Pegasus: A Framework for Sound Continuous Invariant Generation*

Andrew Sogokon[1,2], Stefan Mitsch[1], Yong Kiam Tan[1], Katherine Cordwell[1], and André Platzer[1,3]

[1] Computer Science Department, Carnegie Mellon University
{asogokon|smitsch|yongkiat|kcordwel|aplatzer}@cs.cmu.edu
[2] School of Informatics, University of Edinburgh
[3] Fakultät für Informatik, Technische Universität München

Abstract. *Continuous invariants* are an important component in deductive verification of hybrid and continuous systems. Just like discrete invariants are used to reason about correctness in discrete systems without unrolling their loops forever, continuous invariants are used to reason about differential equations without having to solve them. *Automatic generation* of continuous invariants remains one of the biggest practical challenges to automation of formal proofs of safety in hybrid systems. There are at present many disparate methods available for generating continuous invariants; however, this wealth of diverse techniques presents a number of challenges, with different methods having different strengths and weaknesses. To address some of these challenges, we develop *Pegasus*: an automatic continuous invariant generator which allows for combinations of various methods, and integrate it with the KeYmaera X theorem prover for hybrid systems. We describe some of the architectural aspects of this integration, comment on its methods and challenges, and present an experimental evaluation on a suite of benchmarks.

Keywords: invariant generation, continuous invariants, ordinary differential equations, theorem proving.

1 Introduction

Safety verification problems for ordinary differential equations (ODEs) are continuous analogues to Hoare triples: the objective is to show that an ODE cannot evolve into a designated set of unsafe states from any of its designated initial states. The role of continuous invariants is therefore analogous to that of inductive invariants for discrete program verification. The problem of automatically

* This material is based upon work supported by the National Science Foundation under Award CNS-1739629 and under Graduate Research Fellowship Grant No. DGE-1252522, by AFOSR under grant number FA9550-16-1-0288, and by the Alexander von Humboldt Foundation. The third author was supported by A*STAR, Singapore. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any sponsoring institution, the U.S. government or any other entity.

generating invariants (also known as *invariant synthesis*) is one of the greatest practical challenges in deductive verification of both continuous and discrete systems. In theory, it is even the *only* challenge for hybrid systems safety [35,39].

The proliferation of published techniques [4,24,29,36,42,44,51,58,60] for continuous invariant generation – targeting various classes of systems, and having different strengths and weaknesses – presents a challenge: ideally, one does not want to be restricted by the limitations of one particular generation technique (or a small family of techniques). Instead, it is far more desirable to have a framework that accommodates existing generation methods, allows for their combination, and is extensible with new methods as they become available. In this paper we (partially) meet the above challenge by developing a single framework which allows us to combine invariant generation methods into novel invariant generation *strategies*. In our work, we are guided by the following considerations:

1. Specialized invariant generation methods are effective only when the problem falls within their domain; their use must therefore be targeted.
2. A combination of invariant generation methods can be more practical than any of the methods considered in isolation. A flexible mechanism for combining these methods is thus highly desirable.
3. Reasoning with automatically generated invariants needs to be done in a *sound* fashion: any deficiencies in the generation procedure must not compromise the final verification result.

Our interest in automatic invariant generation is motivated by the pressing need to enhance the level of proof automation in deductive verification tools for hybrid systems. In this work we target the KeYmaera X theorem prover [15].

Contributions. In this paper we describe the design and implementation of a continuous invariant generator[4] – Pegasus – and its integration into KeYmaera X. We outline some of the basic principles in this coupling, the techniques used to generate invariants, and the mechanism used for combining them into more powerful invariant generation strategies. We evaluate this integration on a set of verification benchmarks – with very promising results.

2 Preliminaries

Ordinary Differential Equations. An n-dimensional autonomous system of first-order ODEs has the form: $\boldsymbol{x}' = f(\boldsymbol{x})$, where $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ is a vector of state variables, $\boldsymbol{x}' = (x_1', \ldots, x_n')$ denotes their time-derivatives, i.e. $\frac{dx_i}{dt}$ for each

[4] **An etymological note on naming conventions.** The KeY [3] prover provided the foundation for developing KeYmaera [37], an interactive theorem prover for hybrid systems. The name KeYmaera was a pun on *Chimera*, a hybrid monster from Classical Greek mythology. The tactic language of the new KeYmaera X prover [15] is called Bellerophon [14] after the hero who defeats the Chimera in the myth. In keeping with an established tradition, the invariant generation framework is called Pegasus because the aid of this winged horse was crucial to Bellerophon in his feat.

$i = 1, \ldots, n$, and $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ specify the RHS of the equations that these time-derivatives must obey along solutions to the ODEs. Geometrically, such a system of ODEs defines a *vector field* $f : \mathbb{R}^n \to \mathbb{R}^n$, associating to each point $\boldsymbol{x} \in \mathbb{R}^n$ the vector $f(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})) \in \mathbb{R}^n$. Whenever the state of the system is required to be confined within some prescribed set of states $Q \subseteq \mathbb{R}^n$, which is known as the *evolution constraint*[5], we will write $\boldsymbol{x}' = f(\boldsymbol{x})\ \&\ Q$. If no evolution constraint is specified, Q is assumed to be $\mathbb{R}^n$. A *solution* to the initial value problem for the system of ODEs $\boldsymbol{x}' = f(\boldsymbol{x})$ with initial value $\boldsymbol{x}_0 \in \mathbb{R}^n$ is a differentiable function $\boldsymbol{x}(\boldsymbol{x}_0, t) : (a, b) \to \mathbb{R}^n$ defined for all times $t \in (a, b) \subseteq \mathbb{R} \cup \{\infty, -\infty\}$ where $a < 0 < b$, and such that $\boldsymbol{x}(\boldsymbol{x}_0, 0) = \boldsymbol{x}_0$ and $\frac{d}{dt}\boldsymbol{x}(\boldsymbol{x}_0, t) = f(\boldsymbol{x}(\boldsymbol{x}_0, t))$ for all $t \in (a, b)$. Given a continuously differentiable function $p : \mathbb{R}^n \to \mathbb{R}$, the Lie derivative of p with respect to vector field f equals the time-derivative of p evaluated along the solutions to the system $\boldsymbol{x}' = f(\boldsymbol{x})$; this Lie derivative is denoted by p' and formally defined as $p' \equiv \sum_{i=1}^{n} \frac{\partial p}{\partial x_i} f_i$.

Semi-algebraic Sets. A set $S \subseteq \mathbb{R}^n$ is *semi-algebraic* iff it is characterized by a finite Boolean combination of polynomial equations and inequalities:

$$\bigvee_{i=1}^{l} \left(\bigwedge_{j=1}^{m_i} p_{ij} < 0 \ \wedge \bigwedge_{j=m_i+1}^{M_i} p_{ij} = 0 \right) \tag{1}$$

where $p_{ij} \in \mathbb{R}[x_1, \ldots, x_n]$ are polynomials. By quantifier elimination, every first-order formula of real arithmetic characterizes a semi-algebraic set and can be put into standard form (1) (see e.g. Mishra [32, §8.6]). With a slight abuse of notation, this paper uses formulas and the sets they characterize interchangeably.

Continuous Invariants in Verification. Safety specifications for ODEs and hybrid dynamical systems can be rigorously verified in formal logics, such as *differential dynamic logic* (dL) [34,35] as implemented in the KeYmaera X proof assistant [15] and *hybrid Hoare logic* [28] as implemented in the HHL prover [61]. The use of appropriate continuous invariants is key to these verification approaches as they allow the complexities of the continuous dynamics to be handled rigorously even for ODEs without closed-form solutions. For example, the dL formula $\mathit{Init} \to [\boldsymbol{x}' = f(\boldsymbol{x})\ \&\ Q]\ \mathit{Safe}$ states that the safety property *Safe* is satisfied throughout the continuous evolution of the system $\boldsymbol{x}' = f(\boldsymbol{x})\ \&\ Q$ whenever the system begins its evolution from a state satisfying *Init*. The main dL reasoning principle for verifying such a safety property is given by the following sound rule of inference, with three premisses above the bar and the conclusion below:

$$\text{(Safety)}\ \frac{\mathit{Init} \to I \qquad I \to [\boldsymbol{x}' = f(\boldsymbol{x})\ \&\ Q]\ I \qquad I \to \mathit{Safe}}{\mathit{Init} \to [\boldsymbol{x}' = f(\boldsymbol{x})\ \&\ Q]\ \mathit{Safe}}\,.$$

In this rule, the first and third premiss respectively state that the initial set *Init* is contained within the set I, and that I lies entirely inside the safe set of

[5] Evolution domain constraints are also called *mode invariants* in the context of hybrid automata. We avoid this name to prevent confusion with generated invariants.

states *Safe*. The second premiss states that I is a *continuous invariant*, i.e. I is maintained throughout the continuous evolution of the system whenever it starts inside I, that is, the following dL formula is true in all states:

$$I \rightarrow [\boldsymbol{x}' = f(\boldsymbol{x}) \;\&\; Q]\; I\,. \tag{2}$$

Thus, the problem of verifying safety properties of ODEs reduces to finding an invariant I that can be *proved* to satisfy all three premisses. Semantically, a continuous invariant can also be defined as follows:

Definition 1 (Continuous invariant). *Given a system $\boldsymbol{x}' = f(\boldsymbol{x})$ & Q, the set $I \subseteq \mathbb{R}^n$ is a continuous invariant iff the following statement holds:*

$$\forall\, \boldsymbol{x}_0 \in I\; \forall t \geq 0 :\; \big((\forall\, \tau \in [0,t] :\; \boldsymbol{x}(\boldsymbol{x}_0,\tau) \in Q) \implies \boldsymbol{x}(\boldsymbol{x}_0,t) \in I\,\big).$$

For any given set of initial states $\mathit{Init} \subseteq \mathbb{R}^n$, a continuous invariant I such that $\mathit{Init} \subseteq I$ provides a *sound over-approximation* of the states reachable by the system from *Init* by following the solutions to the ODEs within the domain constraint Q. Indeed, the exact set of states reachable by a continuous system from *Init* provides the *smallest* such invariant.[6] While the definition above features the solution $\boldsymbol{x}(\boldsymbol{x}_0, t)$, which may not be available explicitly, a crucial advantage afforded by continuous invariants is the possibility of checking whether a given set is a continuous invariant *without computing the solution*, i.e. by working *directly with the ODEs*.

3 Sound Invariant Checking and Generation

The problem of *checking* whether a semi-algebraic set $I \subseteq \mathbb{R}^n$ is a continuous invariant of a polynomial system of ODEs $\boldsymbol{x}' = f(\boldsymbol{x}) \,\&\, Q$ was shown to be *decidable* by Liu, Zhan, and Zhao [29]. This decision procedure, henceforth referred to as LZZ, provides a way of automatically checking continuous invariants (2) by exploiting facts about higher-order Lie derivatives of multivariate polynomials appearing in the syntactic description of I and the Noetherian property of the ring $\mathbb{R}[\boldsymbol{x}]$ [18,29]; its implementation requires an algorithm for constructing Gröbner bases [9], as well as a decision procedure for the universal fragment of real arithmetic [47]. A logical alternative for invariant checking is provided by the complete dL axiomatization for differential equation invariants [39]. Whereas using LZZ results in a yes/no answer to an invariance question (2), dL makes it possible to construct a *formal proof of invariance* from a small set of ODE axioms [39] whenever the property holds (or a refutation when it does not).

3.1 Invariant Generation with Template Enumeration

Given a means to perform invariant checking with real arithmetic, an obvious solution to the invariant generation problem (which has been suggested by

[6] Unfortunately, reachable sets rarely have a simple description as semi-algebraic sets.

numerous authors [29,36,55]) involves the so-called *method of template enumeration*, which yields a theoretically complete semi-algorithm (in the sense that it terminates with a positive answer iff that is possible). All it takes *in theory* is to exhaustively enumerate parametric templates for all real arithmetic formulas describing all semi-algebraic sets, and use a quantifier elimination algorithm (such as CAD [8]) to identify whether choices for the template parameters exist that meet the required arithmetic constraints. While templates make this British Museum Algorithm-like approach more successful than, e.g. exhaustively enumerating all proofs [21], the method is nevertheless quite impractical when used with real quantifier elimination.[7] In practice, invariant generation is usually achieved by using incomplete – but more efficient – generation methods. These methods are numerous and vary considerably in their strengths and limitations, creating a wide spectrum of possible trade-offs in performance, the quality, and the form of invariants that one can generate. Effectively navigating this spectrum is an important practical challenge that we seek to address.

3.2 Soundness: Proof Assistants and Invariant Generation

There are a number of design decisions that can be exercised in how reasoning with continuous invariants is performed within a deductive verification framework. A fundamental design decision is how tightly (i) continuous invariance checking and (ii) continuous invariant generation are to be coupled with the implementation of a proof assistant. This space of design choices is exemplified by the HHL prover and KeYmaera X.

The HHL prover [7,61] implements (i) the LZZ decision procedure for invariant checking and (ii) the method of template enumeration for invariant generation based on real quantifier elimination. From the perspective of the HHL prover, these are *trusted external oracles* for checking the validity of statements about continuous invariance; trusting the output of the HHL prover includes trusting the implementation of its LZZ procedure and the invariant generator.

In contrast, KeYmaera X [15] pursues an LCF-style approach, seeking to minimize the soundness-critical code that needs to be trusted in its output. For continuous invariants, it achieves this by (i) checking invariance within the axiomatic framework of dL (rather than trusting external checking procedures) and (ii) accepting *conjectured invariants* generated from a variety of sources but *separately checking* the result. Invariant checking in KeYmaera X is automatic, which is made possible by the use of specialized proof *tactics* [14]; these additionally allow it to use a variety of other (incomplete, but computationally inexpensive) methods for proving continuous invariance [18].

Remark 1. The difference between these two approaches is broadly analogous to the use of trusted decision procedures in PVS [12] and oracles in HOL [5,63] on the one hand, and proof reconstruction (e.g. in Isabelle [62]) on the other.

[7] Quantifier elimination algorithms used in practice have doubly-exponential time complexity in the number of variables [43]. Template enumeration introduces a fresh variable per monomial coefficient, so the approach quickly hits scalability barriers.

4 Invariant Generation Methods in Pegasus

Pegasus is a continuous invariant generator implemented in the Wolfram Language with an interface accessible through both *Mathematica* and KeYmaera X.[8] When KeYmaera X is faced with a continuous safety verification problem that it is unable to prove directly, it automatically invokes Pegasus to help find an appropriate invariant (if possible). As mentioned earlier, KeYmaera X checks *all* the invariants it is supplied with – *including those provided by Pegasus* (see Fig. 1).

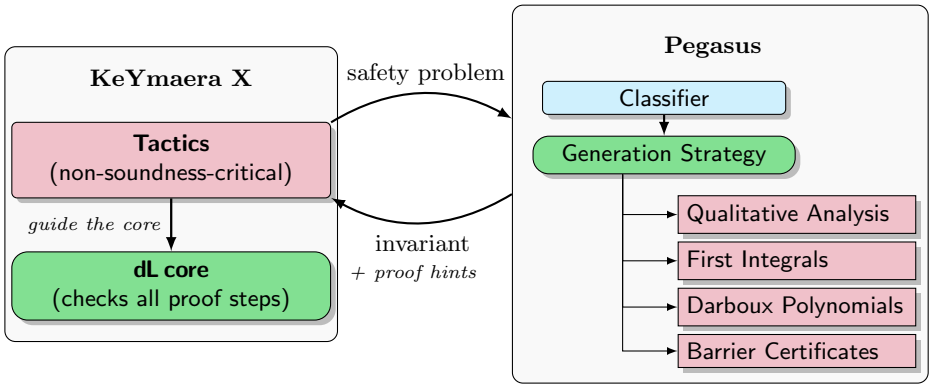


Fig. 1: Sound invariant generation: invariant generator analyses safety problem to provide invariants and proof hints to tactics; the invariants are formally verified to be correct within the soundness-critical dL core

This design ensures that correctness of Pegasus is not integral to the soundness of KeYmaera X. It also presents implementation opportunities for Pegasus:

1. It can freely integrate numerical procedures and heuristic methods while providing *best-effort* guarantees of correctness. Final correctness checks for the generated invariants are left to the purview of KeYmaera X.[9]
2. It records *proof hints* corresponding to various methods that were used to generate continuous invariants. These hints enable KeYmaera X to build more efficient shortcut proofs of continuous invariance [18].

Pegasus currently implements an array of powerful invariant generation methods, which we describe below, beginning with a large family of related methods that are based on *qualitative analysis*, which can be best explained using the machinery of *discrete abstraction* of continuous systems. We first briefly recall the main idea behind this approach.

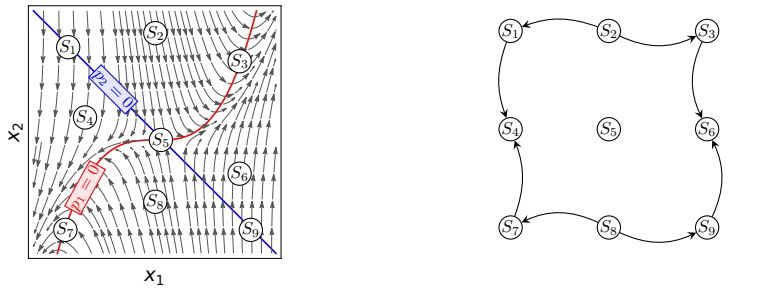
[8] Pegasus (http://pegasus.keymaeraX.org/) is linked to KeYmaera X through the Mathematica interface of KeYmaera X, which translates between the internal data structures of the prover core and the Mathematica data structures.

[9] Naturally, the output from Pegasus can also be checked using a trusted implementation of the LZZ decision procedure before anything is returned. When used with KeYmaera X, though, this additional (soundness-critical) check is unnecessary.

4.1 Exact Discrete Abstraction

Discrete abstraction has been the subject of numerous works [1,57,59]. Briefly, the steps are: (i) discretize the continuous state space of a system by defining *predicates* that correspond to discrete states, (ii) compute a (local) transition relation between the discrete states obtained from the previous step, yielding a discrete transition system which abstracts the behaviour of the original continuous system, and finally (iii) compute reachable sets in the discrete abstraction to obtain an over-approximation of the reachable sets of the original system.

The discrete abstraction is *sound* iff the relation computed in step (ii) has a transition between two discrete states whenever there is a corresponding continuous trajectory of the original system between the two sets corresponding to those discrete states. The abstraction is *exact* iff these are the *only* transitions computed in step (ii). Soundness of the discrete abstraction guarantees that any invariant extracted from the discretization corresponds to an invariant for the original system. Fig. 2 illustrates a discretization of a system of ODEs (Fig. 2a), which results in 9 discrete states in a sound and exact abstraction (Fig. 2b). The state space is discretized using predicates built from sign conditions on polynomials, $p_1, p_2 \in \mathbb{R}[x_1, x_2]$. The discrete states of the abstraction are given by formulas such as $S_1 \equiv p_1 < 0 \wedge p_2 = 0$, $S_2 \equiv p_1 < 0 \wedge p_2 > 0$, and so on.



(a) Discretization with p_1, $p_2 \sim 0$ (b) Sound discrete abstraction

Fig. 2: Discrete abstraction of a two-dimensional system

The ability to construct sound and exact discrete abstractions [51] has an important consequence: if an appropriate semi-algebraic continuous invariant I exists at all, it can always be extracted from a discrete abstraction built from discretizing the state space using sign conditions on the polynomials describing I. The problem of (semi-algebraic) invariant generation therefore reduces to finding appropriate polynomials whose sign conditions can yield suitable discrete abstractions and computing reachable states in these abstractions.

Remark 2. Reachable sets (from the initial states) in discrete abstractions are the smallest invariants with respect to $\subseteq$ (set inclusion) that one can extract. The smallest invariant is the most informative because it allows one to prove the most safety properties, but it may not be the most useful invariant in practice.

In particular, one often wants to work with invariants that have low descriptive complexity *and are easy to prove in the formal proof calculus*. This leads naturally to consider alternative ways of extracting invariants. Pegasus is able to extract reachable sets of discrete abstractions, but favours less costly techniques, such as *differential saturation* [36], which often succeed in quickly extracting more conservative invariants.

Finding "good" polynomials that can abstract the system in useful ways and allow proving properties of interest is generally difficult. While abstraction using predicates that are extracted from the verification problem itself can be surprisingly effective, in certain cases useful predicates may not be syntactically extracted from the problem statement. In order to improve the quality of discrete abstractions, Pegasus employs a separate *classifier*, which extracts features from the verification problem which can then be used to suggest polynomials that are more tailored to the problem at hand. Certain systems have structure that, to a human expert, might suggest an "obvious" choice of good predicates. Below we sketch some basic examples of what is currently possible.

4.2 Targeted Qualitative Analysis

As a motivating example, consider the class of one-dimensional ODEs $x' = f(x)$, where $f \in \mathbb{R}[x]$. A standard way of studying qualitative behaviour in these systems is to inspect the graph of the function $f(x)$ [54]. Fig. 3 illustrates such a graph of $f(x)$, along with a vector field induced by such a system on the real line. By computing the real roots of the polynomial in the right-hand side, i.e the real roots $r_1, \ldots, r_k \in \mathbb{R}$ of $f(x)$, we may form a list of polynomials $x - r_1, \ldots, x - r_k$ that can be used for an *algebraic decomposition* of $\mathbb{R}$ into invariant cells corresponding to real intervals from which an over-approximation of the reachable set can be constructed. Such an algebraic decomposition can be further refined by augmenting the list of polynomials with $x - b_{i1}, \ldots, x - b_{il}$, where $b_{i1}, \ldots, b_{il} \in \mathbb{R}$ are the boundary points of the initial set in the safety specification. From this augmented list, one can exactly construct the *reachable set* of the system by computing the reachable set of the corresponding exact abstraction.

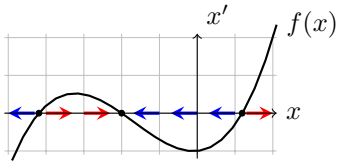


Fig. 3: Qualitative analysis of one-dimensional ODEs $x' = f(x)$

Remark 3. Knowledge of the fact that $x' = f(x)$ is one-dimensional allows one to exploit another useful fact: every one-dimensional system is a *gradient system*, i.e. its motion is generated by a *potential function* $F(x)$ which can be computed directly by integrating $-f(x)$ with respect to x, i.e. $F(x) = -\int f(x)\,dx$. For any $k \in \mathbb{R}$, $F(x) \leq k$ defines a continuous invariant of the system $x' = f(x)$.

In higher dimensions, the behaviour of *linear* systems $\boldsymbol{x}' = A\boldsymbol{x}$ can be studied qualitatively by examining the stability of the fixed point at the origin [2]. The *strongest* algebraic invariants for such systems can be computed for algebraic

initial conditions [44]. Pegasus implements methods targeted at linear systems that takes advantage of facts such as these to suggest useful abstractions from which invariants (not necessarily algebraic) can be extracted. This strategy is similar in spirit to the abstraction methods proposed in the work of Tiwari [56].

Example 1. The linear systems in Fig. 4 exhibit different qualitative behaviours. The invariants (shown in blue), demonstrate unreachability of the unsafe states (shown in red) from the initial states (shown as green discs in Fig. 4).

$$x_1' = -4x_2, \quad x_2' = x_1, \qquad x_1' = 2x_1 - x_2, \quad x_2' = -3x_1 + x_2, \qquad x_1' = -2x_1 + x_2, \quad x_2' = x_1 - 3x_2.$$

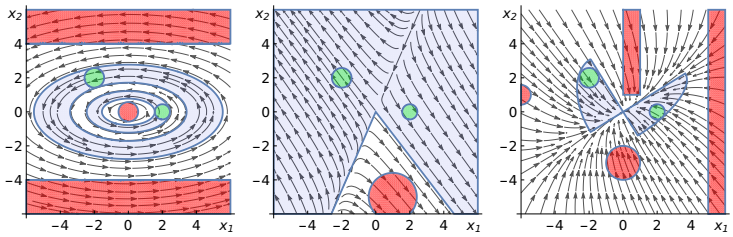


Fig. 4: Automatically generated invariants for linear systems

In the leftmost system, all eigenvalues of the system matrix A are purely imaginary. Pegasus generates annular invariants containing the green discs because trajectories of such systems are always elliptical. For the middle system, the (asymptotic) behaviour of its trajectories is determined by the eigenvectors of its system matrix (eigenvalues are real and of opposite sign [2]). Pegasus uses these eigenvectors to generate two invariant half-planes, one for each green disc. Invariant half-planes are also generated for the rightmost system which is asymptotically *stable* (all real parts of eigenvalues are negative [2]). Pegasus further refines these half-planes with elliptical regions containing the green discs because elliptical regions are invariants for such systems.

4.3 Qualitative Analysis for Non-Linear Systems

General non-linear polynomial systems present a hard class of problems for invariant generation. A number of useful heuristics can be applied to partition the continuous state space, in hopes that the resulting abstraction exhibits a suitable invariant. For example, by factorizing to find polynomials p such that $p = 0$ implies $x_i' = 0$ for some x_i, the flow along the level curve $p = 0$ vanishes in the x_i direction. This information can be used to cheaply approximate the transition relation in the discrete abstraction and to efficiently extract *invariant candidates*. For the non-linear ODE in Fig. 2, the discretization polynomials p_1, p_2 are chosen such that $x_2' = 0$ and $x_1' = 0$ on their respective level curves. This yields a useful discrete abstraction e.g. S_4 is an invariant for the resulting abstraction (Fig. 2b). Other useful sources of polynomials for qualitative analysis of non-linear systems are found in e.g. the summands and irreducible factors of the

right-hand sides of the ODEs, the Lie derivatives of the factors, and physically meaningful quantities such as the *divergence* of the system's vector field.

4.4 General-Purpose Methods

Beyond qualitative analysis, Pegasus implements several general-purpose invariant generation techniques which represent *restricted, but tractable fragments* of the general method of template enumeration. The search for symbolic parameters in these methods is *not* performed using real quantifier elimination, but instead takes place in more tractable theories. We recall these techniques briefly.

First Integrals. The polynomial $p \in \mathbb{R}[\boldsymbol{x}]$ is a *first integral* [19, 2.4.1] of the system $\boldsymbol{x}' = f(\boldsymbol{x})$ iff its Lie derivative p' with respect to the vector field f is the zero polynomial. First integrals are also known as *conserved quantities* because they have the important property that for any $k \in \mathbb{R}$, $p = k$ defines an invariant of the system. For a single first integral p, if one were to use the polynomial $p - k$ to build an abstraction, the abstract state space would not feature any transitions between its states (illustrated in Fig. 5). Thus, one has the freedom to choose value(s) k for which the resulting discrete abstraction suitably partitions the state space. For example, if the initial states lie entirely within $p < k$ and the unsafe ones within $p > k$, then $p < k$ is an invariant separating those sets. Pegasus can search for *all* polynomial first integrals up to a configurable degree bound by solving a system of *linear equations* whose solutions provide the coefficients of the bounded degree polynomial template for the first integral; the solutions are efficiently found using linear algebra [19,49].

Fig. 5: Discrete abstraction with first integrals $p - k$ ($k \in \mathbb{R}$)

Darboux Polynomials. *Darboux polynomials* were first introduced in 1878 [11] to study integrability of polynomial ODEs. Polynomial $p \in \mathbb{R}[\boldsymbol{x}]$ is a *Darboux polynomial* for the system $\boldsymbol{x}' = f(\boldsymbol{x})$ iff $p' = \alpha p$ for some cofactor polynomial $\alpha \in \mathbb{R}[\boldsymbol{x}]$. Like first integrals, discrete abstractions produced with Darboux polynomials result in three states with no transitions between them (as illustrated in Fig. 5, but with $k = 0$). Unlike first integrals, only $p = 0$ is guaranteed to be an invariant of the system. Darboux polynomials have been used for predicate abstraction of continuous systems by Zaki et al. [65], who successfully applied them to verify electrical circuit designs. Automatic generation of Darboux polynomials is an active area of research, with several algorithms proposed within the verification community [24,42,49]. Owing to the importance of Darboux polynomials in the *Prelle-Singer method* [41] for computing elementary closed-form solutions to ODEs, sophisticated algorithms for Darboux polynomial generation were developed earlier in the computer algebra community, e.g. two algorithms were reported by Man [31]. Indeed, we have found these algorithms to be the most practical and implement them in Pegasus.

Barrier Certificates. The method of *barrier certificates* is a popular technique for safety verification of continuous and hybrid systems [40]. Barrier certificates p define an invariant region $p \leq 0$ which separates the initial states (wholly contained within $p \leq 0$) from the unsafe states (wholly contained within $p > 0$) when the Lie derivative p' satisfies certain criteria (e.g. $p' \leq 0$). Generating barrier certificates using the method of template enumeration is possible using both sum-of-squares (SOS) [40] and linear programming (LP) [64] techniques. A number of generalizations of the barrier certificate approach have been developed, which differ in the kinds of conditions that ensure the invariance of $p \leq 0$, e.g. *exponential-type* [25] and *general* barrier certificates [10]. A unified understanding of these generalizations [53] based on classical *comparison systems* [45, Ch II, §3, Ch. IX] leads to a yet more general notion of *vector barrier certificates*. Pegasus is able to search for convex [40], exponential-type [25], and vector barrier certificates [53] using both SOS and LP techniques. However, the resulting barrier certificates often suffer from numerical inaccuracies arising from the use of semi-definite solvers and interior point methods [46]. Pegasus currently uses a simple rounding heuristic on the numerical result and explicitly checks invariance for the resulting (exact) barrier certificate candidates using real quantifier elimination.

5 Strategies for Invariant Generation

The implementation of all the aforementioned invariant generation methods in a single framework is a significant undertaking in itself. The overall goal behind Pegasus, however, is to enable these heterogeneous methods to be effectively deployed and fruitfully combined into *strategies* for invariant generation that are tailored to specific classes of verification problems. Different invariant generation strategies are invoked in Pegasus, depending on the classification of the input problem it receives from the problem *classifier*. In this section, and for the evaluation, we focus on the most challenging and general class of *non-linear* systems in which no further structure is known or assumed beyond the fact that the right-hand sides of the ODEs are polynomials.

The main invariant generation strategy Pegasus uses for general non-linear systems is based on a *differential saturation* procedure [36]. Briefly, the procedure loops through a prescribed *sequence* of invariant generation methods and *successively* attempts to strengthen the domain constraint using invariants found by those methods until the desired safety condition is proved.[10] Notably, this loop allows Pegasus to exploit the strengths of different invariant generation methods, even if it is a priori unclear whether one is better than the other. The precise sequencing of invariant generation methods is also important in this strategy to avoid redundancy. In particular, Pegasus currently orders the methods by

[10] Pegasus analyses problems according to variable dependencies present in their differential equations [36]. For $x_1' = x_1, x_2' = x_1 + x_2$, for example, Pegasus first searches for invariants involving only x_1, before searching for those involving both x_1 and x_2.

computational efficiency, e.g. it first searches for first integrals, followed by Darboux polynomials and barrier certificates. This sequencing allows later (slower) methods to exploit invariants that are quickly generated by earlier methods.

Example 2. The synergy between individual methods exploited by differential saturation is illustrated in Fig. 6 for an example from our benchmarks.

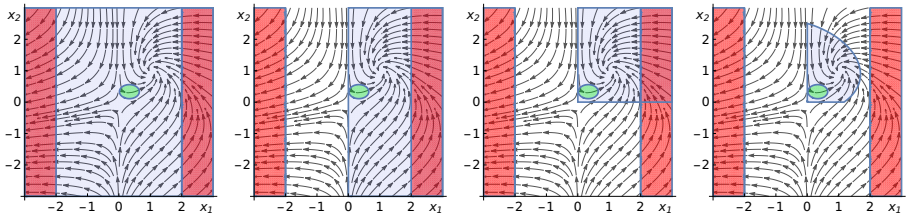


Fig. 6: Invariant synthesis using the differential saturation loop in Pegasus

Initially (leftmost plot), the entire plane (in blue) is under consideration and Pegasus wants to show the safety property that trajectories from the initial states (in green) never reach the unsafe states (in red). In the second plot, Pegasus confines its search to the domain $x_1 > 0$ using the generated Darboux polynomial x_1. In the third plot, using $x_1 > 0$, qualitative analysis finds the invariant $x_2 > 0$ which further confines the evolution domain. Finally (rightmost plot), Pegasus finds a barrier certificate (of polynomial degree 2) that suffices to show the safety property within the strengthened domain (which, by construction, is invariant). The final invariant region *cannot* be directly obtained from a polynomial barrier certificate, but incorporates invariants discovered earlier by other means.

6 Evaluation

We tested Pegasus and its interaction with the ODE proving tactics of KeYmaera X on a benchmark suite of 90 non-linear continuous safety verification problems drawn from the literature [4,10,17,20,22,23,24,29,48,52,64,65]. The suite consists of 53 two-dimensional systems, 11 three-dimensional systems, 12 higher-dimensional ($\geq$4) systems, and 14 *product systems* that were formed by randomly combining pairs of two- and three-dimensional systems. The benchmark was run on commodity hardware: 2.4GHz Intel Core i7 with 16GB memory. We compare the differential saturation strategy to the performance of each invariant generation method in isolation, measuring the duration of generating invariants, duration of checking the generated invariants, and the total proof duration.

Benchmark results for each of the problems are in Fig. 7. Several experimental insights can be drawn from these results: (i) different invariant generation methods generally solve different subsets of the problems, (ii) invariant generation generally dominates overall proof duration although invariant checking becomes more expensive as problem dimension increases, (iii) when multiple methods

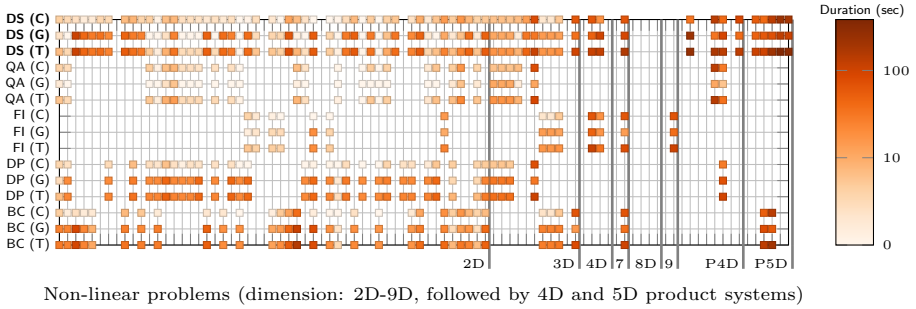


Fig. 7: Comparison of invariant generation methods. Each column represents one benchmark problem and the colour encodes duration (lighter is faster). Empty columns are unsolved. Legend: the combined Differential Saturation (DS) strategy against Qualitative Analysis (QA), First Integrals (FI), Darboux Polynomials (DP), and Barrier Certificates (BC), on total proof duration (T), generation duration (G), and checking duration (C).

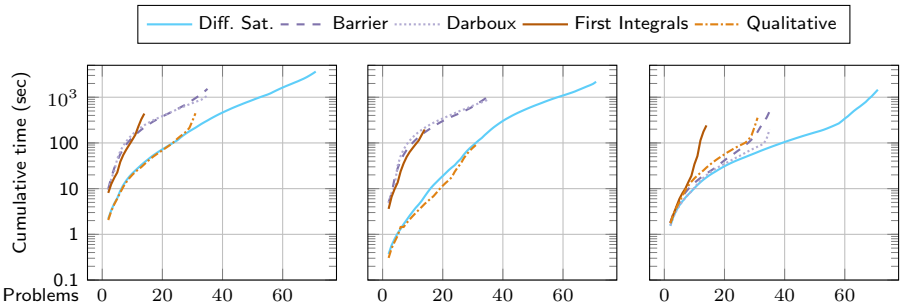
solve a problem, qualitative analysis and first integrals are often quickest, followed by Darboux polynomials and then barrier certificates, (iv) the differential saturation strategy effectively combines invariant generation methods; it solves all but one[11] problem that can be solved by individual methods. It additionally solves 8 problems (of which 5 are product systems) that no individual method solves by itself. Differential saturation is especially effective on product systems because each part of the product may be only solvable using a specific method.

To further evaluate the effectiveness of combining methods by differential saturation, Fig. 8 plots the accumulated duration for solving the fastest n problems. The main insights here are: (i) differential saturation solves the largest number of problems per accumulated time, which means that, despite sequential execution, it often succeeds in trying out the most efficient method first and fails fast when earlier methods fail to apply, (ii) the performance of generating versus checking first integrals is inconclusive and depends on the specific example (see also Fig. 7), (iii) checking barrier certificates and Darboux polynomials is much faster than generating them, and (iv) qualitative analysis is less expensive for generation than other methods.

7 Related Work

Techniques developed for *qualitative simulation* have been applied to prove temporal properties of continuous systems in the work of Shults and Kuipers [50], as well as Loeser, Iwasaki and Fikes [30]. Zhao [66] developed a tool, MAPS, to automatically identify significant features of dynamical systems, such as stability regions, equilibria, and limit cycles. Since our ultimate goal is sound invariant

[11] For this high-dimensional (9D) problem, differential saturation runs out of time trying qualitative analysis methods before attempting to find first integrals.



(a) Total duration (b) Generation (c) Checking

Fig. 8: Cumulative logarithmic time (in seconds) taken to solve the fastest n problems (more problems solved and flatter is better)

generation, we are less interested in a full qualitative analysis of the state space. In the verification community, discrete abstraction of hybrid systems was studied by Alur et al. [1]. The case of systems whose continuous motion is governed by non-linear ODEs was studied in the work of Tiwari and Khanna [57,59]. Tiwari further studied reachability of linear systems [56], using information from real eigenvectors and ideas from qualitative abstraction to generate invariants. Zaki et al. [65] were the first to apply Darboux polynomials to verification of continuous systems using discrete abstraction. Numerous works employ barrier certificates for verification [10,25,40,53,64]. Since we implement many of the above techniques as methods for invariant generation in our framework, our work draws heavily upon ideas developed previously in the verification and hybrid systems communities. Previously [51], we introduced a construction of exact abstractions and applied rudimentary methods from qualitative analysis to compute invariants; in certain ways, our present work also builds on this experience, incorporating some of the techniques as special methods in a more general framework. The coupling between KeYmaera X and Pegasus that we pursue in our work is quite distinct from the use of trusted oracles in the work of Wang et al. [61] (for the HHL prover) and provides a *sound* framework for reasoning with continuous invariants that is significantly less exposed to soundness issues in external tools.

8 Outlook and Challenges

The improvements in continuous invariant generation have a significant impact on the overall proof automation capabilities of KeYmaera X and serve to increase overall system usability and user experience. Improved proof automation will certainly also be useful in future applications of provably correct runtime monitoring frameworks, such as ModelPlex [33], as well as frameworks for generating verified controller executables, such as VeriPhy [6].

Some interesting directions for extending our work include implementation of reachable set computation algorithms for all classes of problems where this is

possible. For instance, semi-algebraic reachable sets may be computed for diagonalizable classes of linear systems with tame eigenvalues [16,26]. The complexity of invariants obtained using these methods may not always make them practical, but they would provide a valuable fallback in cases where simpler invariants cannot be obtained using our currently implemented methods.

A more pressing challenge lies in expanding the collection of safety verification problems for continuous systems. While we have done our best to find compelling examples from the literature, a larger corpus of problems would allow for a more comprehensive empirical evaluation of invariant generation strategies and could reveal interesting new insights that can suggest more effective strategies.

Correctness of decision procedures for real arithmetic is another important challenge. KeYmaera X currently uses Mathematica's implementation of real quantifier elimination to close first-order real arithmetic goals, primarily due to the impressive performance afforded by this implementation (compared to currently existing alternatives). Removing this reliance by efficiently building fully formal proofs of real arithmetic formulas within dL (e.g. through exhibiting appropriate *witnesses* [27,38]) is an important task for the future.

9 Conclusion

Among verification practitioners, the amount of manual effort required for formal verification of hybrid systems is one of the chief criticisms leveled against the use of deductive verification tools. Manually crafting continuous invariants often requires expertise and ingenuity, just like manually selecting support function templates for reachability tools [13], and presents the major practical hurdle in the way of wider industrial adoption of this technology. In this paper, we describe our development of a system designed to help overcome this hurdle by automating the discovery of continuous invariants. To our knowledge, this work represents the first large-scale effort at combining continuous invariant generation methods into a single invariant generation framework and making it possible to create more powerful invariant generation strategies. The approach we pursue is unique in its integration with a theorem prover, which provides formal guarantees that the generated invariants are indeed correct (in the form of dL proofs, *automatically*). The results we observe in our evaluation are highly encouraging and suggest that invariant discovery can be improved considerably, opening many exciting avenues for applications and extensions.

Acknowledgements. The authors would like to thank the anonymous reviewers for their feedback.

References

1. Alur, R., Henzinger, T.A., Lafferriere, G., Pappas, G.J.: Discrete abstractions of hybrid systems. Proceedings of the IEEE **88**(7), 971–984 (2000). https://doi.org/10.1109/5.871304

2. Arrowsmith, D., Place, C.M.: Dynamical Systems: Differential Equations, Maps, and Chaotic Behaviour, vol. 5. CRC Press (1992)
3. Beckert, B., Giese, M., Hähnle, R., Klebanov, V., Rümmer, P., Schlager, S., Schmitt, P.H.: The KeY system 1.0 (deduction component). In: Pfenning, F. (ed.) CADE. LNCS, vol. 4603, pp. 379–384. Springer (2007). https://doi.org/10.1007/978-3-540-73595-3_26
4. Ben Sassi, M.A., Girard, A., Sankaranarayanan, S.: Iterative computation of polyhedral invariants sets for polynomial dynamical systems. In: CDC. pp. 6348–6353. IEEE (2014). https://doi.org/10.1109/CDC.2014.7040384
5. Böhme, S., Weber, T.: Fast LCF-style proof reconstruction for Z3. In: Kaufmann, M., Paulson, L.C. (eds.) ITP. LNCS, vol. 6172, pp. 179–194. Springer (2010). https://doi.org/10.1007/978-3-642-14052-5_14
6. Bohrer, B., Tan, Y.K., Mitsch, S., Myreen, M.O., Platzer, A.: VeriPhy: verified controller executables from verified cyber-physical system models. In: Foster, J.S., Grossman, D. (eds.) PLDI. pp. 617–630. ACM (2018). https://doi.org/10.1145/3192366.3192406
7. Chen, M., Han, X., Tang, T., Wang, S., Yang, M., Zhan, N., Zhao, H., Zou, L.: MARS: A toolchain for modelling, analysis and verification of hybrid systems. In: Hinchey, M.G., Bowen, J.P., Olderog, E. (eds.) Provably Correct Systems, pp. 39–58. NASA Monographs in Systems and Software Engineering, Springer (2017). https://doi.org/10.1007/978-3-319-48628-4_3
8. Collins, G.E.: Quantifier elimination for real closed fields by cylindrical algebraic decompostion, LNCS, vol. 33, pp. 134–183. Springer (1975). https://doi.org/10.1007/3-540-07407-4_17
9. Cox, D.A., Little, J., O'Shea, D.: Ideals, Varieties, and Algorithms. Springer, fourth edn. (2015). https://doi.org/10.1007/978-3-319-16721-3
10. Dai, L., Gan, T., Xia, B., Zhan, N.: Barrier certificates revisited. J. Symb. Comput. **80**, 62–86 (2017). https://doi.org/10.1016/j.jsc.2016.07.010
11. Darboux, J.G.: Mémoire sur les équations différentielles algébriques du premier ordre et du premier degré. Bull. Sci. Math. **2**(1), 151–200 (1878)
12. Denman, W., Muñoz, C.A.: Automated real proving in PVS via MetiTarski. In: Jones, C.B., Pihlajasaari, P., Sun, J. (eds.) FM. LNCS, vol. 8442, pp. 194–199. Springer (2014). https://doi.org/10.1007/978-3-319-06410-9_14
13. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV. LNCS, vol. 6806, pp. 379–395. Springer (2011). https://doi.org/10.1007/978-3-642-22110-1_30
14. Fulton, N., Mitsch, S., Bohrer, B., Platzer, A.: Bellerophon: Tactical theorem proving for hybrid systems. In: Ayala-Rincón, M., Muñoz, C.A. (eds.) ITP. LNCS, vol. 10499, pp. 207–224. Springer (2017). https://doi.org/10.1007/978-3-319-66107-0_14
15. Fulton, N., Mitsch, S., Quesel, J., Völp, M., Platzer, A.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE. LNCS, vol. 9195, pp. 527–538. Springer (2015). https://doi.org/10.1007/978-3-319-21401-6_36
16. Gan, T., Chen, M., Li, Y., Xia, B., Zhan, N.: Reachability analysis for solvable dynamical systems. IEEE Trans. Automat. Contr. **63**(7), 2003–2018 (2018). https://doi.org/10.1109/TAC.2017.2763785
17. Ghorbal, K., Platzer, A.: Characterizing algebraic invariants by differential radical invariants. In: Ábrahám, E., Havelund, K. (eds.) TACAS. LNCS, vol. 8413, pp. 279–294. Springer (2014). https://doi.org/10.1007/978-3-642-54862-8_19

18. Ghorbal, K., Sogokon, A., Platzer, A.: A hierarchy of proof rules for checking positive invariance of algebraic and semi-algebraic sets. Comput. Lang. Syst. Str. **47**, 19–43 (2017). https://doi.org/10.1016/j.cl.2015.11.003
19. Goriely, A.: Integrability and Nonintegrability of Dynamical Systems. World Scientific (2001). https://doi.org/10.1142/3846
20. 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). https://doi.org/10.1007/978-3-540-70545-1_18
21. Herbrand, J.: Recherches sur la théorie de la démonstration. Ph.D. thesis, Université de Paris, Faculté des Sciences (1930)
22. Immler, F., Althoff, M., Chen, X., Fan, C., Frehse, G., Kochdumper, N., Li, Y., Mitra, S., Tomar, M.S., Zamani, M.: ARCH-COMP18 category report: Continuous and hybrid systems with nonlinear dynamics. In: Frehse, G., Althoff, M., Bogomolov, S., Johnson, T.T. (eds.) ARCH. EPiC Series in Computing, vol. 54, pp. 53–70. EasyChair (2018)
23. Kapinski, J., Deshmukh, J.V., Sankaranarayanan, S., Arechiga, N.: Simulation-guided Lyapunov analysis for hybrid dynamical systems. In: Fränzle, M., Lygeros, J. (eds.) HSCC. pp. 133–142. ACM (2014). https://doi.org/10.1145/2562059.2562139
24. Kong, H., Bogomolov, S., Schilling, C., Jiang, Y., Henzinger, T.A.: Safety verification of nonlinear hybrid systems based on invariant clusters. In: Frehse, G., Mitra, S. (eds.) HSCC. pp. 163–172. ACM (2017). https://doi.org/10.1145/3049797.3049814
25. Kong, H., He, F., Song, X., Hung, W.N.N., Gu, M.: Exponential-condition-based barrier certificate generation for safety verification of hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV. LNCS, vol. 8044, pp. 242–257. Springer (2013). https://doi.org/10.1007/978-3-642-39799-8_17
26. Lafferriere, G., Pappas, G.J., Yovine, S.: Symbolic reachability computation for families of linear vector fields. J. Symb. Comput. **32**(3), 231–253 (2001). https://doi.org/10.1006/jsco.2001.0472
27. Li, W., Passmore, G.O., Paulson, L.C.: Deciding univariate polynomial problems using untrusted certificates in Isabelle/HOL. J. Autom. Reas. **62**(1), 69–91 (2019). https://doi.org/10.1007/s10817-017-9424-6
28. Liu, J., Lv, J., Quan, Z., Zhan, N., Zhao, H., Zhou, C., Zou, L.: A calculus for hybrid CSP. In: Ueda, K. (ed.) APLAS. LNCS, vol. 6461, pp. 1–15. Springer (2010). https://doi.org/10.1007/978-3-642-17164-2_1
29. 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). https://doi.org/10.1145/2038642.2038659
30. Loeser, T., Iwasaki, Y., Fikes, R.: Safety verification proofs for physical systems. In: Proc. of the 12th Intl. Workshop on Qualitative Reasoning. pp. 88–95 (1998)
31. Man, Y.: Computing closed form solutions of first order ODEs using the Prelle-Singer procedure. J. Symb. Comput. **16**(5), 423–443 (1993). https://doi.org/10.1006/jsco.1993.1057
32. Mishra, B.: Algorithmic Algebra. Springer (1993). https://doi.org/10.1007/978-1-4612-4344-1
33. Mitsch, S., Platzer, A.: ModelPlex: Verified runtime validation of verified cyber-physical system models. Form. Methods Syst. Des. **49**(1-2), 33–74 (2016). https://doi.org/10.1007/s10703-016-0241-z

34. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. **41**(2), 143–189 (2008)
35. Platzer, A.: The complete proof theory of hybrid systems. In: LICS. pp. 541–550. IEEE (2012). https://doi.org/10.1109/LICS.2012.64
36. Platzer, A., Clarke, E.M.: Computing differential invariants of hybrid systems as fixedpoints. Form. Methods Syst. Des. **35**(1), 98–120 (2009). https://doi.org/10.1007/s10703-009-0079-8
37. Platzer, A., Quesel, J.: KeYmaera: A hybrid theorem prover for hybrid systems (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR. LNCS, vol. 5195, pp. 171–178. Springer (2008). https://doi.org/10.1007/978-3-540-71070-7_15
38. Platzer, A., Quesel, J., Rümmer, P.: Real world verification. In: CADE. LNCS, vol. 5663, pp. 485–501. Springer (2009). https://doi.org/10.1007/978-3-642-02959-2_35
39. Platzer, A., Tan, Y.K.: Differential equation axiomatization: The impressive power of differential ghosts. In: Dawar, A., Grädel, E. (eds.) LICS. pp. 819–828. ACM (2018). https://doi.org/10.1145/3209108.3209147
40. Prajna, S., Jadbabaie, A.: Safety verification of hybrid systems using barrier certificates. In: Alur, R., Pappas, G.J. (eds.) HSCC. LNCS, vol. 2993, pp. 477–492. Springer (2004). https://doi.org/10.1007/978-3-540-24743-2_32
41. Prelle, M.J., Singer, M.F.: Elementary first integrals of differential equations. Transactions of the American Mathematical Society **279**(1), 215–229 (1983)
42. Rebiha, R., Moura, A.V., Matringe, N.: Generating invariants for non-linear hybrid systems. Theor. Comput. Sci. **594**, 180–200 (2015). https://doi.org/10.1016/j.tcs.2015.06.018
43. Renegar, J.: Recent progress on the complexity of the decision problem for the reals. In: Goodman, J.E., Pollack, R., Steiger, W. (eds.) Discrete and Computational Geometry: Papers from the DIMACS Special Year. vol. 6, pp. 287–308. DIMACS/AMS (1990)
44. Rodríguez-Carbonell, E., Tiwari, A.: Generating polynomial invariants for hybrid systems. In: Morari, M., Thiele, L. (eds.) HSCC. LNCS, vol. 3414, pp. 590–605. Springer (2005). https://doi.org/10.1007/978-3-540-31954-2_38
45. Rouche, N., Habets, P., Laloy, M.: Stability Theory by Liapunov's Direct Method, Appl. Math. Sci., vol. 22. Springer (1977). https://doi.org/10.1007/978-1-4684-9362-7
46. Roux, P., Voronin, Y., Sankaranarayanan, S.: Validating numerical semidefinite programming solvers for polynomial invariants. Form. Methods Syst. Des. **53**(2), 286–312 (2018). https://doi.org/10.1007/s10703-017-0302-y
47. Roy, M.F.: Basic algorithms in real algebraic geometry and their complexity: from Sturm's theorem to the existential theory of reals. De Gruyter Expositions in Mathematics **23**, 1–67 (1996)
48. Sankaranarayanan, S.: Automatic invariant generation for hybrid systems using ideal fixed points. In: Johansson, K.H., Yi, W. (eds.) HSCC. pp. 221–230. ACM (2010). https://doi.org/10.1145/1755952.1755984
49. Sankaranarayanan, S., Sipma, H.B., Manna, Z.: Constructing invariants for hybrid systems. Form. Methods Syst. Des. **32**(1), 25–55 (2008). https://doi.org/10.1007/s10703-007-0046-1
50. Shults, B., Kuipers, B.: Proving properties of continuous systems: Qualitative simulation and temporal logic. Artif. Intell. **92**(1-2), 91–129 (1997). https://doi.org/10.1016/S0004-3702(96)00050-1

51. Sogokon, A., Ghorbal, K., Jackson, P.B., Platzer, A.: A method for invariant generation for polynomial continuous systems. In: Jobstmann, B., Leino, K.R.M. (eds.) VMCAI. LNCS, vol. 9583, pp. 268–288. Springer (2016). https://doi.org/10.1007/978-3-662-49122-5_13
52. Sogokon, A., Ghorbal, K., Johnson, T.T.: Non-linear continuous systems for safety verification. In: Frehse, G., Althoff, M. (eds.) ARCH. EPiC Series in Computing, vol. 43, pp. 42–51. EasyChair (2016)
53. Sogokon, A., Ghorbal, K., Tan, Y.K., Platzer, A.: Vector barrier certificates and comparison systems. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E.P. (eds.) FM. LNCS, vol. 10951, pp. 418–437. Springer (2018). https://doi.org/10.1007/978-3-319-95582-7_25
54. Strogatz, S.H.: Nonlinear Dynamics And Chaos. Studies in Nonlinearity, Westview Press (2001)
55. Sturm, T., Tiwari, A.: Verification and synthesis using real quantifier elimination. In: Schost, É., Emiris, I.Z. (eds.) ISSAC. pp. 329–336. ACM (2011). https://doi.org/10.1145/1993886.1993935
56. Tiwari, A.: Approximate reachability for linear systems. In: Maler, O., Pnueli, A. (eds.) HSCC. LNCS, vol. 2623, pp. 514–525. Springer (2003). https://doi.org/10.1007/3-540-36580-X_37
57. Tiwari, A.: Abstractions for hybrid systems. Form. Methods Syst. Des. **32**(1), 57–83 (2008). https://doi.org/10.1007/s10703-007-0044-3
58. Tiwari, A.: Generating box invariants. In: Egerstedt, M., Mishra, B. (eds.) HSCC. LNCS, vol. 4981, pp. 658–661. Springer (2008). https://doi.org/10.1007/978-3-540-78929-1_58
59. Tiwari, A., Khanna, G.: Series of abstractions for hybrid automata. In: Tomlin, C., Greenstreet, M.R. (eds.) HSCC. LNCS, vol. 2289, pp. 465–478. Springer (2002). https://doi.org/10.1007/3-540-45873-5_36
60. Tiwari, A., Khanna, G.: Nonlinear systems: Approximating reach sets. In: Alur, R., Pappas, G.J. (eds.) HSCC. LNCS, vol. 2993, pp. 600–614. Springer (2004). https://doi.org/10.1007/978-3-540-24743-2_40
61. Wang, S., Zhan, N., Zou, L.: An improved HHL prover: An interactive theorem prover for hybrid systems. In: Butler, M.J., Conchon, S., Zaïdi, F. (eds.) ICFEM. LNCS, vol. 9407, pp. 382–399. Springer (2015). https://doi.org/10.1007/978-3-319-25423-4_25
62. Weber, T.: Integrating a SAT solver with an LCF-style theorem prover. Electr. Notes Theor. Comput. Sci. **144**(2), 67–78 (2006). https://doi.org/10.1016/j.entcs.2005.12.007
63. Weber, T.: SMT solvers: new oracles for the HOL theorem prover. STTT **13**(5), 419–429 (2011). https://doi.org/10.1007/s10009-011-0188-8
64. Yang, Z., Huang, C., Chen, X., Lin, W., Liu, Z.: A linear programming relaxation based approach for generating barrier certificates of hybrid systems. In: Fitzgerald, J.S., Heitmeyer, C.L., Gnesi, S., Philippou, A. (eds.) FM. LNCS, vol. 9995, pp. 721–738 (2016). https://doi.org/10.1007/978-3-319-48989-6_44
65. Zaki, M.H., Denman, W., Tahar, S., Bois, G.: Integrating abstraction techniques for formal verification of analog designs. J. Aeros. Comp. Inf. Com. **6**(5), 373–392 (2009). https://doi.org/10.2514/1.44289
66. Zhao, F.: Extracting and representing qualitative behaviors of complex systems in phase space. Artif. Intell. **69**(1-2), 51–92 (1994). https://doi.org/10.1016/0004-3702(94)90078-7