s', \boldsymbol{z})$, using $flow'_{D,B}$ we can prune the box $[0, 2] \times [0, 2] \times [0.5, 1.5] \times [1.2, 1.8]$ to a new box $[1, 1.53333333333] \times [1, 2] \times [1.22034017148, 1.5] \times [1.2, 1.8]$; but, after we apply the pruning algorithm to $reachable_{B'}(s', \boldsymbol{z})$ using $flow_{D,B}$, we can only prune the box $[0, 2] \times [0, 2] \times [0.5, 1.5] \times [1.2, 1.8]$ to a new box $[0.466666666667, 1.53333333333] \times [0, 2] \times [1.22034017148, 1.5] \times [1.2, 1.8]$. So, in fact, the new method is better at the beginning! However, it seems that this improvement at the beginning turns out to be bad luck later since our method is very sensitive to splitting heuristics, and the improved pruning results in different choices of boxes for splitting during the algorithm. This suggests that a detailed study of splitting heuristics, will be able to significantly improve the method further.

In addition to the linear examples from above (Examples 1, 2 and 10), we used the following benchmarks for comparing $flow'_{A,B}$, $eigen_{A,B}$ and $comb_{A,B}$ with results shown in Table 2:

Example 11: A linear, three-dimensional example.
Flow: $(\dot{x}_1, \dot{x}_2, \dot{x}_3) = (0.80x_2 + 0.6x_3 - 1.8, 0.8x_1 + 0.7x_3 - 15.2, 0.6x_1 + 0.7x_2 - 1.8)$;
Empty jump relation; Init: $19 \leq x_1 \leq 20 \wedge 19 \leq x_2 \leq 20 \wedge 19 \leq x_3 \leq 20$;
Unsafe: $x_1 \leq 21 \wedge x_2 \leq 20 \wedge x_3 \geq 22.5$;
The state space: $[15, 24] \times [15, 24] \times [15, 24]$.

Example 12: A linear collision avoidance example similar to Example 7.

We restrict v_l by $\dot{v}_l = 0$, and reset the state space to be $[0, 10] \times [0, 30] \times [0, 30] \times [-2, 5]$ and the initial set to be $-0.8522v_r - 0.1478v_l - 0.3177a_r + gap > 10$.

The results show that the combination decreases the size of the abstraction and the number of calls to the constraint solver. However, as expected, this will for some cases increase the run-time of the method, due to the bigger size of this constraint. This phenomenon is reflected by Examples 1, 2 and 12. But,

Table 2. Computation results for $flow'_{A,B}$, $eigen_{A,B}$ and $comb_{A,B}$

Example	$flow'_{A,B}$			$eigen_{A,B}$			$comb_{A,B}$		
	CPU time	Splitting steps	Pruning number	CPU time	Splitting steps	Pruning number	CPU time	Splitting steps	Pruning number
1	0.020s	2	30	unknown			0.038s	0	3
2	0.38s	58	1203	unknown			5.362s	58	1203
10	686s	1417	270078	12534s	3279	560082	122s	235	46540
11	unknown			unknown			0.319s	5	61
12	unknown			0.465s	0	3	0.756s	0	3

for hard (and thus realistic) problems (e.g., Example 10), the improvement due to the first phenomenon will always dominate: in such cases the time spent on constraint solving will always be dominated by computations on the abstraction, and hence it is essential to keep the abstraction small.

For some cases, the safety property cannot be verified using $eigen_{A,B}$ in our method. For Examples 1 and 2 this can be explained using an observation already discussed in Section 5: the eigenvalues are complex with non-zero imaginary parts, and in such a case, since t occurs several times in the term $c^T(\boldsymbol{x}\cos(bt) + \frac{(\dot{\boldsymbol{x}}-a\boldsymbol{x})}{b}\sin(bt))$, we will get an over-approximating interval for this term.

On Examples 10 and 11 it can be seen nicely that the combined constraint can be stronger than either $flow'_{A,B}$ or $eigen_{A,B}$ in isolation.

Note that we did not use Examples 11 and 12 in Table 1 because their safety properties cannot be verified using either $flow_{A,B}$ or $flow'_{A,B}$.

7 Conclusion

We have provided a detailed study of two types of constraints in the verification of hybrid systems. The overall approach, to formulate reach set computation as a constraint solving problem, and to apply an efficiently over-approximating constraint solver to it, can be extended to various new types of constraints. Specifically we will study the use of higher order Taylor approximations instead of the constraint based on the mean value theorem. Our software is publically available [15], and includes an interface that allows the incorporation of and experimentation with new, user-defined constraints. Based on the gained experience and user feedback, we will optimize the constraint solver especially for the most useful ones.

References

1. H. Anai and V. Weispfenning. Reach set computation using real quantifier elimination. In *Proceedings of 4th International Workshop on Hybrid Systems: Computation and Control (HSCC2001)*, volume 2034 of *LNCS*, pages 63–76. Springer, 2001.

2. C. Belta, J. Schug, T. Dang, V. Kumar, G. Pappas, H. Rubin, and P. Dunlap. Stability and reachability analysis of a hybrid model of luminescence in the marine bacterium vibrio fisheri. In *CDC'01 - Conference on Decision and Control*. Florida, USA, 2001.
3. F. Benhamou and W. J. Older. Applying interval arithmetic to real, integer and Boolean constraints. *Journal of Logic Programming*, 32(1):1–24, 1997.
4. E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement for symbolic model checking. *Journal of the ACM*, 50(5):752–794, 2003.
5. E. Davis. Constraint propagation with interval labels. *Artif. Intell.*, 32(3):281–331, 1987.
6. G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In Morari and Thiele [10].
7. T. Hickey and D. Wittenberg. Rigorous modeling of hybrid systems using interval arithmetic constraints. In R. Alur and G. J. Pappas, editors, *Hybrid Systems: Computation and Control*, number 2993 in LNCS. Springer, 2004.
8. T. J. Hickey. Analytic constraint solving and interval arithmetic. In *Proceedings of the 27th Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 338–351. ACM Press, 2000.
9. T. J. Hickey. Metalevel interval arithmetic and verifiable constraint solving. *Journal of Functional and Logic Programming*, 2001(7), October 2001.
10. M. Morari and L. Thiele, editors. *Hybrid Systems: Computation and Control*, volume 3414 of *LNCS*. Springer, 2005.
11. A. Puri and P. Varaiya. Driving safely in smart cars. In *Proc. of the 1995 American Control Conference*, pages 3597–3599, 1995.
12. S. Ratschan. RSolver. `http://rsolver.sourceforge.net`, 2004. Software package.
13. S. Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*, 2005. To appear.
14. S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. *ACM Journal in Embedded Computing Systems*. to appear.
15. S. Ratschan and Z. She. HSolver. `http://hsolver.sourceforge.net`, 2004. Software package.
16. S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In Morari and Thiele [10].
17. O. Stursberg, S. Kowalewski, and S. Engell. On the generation of timed discrete approximations for continuous systems. *Mathematical and Computer Models of Dynamical Systems*, 6:51–70, 2000.
18. O. Stursberg, S. Kowalewski, I. Hoffmann, and J. Preußig. Comparing timed and hybrid automata as approximations of continuous systems. In P. J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems*, number 1273 in LNCS, pages 361–377. Springer, 1997.
19. A. Tiwari. Approximate reachability for linear systems. In O. Maler and A. Pnueli, editors, *Hybrid Systems: Computation and Control (HSCC)*, volume 2623 of *LNCS*. Springer, 2003.