A method for invariant generation for polynomial continuous systems *

Andrew Sogokon[1], Khalil Ghorbal[2], Paul B. Jackson[1], and André Platzer[2]

[1] University of Edinburgh, LFCS, School of Informatics, Edinburgh, Scotland, UK
a.sogokon@sms.ed.ac.uk, pbj@inf.ed.ac.uk
[2] Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, USA
{kghorbal|aplatzer}@cs.cmu.edu

Abstract This paper presents a method for generating semi-algebraic invariants for systems governed by non-linear polynomial ordinary differential equations under semi-algebraic evolution constraints. Based on the notion of discrete abstraction, our method eliminates unsoundness and unnecessary coarseness found in existing approaches for computing abstractions for non-linear continuous systems and is able to construct invariants with intricate boolean structure, in contrast to invariants typically generated using template-based methods. In order to tackle the state explosion problem associated with discrete abstraction, we present invariant generation algorithms that exploit sound proof rules for safety verification, such as *differential cut* (DC), and a new proof rule that we call *differential divide-and-conquer* (DDC), which splits the verification problem into smaller sub-problems. The resulting invariant generation method is observed to be much more scalable and efficient than the naïve approach, exhibiting orders of magnitude performance improvement on many of the problems.

1 Introduction

Establishing safe operation of embedded systems arising in modern engineering increasingly involves reasoning about the behaviour of hybrid dynamical systems that combine discrete and continuous state evolution. Continuous dynamics is typically specified by ordinary differential equations (ODEs). Non-linear ODEs afford the engineer the means of modelling rich dynamic behaviour that cannot possibly occur in linear systems [12], but are also notoriously difficult to analyse because they rarely possess solutions that can be expressed in closed form.

This paper is concerned with the problem of automating safety verification for continuous systems modelled by non-linear ODEs under evolution constraints, which is a problem of broader interest to automating safety verification for hybrid dynamical systems. To solve the verification problem, one requires a proof that a given continuous system does not evolve into an unsafe state at any future time from some given initial

* This material is based upon work supported by the UK Engineering and Physical Sciences Research Council (EPSRC) under grants EP/I010335/1 and EP/J001058/1, the National Science Foundation by NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181, CNS-0931985 and DARPA FA8750-12-2-0291.

configuration while obeying its evolution constraint. Additionally, given that solutions are rarely available, it is highly desirable to arrive at such a proof by working with the ODEs directly, i.e. *without solving the initial value problem*.

Traditionally, two popular techniques have been used for proving safety properties without computing solutions or putting a finite bound on the duration of evolution in continuous systems: one based on first *soundly abstracting* the continuous system and performing reachability analysis in the resulting discrete transition system, and a *deductive verification* approach that works by reasoning about appropriate *invariants* in the continuous system.

Deductive verification tools for hybrid systems crucially rely on (**i**) the ability to prove invariance assertions about continuous systems (which was solved for the case of semi-algebraic [3] invariants and polynomial ODEs in [14]) and (**ii**) having the means of *automatically generating* continuous invariants sufficient to prove safety assertions about continuous systems. In practice, this latter point is often the main bottleneck when verifying safety of hybrid systems in which the continuous dynamics are non-linear.

Existing automatic procedures for generating invariants for use in deductive frameworks only make limited use of the boolean structure in invariants. Approaches based on abstraction, in computing reachable sets of discrete systems, (implicitly) create invariants with more intricate boolean structure; their limitations currently stem from the conservative nature of the discrete models, whose transition behaviour is often a very coarse over-approximation of the evolution taking place in the continuous system.

A number of approaches have been proposed for generating invariants for continuous systems [24,38,11,28,27,14,44,8,16], which either put serious restrictions on the form of the invariant or rely on the user pre-defining a *template* and then attempt to find an instantiation of the parameters in the template that yields an invariant. In this paper we pursue an alternative approach that automatically generates semi-algebraic continuous invariants from discrete semi-algebraic abstractions of continuous systems. Our rationale is that recent advances in semi-algebraic invariant checking for polynomial ODEs [14] allow deductive provers to work with arbitrary semi-algebraic invariants, yet few methods for invariant generation are able to synthesize interesting invariants with boolean structure that one might find in reachable sets of discrete abstractions. At the same time, discrete abstraction approaches do not take full advantage of the results on invariant checking in constructing the transition relation for the discrete transition system. We seek to address both of these issues.

Currently available methods for creating semi-algebraic abstractions of non-linear polynomial systems [36,37] result in abstractions that are unsound for certain degenerate cases and unnecessarily coarse even in very simple scenarios. Additionally, discrete abstraction is known to scale poorly owing to (in the worst case) an exponential increase in the number of discrete states as the continuous state space is partitioned [37], making it very difficult to refine abstractions. To ameliorate this situation, we give a method for constructing semi-algebraic abstractions that are sound and only as coarse as the partitioning of the continuous state space into discrete regions itself. We then

[3] A semi-algebraic set is a subset of $\mathbb{R}^n$ characterized by a finite boolean combination of sets defined by polynomial equalities and inequalities.

employ ideas from deductive verification to give more scalable and efficient algorithms for generating semi-algebraic invariants for polynomial continuous systems.

Contributions.

In Section 3 of this paper we (**I**) introduce a method for constructing semi-algebraic abstractions of polynomial continuous systems in which transitions between the discrete states occur *if and only if* a corresponding continuous evolution is possible in the continuous system. In Section 4 we give an algorithm for generating semi-algebraic invariants for polynomial continuous systems by efficiently extracting reachable sets from these abstractions. In Section 5 we (**II**) introduce a sound proof rule DDC (*differential divide-and-conquer*) which works to split the safety verification problem into smaller sub-problems by exploiting properties of invariant real algebraic sets and (**III**) give more scalable invariant generation algorithms employing sound proof rules *differential weakening* (DW) [19] and *differential cut* (DC) [19,21] together with the new rule DDC to address the discrete state explosion problem associated with computing abstractions. In Section 6 we (**IV**) evaluate our techniques on a collection of 100 safety verification problems featuring predominantly non-linear ODEs.

2 Preliminaries

To simplify our presentation, we will use the notation for sets and formulas characterizing those sets interchangeably in this paper, e.g. H will denote both a semi-algebraic set $H \subseteq \mathbb{R}^n$ and a formula H in the first-order theory of real arithmetic with free variables in $x_1, \ldots, x_n$ that characterizes this set. In what follows, we shall restrict our attention to autonomous [4] systems of polynomial ordinary differential equations under semi-algebraic evolution domain constraints [5] , i.e. systems of the form:

$$\dot{x}_i = f_i(\boldsymbol{x}), \quad \boldsymbol{x} \in H \subseteq \mathbb{R}^n,$$

where $f_i \in \mathbb{R}[x_1, \ldots, x_n]$ for $1 \leq i \leq n$ and the evolution domain constraint H is semi-algebraic. We will write this concisely using vector notation as $\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H$.

One may wonder at this stage whether restricting attention to polynomial systems represents a severe limitation; after all, non-linearities involving transcendental functions such as $\sin, \cos, e, \ln$, etc., are not uncommon in systems of practical interest. Fortunately, it is often possible to transform such systems into (larger) polynomial systems by introducing fresh variables and eliminating non-polynomial non-linearities in a rather general technique [23], which is known in various scientific communities as *recasting* [30,17] or *differential axiomatization* [19]. Furthermore, it has been shown that such a transformation can be mechanised for a broad class of non-polynomial systems using a terminating algorithm [15]. Likewise, no generality is lost by only considering autonomous systems because any system with explicit time dependence $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, t) \;\&\; H$

[4] In the sense of not having an *explicit* dependence on the time variable t.

[5] Evolution constraints are often used to define operating modes in hybrid and cyber-physical systems (so-called *mode*, or *location invariants* in the parlance of hybrid automata [1,13]).

can be transformed into an autonomous system by introducing a fresh variable to model time evolution, e.g. if we add $\dot{x}_{n+1} = 1$ to the system and replace every instance of t in the system with x_{n+1}.

To state the safety verification problem for continuous systems in full generality we require a set of *initial states* for the system, which we denote by $\psi \subseteq \mathbb{R}^n$, and a set of *safe states* denoted $\phi \subseteq \mathbb{R}^n$. The problem is to prove that starting inside ψ, the system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ & H cannot leave ϕ by evolving inside the evolution domain constraint H. We will only consider semi-algebraic ψ and ϕ in this paper and will state the safety property formally, using notation from differential dynamic logic ($\mathrm{d}\mathcal{L}$) [18], as follows:

$$\psi \rightarrow [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H]\, \phi.$$

The above formula asserts that, starting in any state satisfying the pre-condition (ψ), the system will *necessarily* (box modality [ ]) satisfy the post-condition (ϕ) when following the system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ & H for any amount of time.[6] The semantic definition of the $\mathrm{d}\mathcal{L}$ assertion above is given in terms of the solution, which precisely describes how continuous states evolve over time. A solution to the initial value problem for the system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ with initial value $\boldsymbol{x}_0 \in \mathbb{R}^n$ is a differentiable function $\varphi_t(\boldsymbol{x}_0) : (a, b) \rightarrow \mathbb{R}^n$ defined for t in some non-empty *interval of existence* $(a, b) \subseteq \mathbb{R} \cup \{\infty, -\infty\}$ including zero and such that $\frac{d}{dt}\varphi_t(\boldsymbol{x}_0) = f(\varphi_t(\boldsymbol{x}_0))$ for all $t \in (a, b)$. Formally, the $\mathrm{d}\mathcal{L}$ continuous safety assertion above is valid if the following is true:

$$\forall\, \boldsymbol{x}_0 \in \psi.\, \forall\, \tau \geq 0.\, (\forall\, t \in [0, \tau]\, .\varphi_t(\boldsymbol{x}_0) \in H) \rightarrow \varphi_\tau(\boldsymbol{x}_0) \in \phi.$$

In practice, solutions to non-linear ODEs are almost never available in closed form (by which we understand a *finite* expression in terms of polynomials and elementary functions); even when they are, the resulting sentences often belong to an undecidable theory [26] due to transcendental functions in the closed form expression. Alternatively, the safety verification problem can sometimes be solved directly in a deductive framework. This involves finding an appropriate set $I \subseteq \mathbb{R}^n$, called a *continuous invariant* [22], that satisfies the three premises (above the bar) of the following rule of inference:

$$(\mathit{Safety})\ \frac{H \wedge \psi \rightarrow I \quad I \rightarrow [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H]\, I \quad I \rightarrow \phi}{\psi \rightarrow [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H]\, \phi}$$

to conclude (below the bar) that the system is safe. Continuous invariants generalize *positively invariant sets* [6] to systems under evolution constraints.

Definition 1 (Continuous invariant [22]). *For a continuous system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ & H, a set $I \subseteq \mathbb{R}^n$ is a continuous invariant if and only if*

$$\forall\, \boldsymbol{x}_0 \in I.\, \forall\, \tau \geq 0.\, (\forall\, t \in [0, \tau].\, \varphi_t(\boldsymbol{x}_0) \in H) \rightarrow \varphi_t(\boldsymbol{x}_0) \in I.$$

Intuitively, a continuous invariant is any set of states I such that any motion initialized inside I that respects the evolution constraint H is guaranteed to remain inside I.

[6] Considering the continuous system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ & H as a program, the safety assertion $\psi \rightarrow [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H]\, \phi$ expresses the (continuous) Hoare triple $\{\psi\}\, \dot{\boldsymbol{x}} = f(\boldsymbol{x})$ & $H\, \{\phi\}$.

When H and I are semi-algebraic and f_i are polynomial, a decision procedure for checking whether I is a continuous invariant was reported in [14], enabling us to decide $\mathrm{d}\mathcal{L}$ assertions of the form $I \rightarrow [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \ \& \ H]\ I$. The decision procedure involves computing higher-order Lie derivatives and exploits the ascending chain property of Noetherian rings. The interested reader is invited to consult [14] for a detailed description of the procedure and also [8], where similar ideas were employed. As a direct consequence, every premise of the rule $(Safety)$ is known to be decidable, since ψ, ϕ and H are also assumed to be semi-algebraic, the goals $H \wedge \psi \rightarrow I$ and $I \rightarrow \phi$ can be passed to a decision procedure for real arithmetic [35]. The challenge in applying the rule now lies in *finding* an appropriate continuous invariant I.

3 Discrete Abstraction of Continuous Systems

In a certain sense, with discrete abstraction one seeks to approximate continuous systems by finite discrete transition systems. Such a transformation makes it possible to perform reachability analysis and verify safety properties in the simpler discrete model. The approach works by ensuring that the set of behaviours of the discrete (abstract) system *over-approximates* the set of behaviours of the continuous (concrete) system; this is known as *sound abstraction*. If the discrete abstraction is sound, then any violation of the safety property in the continuous system is necessarily reproduced by the abstract discrete transition system. Conversely, an abstraction is *complete* (with respect to the safety property) when any violation of the safety property in the abstraction is reproduced by the concrete continuous system.

Discrete abstraction of continuous systems was previously studied in [2,3] (for linear systems) and [36,37] (for more general non-linear systems), where a simple method for constructing abstractions was proposed but results in discrete systems that may feature transitions between discrete states that are impossible in the continuous system. In this section we describe the process of constructing sound and *exact* abstractions of non-linear continuous systems. That is, the resulting abstraction will feature a discrete transition between two abstract states *if and only if* a corresponding continuous trajectory is possible in the concrete system. The method we use is fundamentally different from [36,37] in computing the discrete transition relation using a *decision procedure* for continuous invariant assertions [14].

3.1 Constructing the Discrete State Space

In this section we describe a way of partitioning the evolution domain constraint H in the continuous system $\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \ \& \ H$ using a set of polynomial functions.

Definition 2 (Semi-algebraic decomposition). *A semi-algebraic decomposition of a semi-algebraic set $H \subseteq \mathbb{R}^n$ by a set of m polynomials $A \subset \mathbb{R}[x_1, \ldots, x_n]$ is a partition of H into $k \leq 3^m$ regions giving all the* non-empty *intersections of the form $H \cap p_1 \sim_1 0 \cap \cdots \cap p_m \sim_m 0$ where $p_i \in A$ and $\sim_i \in \{<, =, >\}$ for $1 \leq i \leq m$.*

Computing the semi-algebraic decomposition of the evolution domain constraint H for a finite set of polynomials A can be achieved using a simple procedure that we will call

$SemiAlgDecomp$. The decomposition defines a partition of H into k non-empty regions, each corresponding to a single *discrete state*, which we denote by $\mathbf{s}_i$, where $1 \leq i \leq k$. We will denote by S the set of all discrete states obtained from the semi-algebraic decomposition, i.e. $S \equiv \{\mathbf{s}_i \mid 1 \leq i \leq k\}$.

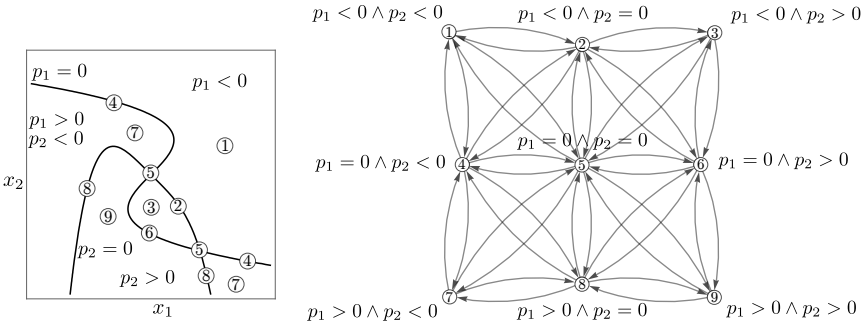
3.2 Constructing the Transition Relation

We now apply the decision procedure for semi-algebraic continuous invariant assertion checking reported in [14] to exactly determine the transition relation $T \subset S \times S$, enabling us to construct *exact* discrete abstractions, which we denote by the pair (S, T). We will write $\mathbf{s}_i \longrightarrow \mathbf{s}_j$ for $(\mathbf{s}_i, \mathbf{s}_j) \in S \times S$, the discrete transition from state $\mathbf{s}_i$ to $\mathbf{s}_j$.

We begin with a transition relation $S \times S$ in which every state is reachable from every other state (including itself) in a single discrete transition. First, let us observe that a continuous solution of the differential equation cannot pass from a discrete state where $p > 0$ (for some polynomial $p \in A$) to a state where $p < 0$ without passing through $p = 0$ first, nor vice versa. Using this intuition, we can give a general definition of what it means for two discrete states to be *neighbouring* (or *adjacent* [34]).

Definition 3. *Let S be the set of discrete states constructed from a semi-algebraic decomposition of H by a finite set of polynomials $A \subset \mathbb{R}[x_1, \ldots, x_n]$. Two discrete states $\mathbf{s}_i, \mathbf{s}_j \in S$, where $i \neq j$, are neighbouring if there are* **no** *points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbf{s}_i \cup \mathbf{s}_j$ such that $p(\boldsymbol{x}_1) < 0$ and $p(\boldsymbol{x}_2) > 0$ for any p in A.*

We can now construct a *neighbouring transition relation* $T_n \subseteq S \times S$ in which only the neighbouring states are reachable in a single transition (note that a state cannot be its own neighbour using our definition). Intuitively, in the neighbouring transition relation one cannot "jump across" $p = 0$ in a single discrete transition; at the same time, any state is reachable from any other state. An abstraction which results from (S, T_n) is still maximally coarse and therefore not very useful (illustrated in Fig. 1).



(a) Discretization S.

(b) Transition system (S, T_n).

Figure 1: Semi-algebraic decomposition of $\mathbb{R}^2$ by $A = \{p_1, p_2\}$ resulting in 9 discrete states $S \subset 2^{\mathbb{R}^2}$ and the neighbouring transition relation $T_n \subset S \times S$.

We are only interested in retaining those discrete transitions for which the corresponding continuous transitions are possible in the original continuous system. In order

to eliminate impossible discrete transitions we need to *decide* an invariance assertion:

$$\mathbf{s}_i \to [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; (\mathbf{s}_i \vee \mathbf{s}_j)]\; \mathbf{s}_i,$$

for each pair of neighbouring discrete states $(\mathbf{s}_i, \mathbf{s}_j) \in T_n$; we will proceed to remove transitions $\mathbf{s}_i \longrightarrow \mathbf{s}_j$ from T_n if and only if the decision procedure for continuous invariance assertions returns True. This process can be mechanized in a terminating abstraction algorithm that we call $ExactAbstraction$. The result is a discrete transition system (S, T) with a transition relation $T \subseteq T_n$ that does not feature discrete transitions that are impossible; we will state this property formally.

Proposition 4. *Abstractions (S, T) are* **exact** *with respect to the discretization, i.e. $\mathbf{s}_i \longrightarrow \mathbf{s}_j$ is in T if and only if*

$$\exists\, \boldsymbol{x}_0 \in \mathbf{s}_i.\; \exists\, \tau > 0.\; \varphi_0(\boldsymbol{x}_0) \in \mathbf{s}_i \;\wedge\; \varphi_\tau(\boldsymbol{x}_0) \in \mathbf{s}_j \quad \textit{and} \quad \forall\, t \in [0, \tau].\; \varphi_t(\boldsymbol{x}_0) \in \mathbf{s}_i \cup \mathbf{s}_j,$$

that is, if and only if the system may evolve continuously from state $\mathbf{s}_i$ into a neighbouring state $\mathbf{s}_j$ without leaving their union $\mathbf{s}_i \cup \mathbf{s}_j$. The abstraction is exactly as coarse as the partition of the evolution constraint H into regions corresponding to discrete states.

One can view the process of removing impossible discrete transitions as a sound refinement of the neighbouring transition relation to $T \subseteq T_n$. In the worst case, using a set of m polynomials for the semi-algebraic decomposition of H will result in 3^m discrete states and a neighbouring transition relation T_n with a total of $7^m - 3^m$ discrete transitions that need to be checked. In practice, both the number of discrete states and the number of transitions in T_n will typically be much lower than the pessimistic worst case bound. Furthermore, removing impossible transitions from T_n is a massively parallel problem, allowing one to exploit multi-core parallelism instead of iterating through the transitions sequentially.

3.3 Sound and exact abstraction

We will now discuss some important differences between earlier work and our approach. The discrete abstraction method reported in [37] is fundamentally different in the way it constructs the transition relation (let us call it $T_\sim \subseteq S \times S$), which is described in [37, Section 3.2.2]. In essence, the method imposes conditions for removing transitions from the neighbouring transition relation T_n in the following way: given two neighbouring states $\mathbf{s}_i$ and $\mathbf{s}_j$, it removes the transition $\mathbf{s}_i \longrightarrow \mathbf{s}_j$ from T_n if *any* of the following conditions are satisfied for any $p \in A$:

1. $\mathbf{s}_i$ has $p < 0$ and $\mathbf{s}_j$ has $p = 0$ and $\mathbf{s}_i \to \frac{dp}{dt} \leq 0$ is true,
2. $\mathbf{s}_i$ has $p > 0$ and $\mathbf{s}_j$ has $p = 0$ and $\mathbf{s}_i \to \frac{dp}{dt} \geq 0$ is true,
3. $\mathbf{s}_i$ has $p = 0$ and $\mathbf{s}_j$ has $p < 0$ and $(\mathbf{s}_i \to \frac{dp}{dt} = 0 \vee \mathbf{s}_i \to \frac{dp}{dt} > 0)$ is true,
4. $\mathbf{s}_i$ has $p = 0$ and $\mathbf{s}_j$ has $p > 0$ and $(\mathbf{s}_i \to \frac{dp}{dt} = 0 \vee \mathbf{s}_i \to \frac{dp}{dt} < 0)$ is true.

Remark 5. The abstraction method in [37] also considers so-called *stuttering* (also *self-looping* [34]) transitions $\mathbf{s}_i \longrightarrow \mathbf{s}_i$, which we disregard here (already in the way we define T_n). This discrepancy makes no practical difference to safety verification as stuttering transitions have no effect on the reachable sets of discrete abstractions.

The approach described in [37] is not (in general) sound when the polynomials in A are allowed to be non-linear. To see this, consider the simple system with constant derivatives $\dot{x}_1 = 1, \dot{x}_2 = 0$ and let $A = \{x_1^2 + x_2, x_2 - x_1^2\}$. The abstraction one obtains (Fig. 2) suggests that the state $x_1^2 + x_2 = 0 \land x_2 - x_1^2 = 0$ (equivalent to $x_1 = 0 \land x_2 = 0$) is invariant under the flow of the system, which is incorrect. The nature of this problem was studied in non-convex analysis; a solution would require reasoning about the *contingent cone* [42], which is not in general computable. A sound and exact abstraction using our approach is shown in Fig. 3.

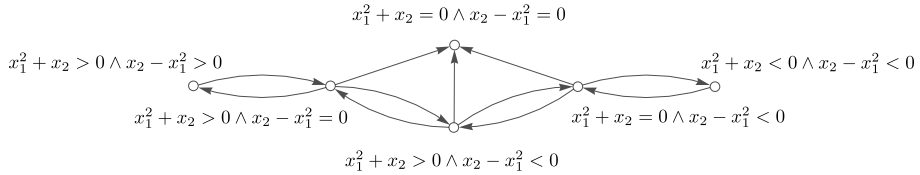


Figure 2: Abstraction $(S, T_\sim)$ generated using method from [37].

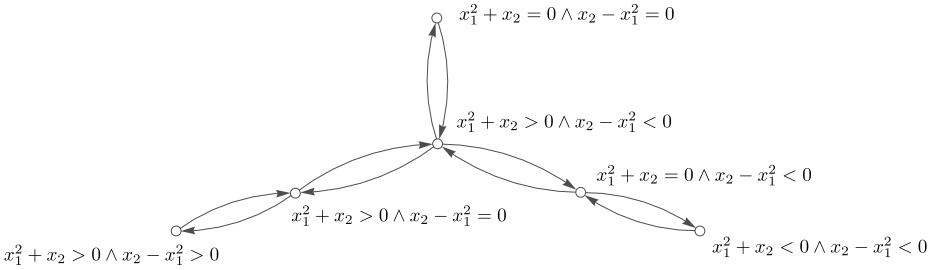


Figure 3: Sound abstraction (S, T) generated by $ExactAbstraction$.

The abstraction method in [37] additionally suffers from coarseness, because it can introduce discrete transitions that correspond to evolutions that are impossible in the concrete continuous system (the abstraction is therefore *inexact*). For instance, consider a planar system of non-linear ordinary differential equations featuring a stable limit cycle in the form of a unit circle enclosing an equilibrium at the origin:

$$\dot{x}_1 = -x_1^3 - x_2^2 x_1 + x_1 + x_2,$$
$$\dot{x}_2 = -x_2^3 - x_1^2 x_2 + x_2 - x_1.$$

Let the system evolve under no evolution constraints and consider a simple discretization by the axes polynomials, i.e. take $A = \{x_1, x_2\}$. The discrete abstraction $(S, T_\sim)$ generated using the method from [37] is shown in Fig. 4. An exact abstraction (S, T) without impossible transitions generated using our approach is shown in Fig. 5. Abstraction $(S, T_\sim)$ considers the origin reachable, while (S, T) does not.

4 Extracting Continuous Invariants from Discrete Abstractions

If one constructs a (sound) discrete abstraction of some system $\dot{\boldsymbol{x}} = f(\boldsymbol{x})$ & H using some finite set of polynomials A, one may verify safety properties by showing that they

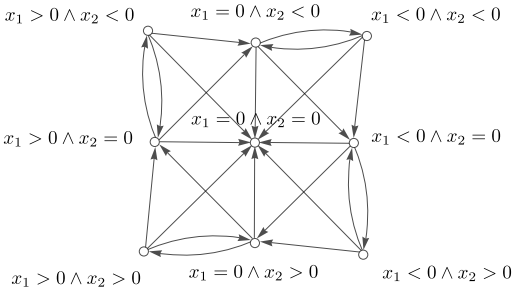


Figure 4: Inexact abstraction $(S, T_{\sim})$ generated using method from [37].

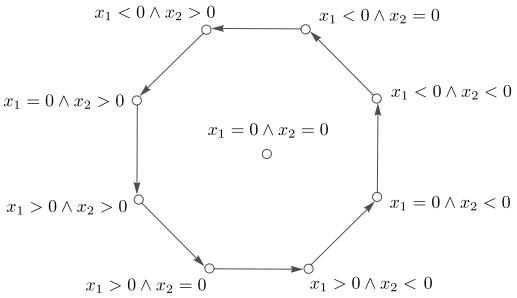


Figure 5: Exact abstraction (S, T) generated by $ExactAbstraction$.

hold in the abstraction. For this, one needs to check whether an unsafe abstract state (i.e. one which contains a state that satisfies the formula $\neg\phi$) is reachable by following the discrete transitions starting from the set of initial abstract states (those defining regions where ψ is satisfiable). If none of the unsafe abstract states are reachable from the initial states in the abstraction, one can conclude that the continuous system is safe.

By computing the forward-reachable set from the set of the initial states ψ in the abstraction, which we denote by $Reach_A^{\rightarrow}(\psi, H) \subseteq H$, one generates a continuous invariant. Provided the abstraction is *exact*, this is the *smallest* continuous invariant with respect to the discretization by the polynomials in A and is furthermore *semi-algebraic*. Formally, we define

$$Reach_A^{\rightarrow}(\psi, H) \equiv \bigvee_{\substack{i \text{ s.t. } \mathbf{s}_i \cap \psi \neq \emptyset, \\ j \text{ s.t. } \mathbf{s}_i \longrightarrow^* \mathbf{s}_j}} \mathbf{s}_j \ ,$$

where $\longrightarrow^*$ represents the reachability relation; that is, $\mathbf{s}_i \longrightarrow^* \mathbf{s}_j$ if state $\mathbf{s}_j$ is reachable from $\mathbf{s}_i$ in zero or more discrete transitions in the exact abstraction (S, T), obtained from the discretization by polynomials in A. Thus, $I \equiv Reach_A^{\rightarrow}(\psi, H)$ is a semi-algebraic set that is (by construction) guaranteed to include the initial set (i.e. $\psi \rightarrow I$) and is a continuous invariant for the system (i.e. $I \rightarrow [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \ \& \ H]\ I$). If it is also true that I does not include any unsafe states (i.e. $I \rightarrow \phi$), then I is sufficient to conclude that the system is safe using the proof rule $(Safety)$ from Section 2.

For invariant generation we are merely interested in extracting a semi-algebraic continuous invariant containing the initial set of states ψ from the abstraction, not the full abstraction (S, T) itself. We now give a simple worklist procedure that we call $LazyReach$ (Algorithm 1) for constructing the set $Reach_A^{\rightarrow}(\psi, H)$ *lazily* (on demand), i.e. without eagerly constructing the exact abstraction (S, T) first.

Algorithm 1: $LazyReach$

Data: $\psi, \dot{\boldsymbol{x}} = f(\boldsymbol{x})\ \&\ H, A$
Result: $Reach_A^{\rightarrow}(\psi, H)$

$S := SemiAlgDecomp(\{H\}, A)$;
$T_n := NeighbourTrans(S)$;
$Visited := \{\mathbf{s} \in S \mid \mathbf{s} \cap \psi \neq \emptyset\}$;
$Processed := \{\}$;
while $|Processed| < |Visited|$ **do**
  $Unprocessed := Visited \setminus Processed$;
  $Processed := Visited$;
  foreach $\mathbf{s}_i$ *in* $Unprocessed$ **do**
    $Validate := \{(\mathbf{s}_i, \mathbf{s}_j) \in T_n \mid \mathbf{s}_j \notin Visited\}$;
    foreach $(\mathbf{s}_i, \mathbf{s}_j)$ *in* $Validate$ **do**
      if $\neg(\mathbf{s}_i \rightarrow [\dot{\boldsymbol{x}} = f(\boldsymbol{x})\ \&\ (\mathbf{s}_i \vee \mathbf{s}_j)]\ \mathbf{s}_i)$ **then**
        $Visited := Visited \cup \{\mathbf{s}_j\}$;
return $\bigvee_{\mathbf{s} \in Visited} \mathbf{s}$

Although the worst-case running time of $LazyReach$ is exponential in $m = |A|$, in practice employing Algorithm 1 is often far more efficient than computing the exact abstraction (S, T) in full and then extracting $Reach_A^{\rightarrow}(\psi, H)$.

5 Tackling Discrete State Explosion

Discrete abstractions of continuous systems suffer from the discrete state explosion problem, i.e. the number of discrete states in the abstraction grows exponentially with the number of polynomials $m = |A|$ used for the discretization.

If one is to consider each individual polynomial $p \in A$, it is intuitive that if one can show that

1. for the initial set of states ψ, the polynomial p is sign-invariant, i.e. $p(\psi) \sim 0$ where $\sim \in \{<, =, >\}$, and
2. that this sign condition defines a continuous invariant for the system, i.e. $p \sim 0 \rightarrow [\dot{\boldsymbol{x}} = f(\boldsymbol{x})\ \&\ H]\ p \sim 0$,

then one can *refine* the evolution constraint to $H \wedge p \sim 0$ and *remove* the polynomial p from A and obtain an abstraction by the polynomials $B \equiv A \setminus \{p\}$ which has the property that

$$Reach_B^{\rightarrow}(\psi, H \wedge p \sim 0) \equiv Reach_A^{\rightarrow}(\psi, H).$$

The number of discrete states generated using B for the semi-algebraic decomposition of $H \wedge p \sim 0$ is at most 3^{m-1} and the process can be repeated for other polynomials that remain in B. This section will explore approaches to tackling the discrete state space explosion based on this observation without making the abstraction unnecessarily coarse. For this purpose we will use sound proof rules *differential cut* and *differential divide-and-conquer*.

5.1 Differential Cut

Platzer and Clarke [22] explored an approach to safety verification based on iteratively refining the evolution constraint H with *differential invariants* (a subset of continuous invariants, see [19]). Such a *sound* refinement of the evolution domain is possible using an inference rule called *differential cut* [21] (henceforth DC). Differential cuts are used repeatedly in a process called *differential saturation* (see [22, Proposition 2]). The DC rule formalizes the idea that it is always sound to restrict the evolution domain H by some continuous invariant F, provided that it includes the initial set ψ, i.e.

$$(\mathrm{DC})\frac{\psi \to [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H]F \quad \psi \to [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H \wedge F]\,\phi}{\psi \to [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H]\,\phi}$$

the original rationale being that it is easier to prove the safety property in the more restricted system in the right premise.

5.2 Differential Divide-and-Conquer

We now introduce a new proof rule, akin to DC, that goes further and exploits a property of sets that are continuous invariants in both positive and negative time directions to split the continuous system into smaller continuous sub-systems between which there is no continuous evolution.

Proposition 6. *The proof rule* DDC *given below (with five premises) is sound.*

$$\begin{array}{c} p = 0 \to [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H]\,p = 0 \\ p = 0 \to [\dot{\boldsymbol{x}} = -f(\boldsymbol{x}) \;\&\; H]\,p = 0 \\ \\ \psi \wedge p > 0 \to [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H \wedge p > 0]\,\phi \\ \psi \wedge p = 0 \to [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H \wedge p = 0]\,\phi \\ \psi \wedge p < 0 \to [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H \wedge p < 0]\,\phi \end{array} (\mathrm{DDC})\frac{}{\psi \to [\dot{\boldsymbol{x}} = f(\boldsymbol{x}) \;\&\; H]\,\phi}$$

Proof. For a continuous function p, no continuous trajectory inside H can cross from a region where $p > 0$ to a region where $p < 0$ without first crossing $p = 0$. If the first two premises hold, then $p = 0$ cannot be left inside H in either positive or negative time, i.e. there are no solutions entering or leaving $p = 0$ inside H. The reachable sets of the system initialized in $\psi \wedge p > 0$, $\psi \wedge p = 0$ and $\psi \wedge p < 0$ are thus disjoint and confined to regions of H where $p > 0$, $p = 0$ and $p < 0$ respectively. The union of these sets constitutes the reachable set of the system initialized in ψ and the result follows. □

Informally, the rule allows one to split the original system into three *dynamically disconnected regions*, that is disjoint regions that are not connected by a continuous flow of the system [7]. Note that unlike DC, the rule DDC does *not* require the initial set ψ to be wholly contained inside $p > 0$, $p = 0$ or $p < 0$. Instead, DDC splits the initial set of

[7] All three regions are *invariant sets* in the terminology of dynamical systems [5, Chapter II].

states into three disjoint initial subsets $\psi \wedge p > 0$, $\psi \wedge p = 0$ and $\psi \wedge p < 0$. The rule DDC thus decomposes the original safety assertion into three independent safety assertions about smaller sub-systems, allowing the user to work on these separately. DDC is of practical interest in cases when *two or more* of the sets $\psi \wedge p > 0$, $\psi \wedge p = 0$ and $\psi \wedge p < 0$ are non-empty (otherwise, ψ lies entirely within $p > 0$, $p = 0$ or $p < 0$ and DC may be applied to refine the constraint).

We now turn to applying the rules DC and DDC to tackle the state space explosion problem. In Algorithm 2 we give a procedure for refining the evolution domain constraint and removing polynomials from A, whenever this is possible, using the proof rules DC and DDC. We call this procedure DWC as it also exploits the sound reasoning principle of *differential weakening* DW [19], i.e.

$$\text{(DW)}\ \frac{H \to \phi}{\psi \to [\dot{\boldsymbol{x}} = f(\boldsymbol{x})\ \&\ H]\ \phi},$$

which simply requires that the evolution domain be contained within the post-condition to conclude that the system is safe.

Algorithm 2: DWC

```
   Data: ψ, ẋ = f(x) & H, φ, A
   Result: Continuous invariant I s.t. ψ ⊆ I
1  if H ∧ ψ → False then
2      return False
3  if H → φ then
4      return H                                                        // DW
5  foreach p ∈ A do
6      if (H ∧ ψ → p > 0) ∧ (p > 0 → [ẋ = f(x) & H] p > 0) then
7          return DWC(ψ, ẋ = f(x) & H ∧ p > 0, φ, A \ {p})             // DC
8      if (H ∧ ψ → p < 0) ∧ (p < 0 → [ẋ = f(x) & H] p < 0) then
9          return DWC(ψ, ẋ = f(x) & H ∧ p < 0, φ, A \ {p})             // DC
10     if (H ∧ ψ → p = 0) ∧ (p = 0 → [ẋ = f(x) & H] p = 0) then
11         return DWC(ψ, ẋ = f(x) & H ∧ p = 0, φ, A \ {p})             // DC
12 foreach p ∈ A do
13     if (p = 0 → [ẋ = f(x) & H] p = 0) ∧ (p = 0 → [ẋ = −f(x) & H] p = 0) then
14         GT := DWC(ψ ∧ p > 0, ẋ = f(x) & H ∧ p > 0, φ, A \ {p});
15         EQ := DWC(ψ ∧ p = 0, ẋ = f(x) & H ∧ p = 0, φ, A \ {p});
16         LT := DWC(ψ ∧ p < 0, ẋ = f(x) & H ∧ p < 0, φ, A \ {p});
17         return GT ∨ EQ ∨ LT                                         // DDC
18 return H
```

On lines 3 and 4, DWC applies the rule DW as a sufficiency check for termination. On lines 7, 9 and 11 the procedure discards those p for which $p > 0$, $p < 0$ or $p = 0$ describe a continuous invariant containing the initial set ψ (conditionals on lines 6, 8 and 10). This step corresponds to an application of the rule DC with $F \equiv p > 0$, $F \equiv p < 0$ and $F \equiv p = 0$ which, if the rule application is successful, are used to refine the evolution constraint H in the recursive call. If $p = 0$ is an invariant in both positive and negative time and does *not* contain all the initial states ψ, one can use the proof rule DDC to work with 3 smaller sub-systems of the original system whose reachable set may be constructed by *combining the reachable sets of these smaller systems*. This idea is implemented on lines 13-17 of Algorithm 2, where DWC recurses on the 3 smaller sub-systems and removes the polynomial p (used to divide the system) from A. The over-approximations of reachable sets obtained using these 3 recursive

calls are then combined into a union (line 17), which gives an over-approximation of the reachable set for the original system. Finally, when no further progress can be made, the procedure returns the evolution constraint H (line 18). Because the procedure only involves applying sound proof rules, one may view DWC as a *proof strategy* that can be implemented in a theorem prover. Indeed, if the procedure returns a result while there are still polynomials remaining in A, one has a proof of safety involving only the proof rules DW, DC and DDC.

Unlike $LazyReach$, the invariant generation procedure DWC will not (in general) always be able to find a sufficiently strong continuous invariant to prove the safety property, even if one exists in the semi-algebraic abstraction by the polynomials A. The invariants DWC is able to generate are thus generally coarser than those generated using $LazyReach$. However, we observe that in the worst case the running-time of DWC is only quadratic in the number of polynomials $m = |A|$, i.e. $\mathcal{T}_{DWC}(m) = O(m^2)$, compared the exponential time complexity of $LazyReach$.

We now combine the procedure DWC together with the $LazyReach$ algorithm by replacing $\mathbf{return}\ H$ on the final line (18) in DWC with

$$\mathbf{return}\ LazyReach(\psi, \dot{\boldsymbol{x}} = f(\boldsymbol{x})\ \&\ H, A).$$

We call the resulting new invariant generation procedure $DWCL$. Instead of returning H when no further progress can be made with DWC, $DWCL$ falls back to using the more expensive $LazyReach$ algorithm with the remaining polynomials. This combined procedure is theoretically as powerful as $LazyReach$, i.e. is capable of extracting the exact reachable set $Reach_A^{\rightarrow}(\psi, H)$ if necessary, but in practice also as fast as DWC, although theoretically the running time of $DWCL$ remains exponential in m.

Example 7 (Invariant generated using $DWCL$). Consider the non-linear planar system from [7, Ex. 10.7, p. 281] (with $H = \mathbb{R}^2$):

$$\begin{aligned}
\dot{x}_1 &= 2x_1\left(x_1^2 - 3\right)\left(4x_1^2 - 3\right)\left(x_1^2 + 21x_2^2 - 12\right), \\
\dot{x}_2 &= x_2\big(35x_1^6 + 105x_2^2x_1^4 - 315x_1^4 - 63x_2^4x_1^2 + 378x_1^2 + 27x_2^6 - 189x_2^4 + 378x_2^2 - 216\big),
\end{aligned}$$

As an initial set, take $\psi \equiv (x_1 - 1)^2 + x_2^2 < \frac{1}{4}$ and let $\phi \equiv x_1^2 + x_2^2 < 8$ be the post-condition. Consider an abstraction of this system using the irreducible polynomial factors of the right-hand side of the system of ODEs and the post-condition, i.e. let

$$\begin{aligned}
A = \{&x_1, x_1^2 - 3, 4x_1^2 - 3, x_2, x_1^2 + x_2^2 - 8, x_1^2 + 21x_2^2 - 12, \\
&35x_1^6 + 105x_2^2x_1^4 - 315x_1^4 - 63x_2^4x_1^2 + 378x_1^2 + 27x_2^6 - 189x_2^4 + 378x_2^2 - 216\}.
\end{aligned}$$

There are 7 abstraction polynomials in total, which in the worst case could lead to $3^7 = 2187$ discrete states and $7^7 - 3^7 = 821356$ discrete transitions in the neighbouring transition relation T_n. In practice, applying $LazyReach$ to generate the reachable set $Reach_A^{\rightarrow}(\psi, H)$ for this problem takes an unreasonable amount of time. The procedure DWC takes significantly less time to run, but is unable to find a suitable invariant using DW, DC and DDC alone. Our implementation of the combined procedure $DWCL$ is

able to generate the following continuous invariant $I \subset \phi$ in 104 seconds: [8]

$$\Big(\Big(35x_1^6 + 105\left(x_2^2 - 3\right)x_1^4 + 27\left(x_2^6 - 7x_2^4 + 14x_2^2 - 8\right) < 63x_1^2\left(x_2^4 - 6\right) \vee x_2 = 0\Big)$$
$$\wedge\, 4x_1^2 = 3 \wedge x_1 > 0\Big) \vee \left(x_2 = 0 \wedge \left(0 < x_1 < \frac{\sqrt{3}}{2} \vee \frac{\sqrt{3}}{2} < x_1 < \sqrt{3}\right)\right)$$
$$\vee \left(35x_1^6 + 105\left(x_2^2 - 3\right)x_1^4 + 27\left(x_2^6 - 7x_2^4 + 14x_2^2 - 8\right) < 63x_1^2\left(x_2^4 - 6\right)\right.$$
$$\left.\wedge\, x_1^2 + 21x_2^2 < 12 \wedge \left(0 < x_1 < \frac{\sqrt{3}}{2} \vee \left(2x_1 > \sqrt{3} \wedge x_1^2 < 3 \wedge x_2 \neq 0\right)\right)\right).$$

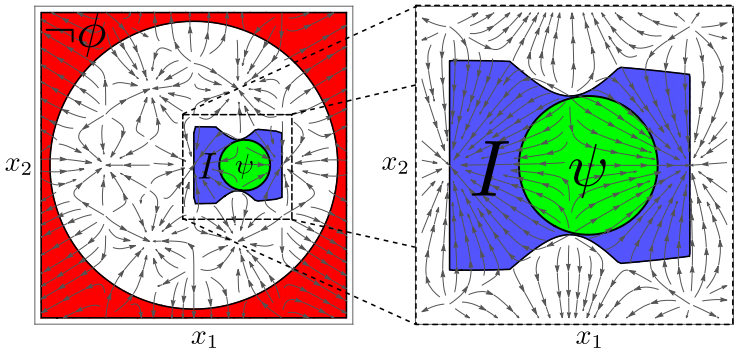


Figure 6: Phase portrait, unsafe states $\neg\phi$ (**red**), initial set ψ (**green**) and a generated continuous invariant $I \subset \phi$ (**blue**).

For this problem, the procedure $DWCL$ makes repeated use of both DC and DDC (each is used 4 times in total) before falling back to $LazyReach$, which in every instance is given 3 polynomials that remain to perform the abstraction (down from 7 in the original list A).

5.3 Sources of polynomials for abstraction

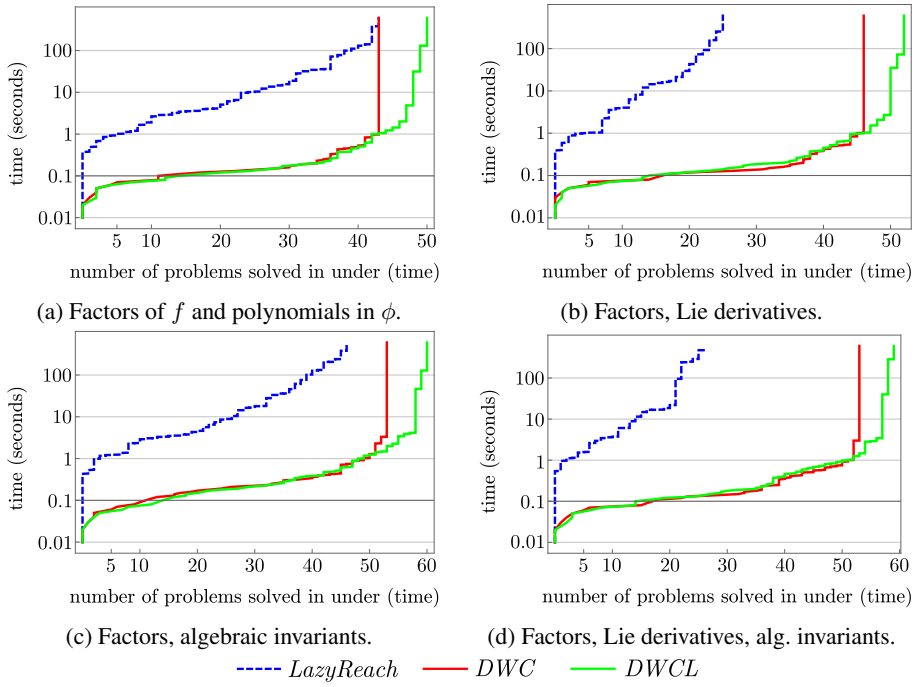
Discrete semi-algebraic abstraction relies on the user supplying a set of polynomials A to construct the set of discrete states through semi-algebraic decomposition of the evolution constraint. The verification problem itself is often a good source of polynomials; e.g. they could come from the description of the (semi-algebraic) post-condition ϕ, the pre-condition ψ, or indeed from the right-hand side of the (polynomial) system of ODEs, i.e. the polynomials $f_1, f_2, \ldots, f_n$, their irreducible factors, etc. The use of Lie derivatives as a source of polynomials for abstraction was previously investigated in [37] (see also [39] for related work). In [43] abstraction is explored using *Darboux polynomials* (see [10,9]), whose real roots are invariant under the flow of the system. Recent results on real algebraic invariant checking [8] enable us to consider a more general class of polynomials that share this property but are not necessarily Darboux.

[8] expression simplified in *Mathematica*.

6 Practical Evaluation

In this section we compare the performance of our invariant generation algorithms $LazyReach$, DWC and $DWCL$ on a set of 100 safety verification problems for continuous systems. The differential equations used in these problems are predominantly non-linear and originate from examples found in texts on dynamical systems [33,4,10,41,7,5], papers on the qualitative theory of ODEs and safety verification of continuous and hybrid systems [11,25,40,31,32]. [9]

The running time performance [10] of the algorithms is summarised in Figure 7. In the graphs, the vertical axis gives the dependent time variable (in seconds on a log scale) and the horizontal axis denotes the number of problems that could be solved in under the time given by the curve for each algorithm. By *solved* we understand that a semi-algebraic continuous invariant has been successfully generated and that it implies the postcondition, i.e. is sufficient to prove the safety assertion.



(a) Factors of f and polynomials in ϕ.

(b) Factors, Lie derivatives.

(c) Factors, algebraic invariants.

(d) Factors, Lie derivatives, alg. invariants.

Figure 7: Safety verification performance.

In our experiments we:

1. use polynomial factors of the right-hand side of the ODEs together with the factors of the polynomials appearing in the postcondition ϕ to create the set of polynomials A for the semi-algebraic decomposition (Fig. 7a),

[9] See `http://homepages.inf.ed.ac.uk/s0805753/invgen` for the problems.
[10] The comparison was performed on an *i5-3570K* CPU clocked at 3.40GHz.

2. extend the set A generated as in (1.) with Lie derivatives of every polynomial in A (Fig. 7b), and
3. explore the utility of using polynomials whose real roots are invariant real algebraic sets by extending the list of polynomials generated in (1.) and (2.) with polynomials generated using a method presented in [8] (Fig. 7c and Fig. 7d respecitvely).

In our results we observe that the DWC algorithm is significantly faster than $LazyReach$, confirming our hopes for gains in efficiency. We observe that, when using polynomial factors of the ODEs and the postcondition to abstract the system, $LazyReach$ was able to prove as many problems as DWC (43), although the set of problems solved is different. This is not surprising, since a proof strategy involving DW, DC and DDC, while very efficient, cannot in general be used to extract reachable sets of exact abstractions like the more expensive $LazyReach$. The combined method $DWCL$ (using DW, DC, and DDC before falling back to $LazyReach$) is seen to be both as practically efficient as DWC and able to solve more problems (50) than $LazyReach$ under a 600 second timeout; of course, given enough time, $DWCL$ and $LazyReach$ will both succeed at solving exactly the same problems (with $LazyReach$ taking significantly more time).

Adding the first Lie derivatives of the polynomial factors of the ODE and the postcondition effectively doubles the size of the list A which, unsurprisingly, leads to diminished performance of $LazyReach$ (only 25 problems solved) because it is heavily affected by the discrete state explosion problem. However, DWC is seen to perform slightly better than it did without the Lie derivatives in the list, solving a total of 46 safety verification problems. The $DWCL$ algorithm succeeds at proving safety in 52 of the problems.

We observe that adding algebraic invariants to the list of polynomial factors of the ODE and the postcondition resulted in a palpable improvement in the number of problems that could be solved. This is very clearly visible in the case of DWC, which is guaranteed to process every algebraic invariant by applying the proof rules DC and DDC. Overall, for this choice of polynomials we see $LazyReach$ solving 46, DWC solving 52, and $DWCL$ solving 60 problems out of 100 (see Fig. 7c). Likewise, by adding algebraic invariants to the list of polynomial factors and their Lie derivatives (as in 2.) we were able to solve 26, 53 and 59 problems using $LazyReach$, DWC and $DWCL$ respectively (Fig. 7d).

Overall, in every set of benchmarks we observe only one problem for which the algorithm DWC times out after 600 seconds, whereas $LazyReach$ times out for many of problems (e.g. in the experiments shown in Fig. 7d $LazyReach$ timed out on 59 of the problems and was unable to produce a suitable invariant within the time limit in only 15 instances). The procedure $DWCL$ generally times out more often than DWC, but significantly less frequently than $LazyReach$ (e.g. 25 problems from Fig. 7d resulted in a timeout, and 16 could not be solved using the resulting invariant).

These results are very encouraging as they demonstrate that the discrete state explosion problem can, to a certain extent, be addressed using algorithms such as $DWCL$ and that methods for automatic algebraic invariant generation (such as that in [8]) can be used to generate polynomials that will often improve the quality of the resulting abstractions.

It is perhaps surprising to see that many of the atomic predicates featuring irreducible factors of polynomials harvested from the problem define continuous invariants. As such, these polynomials are eminently suitable for processing using our algorithms DWC and $DWCL$ without incurring the performance penalty associated with building finer abstractions using the conventional approach.

7 Related Work

In [44], the authors apply their earlier results about checking semi-algebraic continuous invariants to address the invariant generation problem using approaches such as predefining parametric templates and restricting attention to classes of invariants (such as polyhedra), as well as using qualitative analysis techniques to suggest invariant templates. Our approach is different in that we do not rely on parametric templates and put no restrictions on the form of the semi-algebraic invariant which may be generated. Discrete abstraction of linear systems using *linear polynomials* to discretize the state space was investigated in [2,3]. A method for abstracting non-linear systems using non-linear polynomials was studied in [36,37], but results in abstractions that are inexact; the fundamental differences between this approach and our work is discussed at length in Section 3. A powerful technique called *relational abstraction* was introduced in [29]. With relational abstraction one aims to over-approximate the finite time reachability relation between states in a continuous system. Computing relational abstractions requires searching for appropriate invariants in a larger auxiliary continuous system; once a relational abstraction is available, one may use it to extract a continuous invariant containing any given initial state of the system. Computing good relational abstractions for non-linear systems is in practice expensive because it involves searching for invariants in continuous systems with double the original number of state variables.

8 Conclusion

This paper presented a powerful method for automatically discovering continuous invariants that can be used in a formal deductive system to prove safety assertions about continuous systems. We removed some important theoretical limitations (unsoundness and coarseness) in existing methods for constructing discrete abstractions of non-linear continuous systems and presented scalable and efficient algorithms for continuous invariant generation that combine discrete semi-algebraic abstraction with sound proof rules for deductive safety verification. Verification of hybrid systems constructively reduces to proving properties of differential equations [18,20], which provides a wider context for the future development of our work. The results we observe are highly encouraging, but much further work remains before safety verification (in the continuous fragment) of hybrid systems can enjoy a high level of automation. For instance, now that issues associated with inexact abstractions have been removed, the (difficult [39]) problem of finding a good choice of polynomials for constructing the semi-algebraic predicates is the only factor that determines the success of our approach. We observed that polynomials whose real roots themselves define invariants [8] can often be used to improve the quality of abstractions; however the broader problem of choosing the right polynomials leaves many interesting questions for future research.

References

1. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.H.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. pp. 209–229. Springer-Verlag (1992)
2. Alur, R., Dang, T., Ivančić, F.: Progress on reachability analysis of hybrid systems using predicate abstraction. In: HSCC. pp. 4–19 (2003)
3. Alur, R., Dang, T., Ivančić, F.: Predicate abstraction for reachability analysis of hybrid systems. ACM Trans. Embedded Comput. Syst. 5(1), 152–199 (2006)
4. Arrowsmith, D., Place, C.: Dynamical systems. Differential equations, maps and chaotic behaviour. Chapman & Hall (1992)
5. Bhatia, N.P., Szegő, G.P.: Stability Theory of Dynamical Systems. Springer (1970)
6. Blanchini, F.: Set invariance in control. Automatica 35(11), 1747–1767 (1999)
7. Dumortier, F., Llibre, J., Artés, J.C.: Qualitative Theory of Planar Differential Systems. Springer (2006)
8. Ghorbal, K., Platzer, A.: Characterizing algebraic invariants by differential radical invariants. In: TACAS. pp. 279–294 (2014)
9. Ghorbal, K., Sogokon, A., Platzer, A.: A hierarchy of proof rules for checking differential invariance of algebraic sets. In: VMCAI. pp. 431–448 (2015)
10. Goriely, A.: Integrability and Nonintegrability of Dynamical Systems. Advanced series in nonlinear dynamics, World Scientific (2001)
11. Gulwani, S., Tiwari, A.: Constraint-based approach for analysis of hybrid systems. In: Gupta, A., Malik, S. (eds.) CAV, LNCS, vol. 5123, pp. 190–203. Springer (2008)
12. Hale, J.K., LaSalle, J.P.: Differential equations: Linearity vs. nonlinearity. SIAM Review 5(3), 249–272 (Jul 1963)
13. Henzinger, T.A.: The theory of hybrid automata. pp. 278–292. IEEE Computer Society Press (1996)
14. Liu, J., Zhan, N., Zhao, H.: Computing semi-algebraic invariants for polynomial dynamical systems. In: Chakraborty, S., Jerraya, A., Baruah, S.K., Fischmeister, S. (eds.) EMSOFT. pp. 97–106. ACM (2011)
15. Liu, J., Zhan, N., Zhao, H., Zou, L.: Abstraction of elementary hybrid systems by variable transformation. In: FM 2015. pp. 360–377 (2015)
16. Matringe, N., Moura, A.V., Rebiha, R.: Generating invariants for non-linear hybrid systems by linear algebraic methods. In: Cousot, R., Martel, M. (eds.) SAS. LNCS, vol. 6337, pp. 373–389. Springer (2010)
17. Papachristodoulou, A., Prajna, S.: Analysis of non-polynomial systems using the sum of squares decomposition. In: Henrion, D., Garulli, A. (eds.) Positive Polynomials in Control, Lecture Notes in Control and Information Science, vol. 312, pp. 23–43. Springer (2005)
18. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reasoning 41(2), 143–189 (2008)
19. Platzer, A.: Differential-algebraic dynamic logic for differential-algebraic programs. J. Log. Comput. 20(1), 309–352 (2010)
20. Platzer, A.: The complete proof theory of hybrid systems. In: LICS. pp. 541–550. IEEE (2012)
21. Platzer, A.: The structure of differential invariants and differential cut elimination. LMCS 8(4), 1–38 (2012)
22. Platzer, A., Clarke, E.M.: Computing differential invariants of hybrid systems as fixedpoints. In: CAV. pp. 176–189 (2008)
23. Powers, J.E.: Elimination of special functions from differential equations. Communications of the ACM 2(3), 3–4 (Mar 1959)

24. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: HSCC. pp. 477–492. Springer (2004)
25. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation-based abstraction refinement. ACM Transactions on Embedded Computing Systems 6(1) (Feb 2007)
26. Richardson, D.: Some undecidable problems involving elementary functions of a real variable. Journal of Symbolic Logic 33(4), 514–520 (12 1968)
27. Sankaranarayanan, S.: Automatic invariant generation for hybrid systems using ideal fixed points. In: HSCC. pp. 221–230 (2010)
28. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constructing invariants for hybrid systems. FMSD 32(1), 25–55 (2008)
29. Sankaranarayanan, S., Tiwari, A.: Relational abstractions for continuous and hybrid systems. In: CAV. pp. 686–702 (2011)
30. Savageau, M.A., Voit, E.O.: Recasting nonlinear differential equations as S-systems: a canonical nonlinear form . Mathematical Biosciences 87(1), 83 – 115 (1987)
31. Schlomiuk, D.: Algebraic and geometric aspects of the theory of polynomial vector fields. In: Schlomiuk, D. (ed.) Bifurcations and Periodic Orbits of Vector Fields, NATO ASI Series, vol. 408, pp. 429–467. Springer Netherlands (1993)
32. Schlomiuk, D.: Algebraic particular integrals, integrability and the problem of the center. Transactions of the American Mathematical Society 338(2), 799–841 (1993)
33. Strogatz, S.H.: Nonlinear Dynamics and Chaos. Westview Press (1994)
34. Stursberg, O., Kowalewski, S., Hoffmann, I., Preußig, J.: Comparing timed and hybrid automata as approximations of continuous systems. In: Hybrid Systems IV. pp. 361–377 (1996)
35. Tarski, A.: A decision method for elementary algebra and geometry. Bulletin of the American Mathematical Society 59 (1951)
36. Tiwari, A., Khanna, G.: Series of abstractions for hybrid automata. In: Tomlin, C.J., Greenstreet, M.R. (eds.) HSCC. LNCS, vol. 2289, pp. 465–478. Springer (Mar 2002)
37. Tiwari, A.: Abstractions for hybrid systems. FMSD 32(1), 57–83 (2008)
38. Tiwari, A.: Generating box invariants. In: Egerstedt, M., Mishra, B. (eds.) HSCC, LNCS, vol. 4981, pp. 658–661. Springer (2008)
39. Tiwari, A., Khanna, G.: Nonlinear systems: Approximating reach sets. In: HSCC. pp. 600–614 (2004)
40. Wang, T.C., Lall, S., West, M.: Polynomial level-set method for polynomial system reachable set estimation. IEEE Transactions on Automatic Control 58(10), 2508–2521 (2013)
41. Wiggins, S.: Introduction to Applied Nonlinear Dynamical Systems and Chaos. Texts in Applied Mathematics, Springer, second edn. (2003)
42. Wu, Z.: Tangent cone and contingent cone to the intersection of two closed sets. Nonlinear Analysis: Theory, Methods & Applications 73(5), 1203 – 1220 (2010)
43. Zaki, M.H., Tahar, S., Bois, G.: A symbolic approach for the safety verification of continuous systems. In: Proceedings of the International Conference on Computational Sciences. pp. 93–100 (2007)
44. Zhao, H., Zhan, N., Kapur, D.: Synthesizing switching controllers for hybrid systems by generating invariants. In: Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday. pp. 354–373 (2013)