Formal Methods in System Design
https://doi.org/10.1007/s10703-020-00355-z

Pegasus: sound continuous invariant generation

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

Received: 18 April 2020 / Accepted: 11 November 2020

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 having to unroll their loops, 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 the automation of formal proofs of safety for 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.

This material is based upon work supported by the National Science Foundation under Award CNS-1739629 and under Graduate Research Fellowship Grants Nos. DGE1252522 and DGE1745016, by AFOSR under Grant Number FA9550-16-1-0288, by the United States Air Force and DARPA under Contract No. FA8750-18-C-0092, 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.

✉ Andrew Sogokon
asogokon@cs.cmu.edu

Stefan Mitsch
smitsch@cs.cmu.edu

Yong Kiam Tan
yongkiat@cs.cmu.edu

Katherine Cordwell
kcordwel@cs.cmu.edu

André Platzer
aplatzer@cs.cmu.edu

[1] Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA

[2] Present Address: ECS, University of Southampton, Southampton, UK

Published online: 20 January 2021

Keywords Invariant generation · Continuous invariants · Ordinary differential equations · Theorem proving

1 Introduction

Safety verification problems for ordinary differential equations (ODEs) are continuous analogs to Hoare triples: the objective is to show that an ODE cannot evolve out of a designated set of safe states from any of its designated initial states. The role of continuous invariants is broadly analogous to that of inductive invariants for discrete program verification. A continuous invariant is a set of states that can never be left when following the ODE from that set; such an invariant implies safety when it contains all of the initial states and is also a subset of the safe states. The problem of automatically 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 actually the *only* challenge for hybrid systems safety [57].

The proliferation of published techniques [6,39,44,61,68,70,81,89,91] for continuous invariant generation—targeting various classes of systems, and having different strengths and weaknesses—presents a complication: ideally, one does not want to be restricted by the limitations of one particular generation technique (or 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 article 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 and reconfigurable 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 [25].

Contributions. This article is an extended version of the conference paper [84]. The article describes the design and implementation of a continuous invariant generator (Pegasus)[1] and its integration into KeYmaera X. It outlines some of the principles behind this coupling, the techniques used to generate invariants, and the mechanism used for combining them into more powerful invariant generation strategies. An evaluation of this integration on a set

[1] **An etymological note on naming conventions.** The KeY [4] prover provided the foundation for developing KeYmaera [62], an interactive theorem prover for hybrid systems. The name KeYmaera was a pun on the *Chimaera*, a hybrid monster from Classical Greek mythology. The tactic language of the new (aXiomatic) KeYmaera X prover [25] is called Bellerophon [24], after the hero who defeats the Chimaera 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.

of verification benchmarks is presented—with very promising results. The present article extends our previous work [84] with:

1. Extensive coverage of the *methods* for generating continuous invariants employed by Pegasus (Sect. 4), including extended descriptions of several invariant generation methods, as well as new material on *conic abstractions* [7] and on the theory and practice of generating *rational first integrals* for non-linear and linear systems [21,22,30,47,48,77]. The extended article also includes a detailed account of the pitfalls and caveats associated with the various invariant generation and checking methods (Sects. 3–6).
2. New insights on invariant generation *strategies* based on combining various invariant generation methods (Sect. 5), including various configuration options for the *differential saturation* [61] strategy and a new strategy based on *differential divide-and-conquer* [81].
3. An extended benchmark suite with 60 new problems on top of the 90 existing ones (Sect. 6), together with extended experimental evaluation and analysis of various invariant generation strategy configurations.

Structure of this article. Mathematical preliminaries and definitions are reviewed in Sect. 2. Section 3 recalls the problem of continuous invariant *checking* and describes our architecture for *sound* invariant checking and generation. Sections 4 and 5 describe some of the methods employed by Pegasus for generating continuous invariants, along with mechanisms for their combination. Section 6 presents an empirical evaluation of our integration with KeYmaera X on a suite of verification benchmarks. Section 7 reviews related work and Sect. 8 discusses the outlook and possible further extensions. Section 9 ends with a summary and concluding remarks. Coloured versions of all figures are available online.

2 Preliminaries

Ordinary Differential Equations. An n-dimensional autonomous system of first-order ODEs has the form: $\mathbf{x}' = f(\mathbf{x})$, where $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ is a vector of state variables, $\mathbf{x}' = (x_1', \ldots, x_n')$ denotes their time-derivatives, i.e. $\frac{dx_i}{dt}$ for each $i = 1, \ldots, n$, and $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x}))$ specify the right-hand side (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 $\mathbf{x} \in \mathbb{R}^n$ the vector $f(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})) \in \mathbb{R}^n$ specifying in which direction the continuous system evolves at $\mathbf{x}$. Whenever the state of the system is required to be confined within some prescribed set of states $Q \subseteq \mathbb{R}^n$, called its *evolution domain constraint*,[2] we will write $\mathbf{x}' = f(\mathbf{x})$ & Q. If no evolution domain constraint is specified, then $Q = \mathbb{R}^n$. A *solution* to the initial value problem for the system of ODEs $\mathbf{x}' = f(\mathbf{x})$ with initial value $\mathbf{x}_0 \in \mathbb{R}^n$ is a differentiable function $\mathbf{x}(\mathbf{x}_0, t) : (a, b) \to \mathbb{R}^n$ defined on some *maximal interval of existence* $(a, b) \subseteq \mathbb{R} \cup \{\infty, -\infty\}$ where $a < 0 < b$, and such that $\mathbf{x}(\mathbf{x}_0, 0) = \mathbf{x}_0$ and $\frac{d}{dt}\mathbf{x}(\mathbf{x}_0, t) = f(\mathbf{x}(\mathbf{x}_0, t))$ for all $t \in (a, b)$. The *Lie derivative* of a continuously differentiable function $p : \mathbb{R}^n \to \mathbb{R}$ with respect to vector field f is defined as $p' \equiv \sum_{i=1}^n \frac{\partial p}{\partial x_i} f_i$

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

and equals the time-derivative of p evaluated along the solutions to the system $\mathbf{x}' = f(\mathbf{x})$ [60,64].

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 $p_{ij} \in \mathbb{R}[x_1, \ldots, x_n]$ (i.e. p_{ij} are multivariate polynomials in the indeterminates $x_1, \ldots, x_n$, with real coefficients). By quantifier elimination, every first-order formula of real arithmetic characterizes a semi-algebraic set and can be expressed in the form (1), see e.g. Mishra [49, §8.6]. With an abuse of notation, this article uses formulas and the sets they characterize interchangeably.

Continuous Invariants in Verification. Safety specifications for ODEs and hybrid systems can be rigorously verified in formal logics, such as *differential dynamic logic* (dL) [56,59,60] as implemented in the KeYmaera X proof assistant [25] and *hybrid Hoare logic* [43] as implemented in the HHL prover [92]. 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 $\mathit{Init} \rightarrow [\mathbf{x}' = f(\mathbf{x}) \ \& \ Q]\ \mathit{Safe}$ states that the safety property *Safe* is satisfied throughout the continuous evolution of the system $\mathbf{x}' = f(\mathbf{x}) \ \& \ Q$ whenever the system begins its evolution from a state satisfying *Init*. The invariant reasoning principle for verifying such a safety property is given by the following sound rule of inference in dL, with three premisses above the bar and the conclusion below:

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

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 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:

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

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 $\mathbf{x}' = f(\mathbf{x}) \ \& \ Q$, the set $I \subseteq \mathbb{R}^n$ is a continuous invariant iff the following statement holds:[3]

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

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 evolution domain constraint Q. Indeed, the exact set of states reachable by a continuous system from *Init* provides the *smallest* such

[3] To simplify notation, $\forall\, t \geq 0$ is implicitly assumed to quantify over all times $t \geq 0$ in the maximal interval of existence of the ODE solution from $\mathbf{x}_0$, i.e., where $\mathbf{x}(\mathbf{x}_0, t)$ is defined.

invariant.[4] While Def. 1 above features the solution $\mathbf{x}(\mathbf{x}_0, t)$, 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 $\mathbf{x}' = f(\mathbf{x}) \,\&\, Q$ was shown to be *decidable* by Liu, Zhan, and Zhao [44]. 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 $\mathbb{R}[\mathbf{x}]$ [28,44]; its implementation requires an algorithm for constructing Gröbner bases [15], as well as a decision procedure for the universal fragment of real arithmetic [73]. A logical alternative for invariant checking is provided by the complete dL axiomatization for differential equation invariants [64]. 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 [64] whenever the property holds (or a refutation whenever it does not).

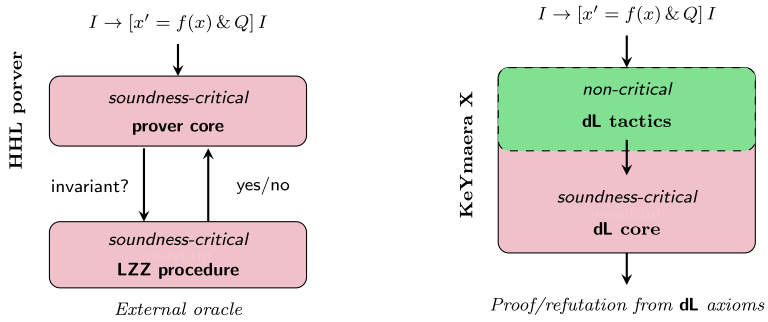
3.1 Invariant generation with template enumeration

Given a means to perform invariant checking with real arithmetic, an obvious solution to the invariant generation problem (which has been suggested by numerous authors [44,61,86]) involves the *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 with the given templates. A template is a parametric formula, such as

$$a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 < 0 \wedge b_0 + b_1 x + b_2 y \geq 0 \ ,$$

composed from polynomials in the state variables (in this example x, y) with symbolic coefficients (here a_0,a_1,a_2,a_3,a_4,a_5 and b_0,b_1,b_2), which are interpreted over the reals. All it takes *in theory* is to exhaustively enumerate parametric templates matching *all* real arithmetic formulas describing all semi-algebraic sets, and use a quantifier elimination algorithm (such as CAD [14]) 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 [34], the method is nevertheless quite impractical for the resulting real arithmetic [58]. To appreciate why, let us only remark that quantifier elimination algorithms for real arithmetic used in practice have doubly-exponential time complexity in the number of variables [69]. Template enumeration treats every monomial coefficient in the template as a fresh variable, leading to exponentially many real arithmetic variables, which makes this approach highly unscalable. In practice, invariant generation is achieved by using incomplete—but considerably 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 this article seeks to address.

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



(a) PVS-style (b) LCF-style

Fig. 1 Alternative prover architectures for *checking* conjectured continuous invariants, i.e. formulas for the form $I \to [\mathbf{x}' = f(\mathbf{x})\ \&\ Q]\ I$

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 the prover. This space of design choices is exemplified by the HHL prover and the KeYmaera X prover.

The HHL prover [12,92] implements (i) the LZZ decision procedure for invariant checking and (ii) the method of template enumeration for invariant generation based on real quantifier elimination and Gröbner bases. 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 (and any arithmetic tool either of them use).

In contrast, KeYmaera X [25] pursues an LCF-style approach, seeking to minimize the soundness-critical code that needs to be trusted in its output [51]. 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 [64], which is made possible by the use of specialized proof *tactics* [24]; these additionally allow it to use a variety of other (incomplete, but computationally inexpensive) methods for proving continuous invariance [28].

Remark 1 The difference between these two approaches (Fig. 1) is broadly analogous to the use of trusted decision procedures in PVS [18] and oracles in HOL [8,94] on the one hand, and LCF-style proof reconstruction (e.g. in Isabelle [93]) on the other.

Remark 2 KeYmaera X also supports witness checking for the universal fragment of real arithmetic [63] resulting from ODE invariance checking [64]. In theory, this leads to a complete LCF-style approach, but in practice, the performance of real arithmetic witness generation is only competitive with second-tier quantifier elimination [63].

3.3 Syntactic representation of invariants

A subtle issue that arises when interfacing with provers like KeYmaera X or the HHL prover is which terms can be *syntactically* represented in the prover. The choice of representation limits the kinds of invariants that can be described (or generated), but it is an important consideration for computational efficiency and soundness purposes. For example, Noetherian functions support a sound and complete axiomatization of invariants in dL [64] but can lead to undecidable arithmetic. Rational functions and roots could be supported [9] but would increase the complexity of the required symbolic computations. For decidability of the invariance and arithmetic questions, this article only considers semi-algebraic invariants, i.e., those built from polynomials as in (1).

A similar issue arises even when restricted to polynomial terms. Naïvely, for maximum flexibility, one would like to describe invariants using polynomials $p \in \mathbb{R}[x]$ that have arbitrary real-valued coefficients. In practice though, only *computable* subfields K of $\mathbb{R}$ can be effectively represented and used on a computer. Thus, any computational tool must necessarily work with polynomials $p \in K[x]$ over some choice of representation for the field of coefficients K. Real algebraic numbers $K = \bar{\mathbb{Q}}$ would work as coefficients, but they increase the complexity of symbolic computations due to the added need to work with polynomial ideal arithmetic for coefficients and can also lead to some subtleties with the non-differentiability of the resulting root function itself [9]. On the other extreme, floating point numbers are computationally efficient but they do not form a field, and would also cause numerical errors that make it harder to obtain sound and exact answers in the end. For these reasons, KeYmaera X works with polynomials $p \in \mathbb{Q}[x]$ that have rational coefficients.[5] This results in fast evaluations and symbolic computations, and a reasonable (although nontrivial) complexity for the resulting real arithmetic validity decision problem. Many invariant generation techniques described in this article are fairly general and agnostic to the precise choice of field K. Thus, the rest of this article elides this subtlety and describes the invariant generation algorithms over $p \in \mathbb{R}[x]$, i.e., with $\mathbb{R}$ as the coefficient field.

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.[6] 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). KeYmaera X checks *all* the invariants it is supplied with—*including those provided by Pegasus* (see Fig. 2). This design ensures that any correctness issues in Pegasus cannot compromise the soundness of KeYmaera X. It also presents implementation opportunities:

[5] In practice, some generation methods may need to internally use floating point arithmetic when interfacing with numerical solvers, but Pegasus then applies rounding procedures to obtain polynomials with rational coefficients.

[6] 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.

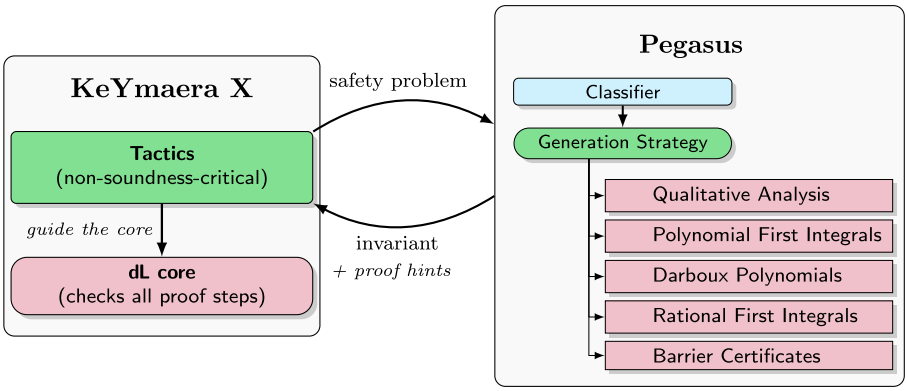


Fig. 2 Sound invariant generation: invariant generator analyzes safety problem to provide invariants and proof hints to tactics; the invariants are formally verified to be correct within the soundness-critical dL core

1. Pegasus 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.[7]
2. Pegasus records *proof hints* corresponding to the various methods that were used to generate continuous invariants. These hints enable KeYmaera X to build more efficient shortcut proofs of continuous invariance [28].

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.

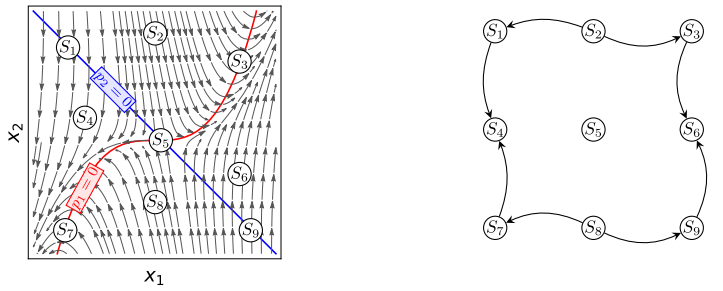
4.1 Exact discrete abstraction

Discrete abstraction is the subject of numerous works [2,88,90]. 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 behavior 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 continuous system.

A discrete abstraction is *sound* iff the relation computed in step (ii) has a transition between two discrete states whenever there is a corresponding trajectory of the original continuous system between the two neighboring 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. Exactness implies that no invariants are lost that are representable in the abstraction at all.

Figure 3 illustrates a discretization of a system of ODEs (Fig. 3a), which results in 9 discrete states in a sound and exact abstraction (Fig. 3b). The state space is discretized using predicates

[7] 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.



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

Fig. 3 Discrete abstraction of a two-dimensional system

built from sign conditions on polynomials, $p_1, p_2 \in \mathbb{R}[x_1, x_2]$.[8] 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. The question whether there should be a discrete transition from S_1 to S_2 in the abstraction may be equivalently cast as the following question: is S_1 a continuous invariant of the system $\mathbf{x}' = f(\mathbf{x})$ under evolution domain constraint $S_1 \vee S_2$, i.e. is the following dL formula valid?

$$S_1 \rightarrow [\mathbf{x}' = f(\mathbf{x}) \,\&\, S_1 \vee S_2]\, S_1 \ .$$

This question can be answered with a decision procedure such as LZZ or formally proved/disproved using dL, as discussed in Sect. 3. If S_1 is a continuous invariant under this evolution domain constraint, then there are no states satisfying S_1 from which the system continuously evolves into a state satisfying S_2 along a trajectory that remains within the union $S_1 \cup S_2$ and thus there should not be a transition from S_1 to S_2 if the discrete abstraction is to be exact; on the other hand, if S_1 is not a continuous invariant, then there must be such a transition if the abstraction is to be sound.

The ability to construct sound and exact discrete abstractions [81] 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 3 Reachable sets (from the initial states) in discrete abstractions are the smallest invariants with respect to $\subseteq$ (set inclusion) that are representable in that abstraction. 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* [61], which often succeed in more 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

[8] *Sign conditions* on a polynomial p are atomic formulas $p < 0$, $p = 0$, and $p > 0$.

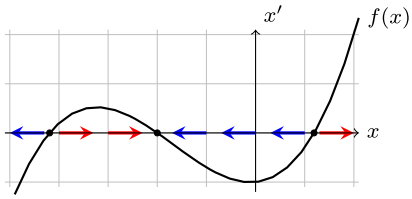


Fig. 4 Qualitative analysis of one-dimensional ODEs $x' = f(x)$

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 behavior in these systems is to inspect the graph of the function $f(x)$ [85]. Figure 4 illustrates such a graph of $f(x)$, along with a vector field induced by such a system on the real line.

The ODE $x' = f(x)$ is at an *equilibrium* without any motion at points where $f(x) = 0$. 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 subregions 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 $x - b_1, \ldots, x - b_l$, where $b_1, \ldots, b_l \in \mathbb{R}$ 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.

Remark 4 If $x' = f(x)$ is one-dimensional, one can 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 one-dimensional system $x' = f(x)$.

In higher dimensions, the behavior of *linear* systems $\mathbf{x}' = A\mathbf{x}$ with a constant coefficient matrix A can be studied qualitatively by examining the eigenvalues and eigenvectors[9] of the matrix A [3]. Pegasus implements methods targeted at linear systems that take advantage of facts such as these to suggest useful abstractions from which invariants can be extracted. The current strategy is similar in spirit to the abstraction methods proposed in the work of Tiwari [87], and works by computing linear forms describing the invariant half-spaces in the state space of linear systems. Briefly, whenever the system matrix A has a real eigenvalue $\lambda \in \mathbb{R}$, by considering an eigenvector $\mathbf{v}$ of the *transpose* matrix A^T, which is associated with the

[9] A vector $\mathbf{v} \in \mathbb{R}^n$ is an *eigenvector* for *eigenvalue* $\lambda \in \mathbb{C}$ of matrix $A \in \mathbb{R}^{n \times n}$ iff $A\mathbf{v} = \lambda\mathbf{v}$. In direction $\mathbf{v}$, the ODE $\mathbf{x}' = A\mathbf{x}$, thus, converges to 0 if $\lambda < 0$ or diverges if $\lambda > 0$.

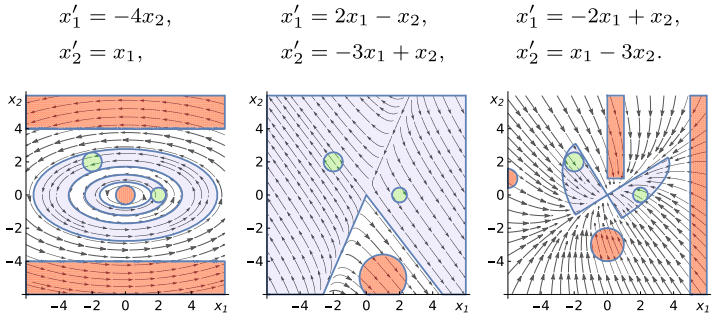


Fig. 5 Automatically generated invariants for linear systems

eigenvalue λ (recall that the eigenvalues of square matrices A and A^T are the same), one may construct the linear form $p = \mathbf{v}^T\mathbf{x}$, which has the property that [87, § 2]:

$$p' = \mathbf{v}^T\mathbf{x}' = \mathbf{v}^T A\mathbf{x} = (A\mathbf{v})^T\mathbf{x} = (\lambda\mathbf{v})^T\mathbf{x} = \lambda p \ .$$

Such linear forms correspond to a special case of so-called *Darboux polynomials*, which will be described in more detail in Sect. 4.4.2 and have the property that $p > 0$, $p = 0$, and $p < 0$ define invariant regions in state space (the fact that λ is a real number also allows us to construct invariants $p \leq k$, where k is an appropriately chosen offset depending on the sign of λ).

Additionally, when all the eigenvalues of the system matrix A have strictly negative real parts, the origin $\mathbf{0}$ is asymptotically stable and one may construct a *Lyapunov function* (see [38, Ch. 3], [80, Ch. 3]) for the linear system by solving the *Lyapunov equation* $A^T P + PA = Q$ where Q is some given negative-definite matrix,[10] and the solution P is positive-definite (see [80, Ch. 3, §3.5]); the quadratic Lyapunov function V for the stable system is given by $V(\mathbf{x}) = \mathbf{x}^T P\mathbf{x}$. Every sub-level set $V \leq k$ defines a continuous invariant of the system; Fig. 5 (right) illustrates the kind of invariants that can be obtained by using Lyapunov functions together with invariant half-planes to perform abstraction of linear systems.

Example 1 The linear systems in Fig. 5 exhibit different qualitative behaviors. The invariants (shown in blue), demonstrate unreachability of the unsafe states (shown in red) from the initial states (shown as green disks in Fig. 5). In the leftmost system, all eigenvalues of the system matrix A are purely imaginary. Pegasus generates annular invariants containing the green disks because trajectories of such systems are always elliptical. For the middle system, the (asymptotic) behavior of its trajectories is determined by the eigenvectors of its system matrix (eigenvalues are real and of opposite sign [3]). 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 [3]). Pegasus further refines these half-planes with suitable elliptical regions containing the green disks because elliptical regions are invariants for such systems.

⚠ *In textbook examples of linear systems, one usually finds matrices with eigenvalues and eigenvectors that can be described using rational numbers. However, the situation is not always that nice in practice: eigenvectors of matrices will often feature* irrational *components,*

[10] An $n \times n$ matrix Q is *negative-definite* if it is symmetric, i.e. $Q = Q^T$, and $\mathbf{x}^T Q\mathbf{x} < 0$ for all $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$; a symmetric matrix P is *positive-definite* if $\mathbf{x}^T P\mathbf{x} > 0$ for all $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$.

which in the case of the example above leads to invariant half-planes described by linear polynomials with irrational coefficients. It is therefore important to have the means of working with irrational real numbers in the invariant generator and the prover.

In special cases when the verification problem features a purely *algebraic initial set*, the strongest algebraic invariants for linear systems (i.e. the smallest continuous invariants that can be described by polynomial equalities $p = 0$) can be computed following the method of Rodríguez-Carbonell and Tiwari [70], which we implement in Pegasus.

Remark 5 Bogomolov et al. [7] introduced a technique called *conic abstractions* that combines discrete abstraction of affine systems with an associated reachability analysis method. It is particularly powerful for systems $\mathbf{x}' = A\mathbf{x}$ in which the matrix A is diagonalizable,[11] where the authors' experiments suggest it outperforms other tools for linear reachability analysis, like SpaceEx [23]. The eponymous idea behind the method is to partition state space into a number of regions (i.e., *cones*), so that within each cone the change in angle of the vector field (i.e., the *twisting*) is bounded by a tunable parameter θ. Given any point in the vector field, then, this construction gives a known range of possible slopes for the vector at that point. This is useful information for the subsequent reachability analysis—instead of simply computing the transition relation between neighboring cones, as in Sect. 4.1, the algorithm [7] uses the twisting information to determine what portions of each cone is potentially reachable from an initial set. We experimented with the conic abstraction method in a limited setting: bounded linear 2-dimensional systems. The major obstacle inhibiting a complete implementation is that Mathematica's native support for polyhedra computations does not quite meet the demands of the algorithm. Our limited implementation is not able to return an exact invariant region—instead, we produce promising visualizations of the invariant generated for two examples from Fig. 5 (see Fig. 6).[12] With better support for polyhedra computations, this could be an exciting direction for future implementation by interfacing Pegasus with the Parma Polyhedra Library.

4.3 Qualitative analysis for non-linear systems

General non-linear polynomial systems of ODEs present a hard class of problems for invariant generation. A number of useful heuristics can be applied to partition the continuous state space of these systems, in the hope that the resulting abstraction exhibits a suitable invariant. For example, factorizing the RHS of a differential equation $x_i' = f_i(x)$ yields a set of irreducible polynomial factors $p_1, \ldots, p_k$ such that $f_i = \prod_{j=1}^{k} p_j$, which implies that the flow along the curves $p_j = 0$ 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. 3, 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. 3b). 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

[11] The matrix A is *diagonalizable* iff it can be written as $A = PDP^{-1}$ for some invertible matrix P and diagonal matrix D.

[12] The conic abstractions approach does not work directly with the leftmost example from Fig. 5 because the example's system matrix has purely imaginary eigenvalues and is consequently not diagonalizable (a key requirement for termination of the approach [7]).

$x_1' = 2x_1 - x_2,$
$x_2' = -3x_1 + x_2.$
(Fig. 5 middle example)

$x_1' = -2x_1 + x_2,$
$x_2' = x_1 - 3x_2.$
(Fig. 5 right example)

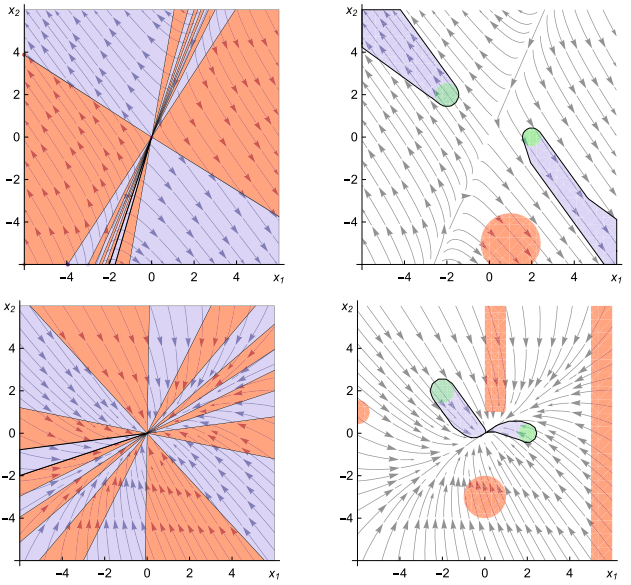


Fig. 6 A visualization of our implementation of the conic abstractions method (each example is shown row-wise). The left figures show the generated conic partition into 20 cones (alternating red and blue colors). The right figures show the reachable set computation (in blue) from the same green initial sets as in Fig. 5. These reachable sets, which are invariant sets, suffice to show that the ODE never reaches any unsafe states (in red). The method automatically produces finer partitions of the state space (using more cones) when the direction of the vector field changes more drastically. The top partition concentrates several cones around its unstable manifold [13,85] (the line $y = \frac{1}{6}(1 + \sqrt{13})x$), while the bottom partition has more evenly spaced out cones

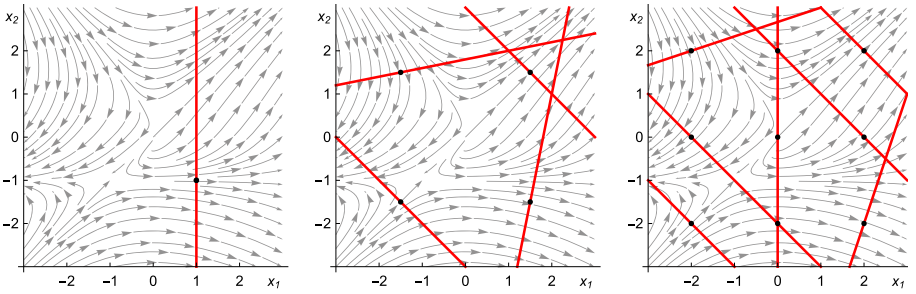


Fig. 7 Abstractions using locally transverse linear forms (shown as red lines) generated from a grid of points (in black)

of the factors, and physically meaningful quantities such as the *divergence* of the system's vector field.

Locally transverse linear forms. A simple geometric idea can sometimes help generate linear polynomials for abstraction. For a system of ODEs $\mathbf{x}' = f(\mathbf{x})$, which may be non-linear, and a regular point $\mathbf{x}_0 \in \mathbb{R}^n$ with $f(\mathbf{x}_0) \neq \mathbf{0}$, one may construct the linear form $f(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0)$, which has the property that its zero set is locally *transverse* to the vector field near $\mathbf{x}_0$.[13] With a

[13] By continuity of $f(\cdot)$, the vectors $f(\mathbf{x})$ are sufficiently close to $f(\mathbf{x}_0)$ for points $\mathbf{x}$ in a small neighborhood around $\mathbf{x}_0$. Therefore, all ODE solutions in this neighborhood can only cross $f(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0)$ in the same direction as $f(\mathbf{x}_0)$.

Fig. 8 Discrete abstraction with first integral $p - k$ ($k \in \mathbb{R}$)

sufficiently fine partitioning using regular points, one has a good chance of finding invariant regions in the abstraction. In problems where the evolution domain constraint describes a bounded set, it is possible to obtain useful abstractions by choosing a finite number of regular points $\mathbf{x}_0$ within the set and partitioning the constraint with the corresponding locally transverse linear forms (as illustrated in Fig. 7). Of course, choosing "good" points is the main problem in this method; one possibility is to use evenly-spaced points forming a grid covering the evolution domain constraint.

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.

4.4.1 Polynomial first integrals

A polynomial $p \in \mathbb{R}[\mathbf{x}]$ is a *first integral* [31, 2.4.1] (also see [65, § 23]) of the system $\mathbf{x}' = f(\mathbf{x})$ 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 an important property: their value never changes along the solutions to ODEs; that is to say, for any $k \in \mathbb{R}$, $p = k$ is an invariant of the system.[14] For a single first integral p, if one were to use (the signs of) the polynomial $p - k$ to build an abstraction, the abstract state space would not feature any transitions between its states (illustrated in Fig. 8). Thus, one has the freedom to choose values 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. This is known as the *method of undetermined coefficients*; we illustrate the main steps of the method in the following example.

Example 2 (Kasner's equations) Consider the non-linear system of ODEs describing a special case of Einstein's gravitational equations [37]

$$\begin{aligned} x_1' &= x_2 x_3 - x_1^2 \,, \\ x_2' &= x_3 x_1 - x_2^2 \,, \\ x_3' &= x_1 x_2 - x_3^2 \,, \end{aligned}$$

and a polynomial template of maximum degree 2 in the state variables x_1, x_2, x_3:

$$p_{\mathbf{a},2} = a_0 + a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_1^2 + a_5 x_1 x_2 + a_6 x_1 x_3 + a_7 x_2^2 + a_8 x_2 x_3 + a_9 x_3^2 \,.$$

[14] Strictly speaking, first integrals and conserved quantities are *not the same*: a first integral may only be considered a conserved quantity in regions where it is defined. In this case, however, polynomial functions are defined everywhere in $\mathbb{R}^n$ and the two notions coincide.

Computing the Lie derivative of this template with respect to the system, i.e. $(p_{\mathbf{a},2})' = \frac{\partial p_{\mathbf{a},2}}{\partial x_1} x_1' + \frac{\partial p_{\mathbf{a},2}}{\partial x_2} x_2' + \frac{\partial p_{\mathbf{a},2}}{\partial x_3} x_3'$ gives a degree 3 parametric polynomial:

$$\begin{aligned}(p_{\mathbf{a},2})' = &-a_1 x_1^2 + a_3 x_1 x_2 + a_2 x_1 x_3 - a_2 x_2^2 + a_1 x_2 x_3 - a_3 x_3^2 - 2a_4 x_1^3 \\ &+ (a_6 - a_5) x_1^2 x_2 + (a_5 - a_6) x_1^2 x_3 + (a_8 - a_5) x_1 x_2^2 \\ &+ (2a_4 + 2a_7 + 2a_9) x_1 x_2 x_3 + (a_8 - a_6) x_1 x_3^2 - 2a_7 x_2^3 \\ &+ (a_5 - a_8) x_2^2 x_3 + (a_6 - a_8) x_2 x_3^2 - 2a_9 x_3^3 \ .\end{aligned}$$

In order to find a first integral, one is required to solve the equation $(p_{\mathbf{a},2})' = 0$, but a polynomial is 0 precisely when all of its coefficients are 0. Thus, by equating all coefficients of the Lie derivative to 0, finding a first integral reduces to solving a *linear system of equations* over the symbolic coefficients $a_0, \ldots, a_9$:

$$\begin{aligned}&-a_1 = 0, a_3 = 0, a_2 = 0, -a_2 = 0, a_1 = 0, -a_3 = 0, -2a_4 = 0, (a_6 - a_5) = 0, \\ &(a_5 - a_6) = 0, (a_8 - a_5) = 0, (2a_4 + 2a_7 + 2a_9) = 0, (a_8 - a_6) = 0, \\ &-2a_7 = 0, (a_5 - a_8) = 0, (a_6 - a_8) = 0, -2a_9 = 0 \ .\end{aligned}$$

Solutions are efficiently found using linear algebra [31, § 2.4.1]. In this example, a non-trivial solution yields the polynomial first integral $x_1 x_2 + x_1 x_3 + x_2 x_3$. Moreover, *all* first integrals of degree (up to) two provide concrete instances of the coefficients a and so must correspond to a solution of these equations.

When a polynomial first integral p is computed, one has the freedom of choosing its initial value, which is guaranteed to remain invariant throughout the evolution of the system. In the above example, one may choose any real number k and partition the state space into invariant regions defined by the sign conditions on the polynomial $x_1 x_2 + x_1 x_3 + x_2 x_3 - k$. To obtain a tight over-approximation of the reachable set from the initial set of states given in the verification problem, one may choose k by maximizing and minimizing the value of the first integral p on the initial set of states within the evolution domain constraint, i.e., one may search for the real values (if they exist):

$$k_{\max} = \max_{\mathbf{x} \in \mathit{Init} \cap Q} p(\mathbf{x}) \,, \qquad k_{\min} = \min_{\mathbf{x} \in \mathit{Init} \cap Q} p(\mathbf{x}) \ .$$

If finite values $k_{\max}$ and $k_{\min}$ can be obtained, one may generate a continuous invariant $k_{\min} \leq p \wedge p \leq k_{\max}$ (or just $p = k_{\min}$ if $k_{\max} = k_{\min}$).

⚠ *Maximizing/minimizing multivariate polynomials subject to semi-algebraic constraints often leads to irrational and real algebraic numbers as exact maxima/minima. Numerical algorithms will yield values that are near-optimal, which may require them to be increased/decreased by some amount before a genuine invariant is constructed as described above.*

⚠ *The set $\mathit{Init} \cap Q$ may have* multiple connected components, *and tighter invariants may be obtained from first integrals when the value k is optimized subject to each connected component separately. A cheap way to approximate the connected components is to normalize $\mathit{Init} \wedge Q$ to disjunctive normal form and consider each disjunct as a separate component.*

If more than one independent first integral for a system is found, one may construct finer abstractions and generate tighter invariants over-approximating the reachable set. A particularly interesting case is when an n-dimensional system of ODEs has $n - 1$ *functionally independent* algebraic first integrals: such a system is said to be *algebraically integrable*

[31,52]. In such a system, given a state $\mathbf{x}_0 \in \mathbb{R}^n$, one may evaluate the first integrals $p_1, p_2, \ldots, p_{n-1}$ at that state to obtain a continuous invariant given by

$$p_1 = p_1(\mathbf{x}_0) \wedge p_2 = p_2(\mathbf{x}_0) \wedge \cdots \wedge p_{n-1} = p_{n-1}(\mathbf{x}_0) \ .$$

If the first integrals are functionally independent, i.e. when the matrix

$$[\nabla p_1 \ \nabla p_2 \ \cdots \ \nabla p_{n-1}]$$

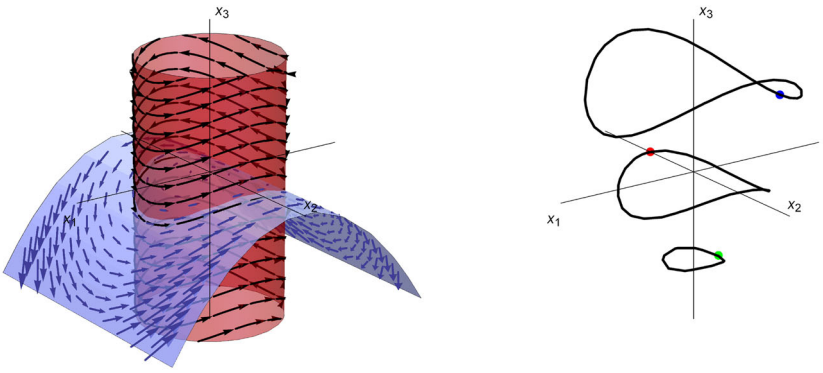
whose columns are formed by the gradients $\nabla p_i \equiv \left(\frac{\partial p_i}{\partial x_1}, \frac{\partial p_i}{\partial x_2}, \ldots, \frac{\partial p_i}{\partial x_n}\right)^T$ has *full rank* at $\mathbf{x}_0$ (i.e. when the vectors ∇p_i evaluated at $\mathbf{x}_0$ are linearly independent, see e.g. [52]), the resulting conjunctive formula (locally) describes a 1-dimensional invariant curve in n-dimensional state space and provides the tightest possible algebraic invariant containing $\mathbf{x}_0$.

Example 3 (Algebraic integrability) Consider the non-linear system

$$\begin{aligned} x_1' &= -x_2 \ , \\ x_2' &= x_1 \ , \\ x_3' &= x_1 x_2 \ . \end{aligned}$$

Using a quadratic polynomial template $p_{\mathbf{a},2}$ and solving the linear system of equations corresponding to the equality $(p_{\mathbf{a},2})' = 0$ as described in Example 2, one obtains the first integrals $p_1 = x_1^2 + x_2^2$ and $p_2 = x_1^2 + x_3$. The level sets described by $p_1 = k_1$ and $p_2 = k_2$ are invariants for any $k_1, k_2 \in \mathbb{R}$. A level set of a first integral corresponds to an invariant surface to which the system's vector field is tangent at all points on the surface. For example, Fig. 9a illustrates two invariant surfaces of this system, which are described by $p_1 = 1$ (corresponding to the red cylinder) and $p_2 = 0$ (corresponding to the blue inverted parabolic surface). Taking $\mathbf{x}_0 = (0, 1, 0)^T$, one can easily check that the first integrals p_1 and p_2 are functionally independent:

$$[\nabla p_1 \ \nabla p_2] = \begin{bmatrix} \frac{\partial p_1}{x_1} & \frac{\partial p_2}{x_1} \\ \frac{\partial p_1}{x_2} & \frac{\partial p_2}{x_2} \\ \frac{\partial p_1}{x_3} & \frac{\partial p_2}{x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 & 2x_1 \\ 2x_2 & 0 \\ 0 & 1 \end{bmatrix} , \text{ which at } \mathbf{x}_0 \text{ becomes } \begin{bmatrix} 0 & 0 \\ 2 & 0 \\ 0 & 1 \end{bmatrix}$$



(a) Invariant surfaces $p_1 = 1$ and $p_2 = 0$ (b) Invariant curves through points

Fig. 9 Invariant level sets of two independent first integrals (left) whose intersections define invariant curves (right)

and is full rank. Since the system of ODEs is 3-dimensional and we have $2 = 3-1$ independent algebraic first integrals, this system is algebraically integrable.[15] Intuitively, the invariant level surfaces of first integrals will intersect transversally (i.e. will not be tangent) if the first integrals are functionally independent. Each such intersection results in an invariant which is of lower dimension: for example, the intersection of the two invariant surfaces in Fig. 9a (i.e. $p_1 = 1 \wedge p_2 = 0$) corresponds to the invariant *space curve*—a one-dimensional object in 3-dimensional space—which contains the point $\mathbf{x}_0 = (0, 1, 0)^T$, as illustrated in Fig. 9b by the middle curve going through the red point $\mathbf{x}_0$.[16] One may choose other points $\mathbf{x}_0$ and use them to evaluate the first integrals $p_1(\mathbf{x}_0)$ and $p_2(\mathbf{x}_0)$, from which one can construct other invariant curves described by $p_1 = p_1(\mathbf{x}_0) \wedge p_2 = p_2(\mathbf{x}_0)$ (as in Fig. 9b).

4.4.2 Darboux polynomials

Darboux polynomials were first introduced in 1878 [17] to study integrability of polynomial ODEs. A polynomial $p \in \mathbb{R}[\mathbf{x}]$ is said to be a *Darboux polynomial* for the system $\mathbf{x}' = f(\mathbf{x})$ if and only if $p' = \alpha p$ for some polynomial $\alpha \in \mathbb{R}[\mathbf{x}]$, which is known as the *cofactor* of p. Like first integrals, discrete abstractions produced with Darboux polynomials result in three states with no transitions between them (as illustrated in Fig. 8, 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. [97], who successfully applied them to verify electrical circuit designs.

The problem of generating Darboux polynomials is generally far more difficult than that of generating polynomial first integrals (which represent the special case of Darboux polynomials where the cofactor α is 0 in the equation $p' = \alpha p$). A modification of the method of undetermined coefficients described in the previous section can likewise be applied to search for Darboux polynomials. However, in order to apply this method, one is required to provide a polynomial template for *both* the Darboux polynomial *and* for its cofactor. Whenever one has a polynomial system of ODEs $\mathbf{x}' = f(\mathbf{x})$ in which the maximum polynomial degree of the components $f_1, f_2, \ldots, f_n$ of f is some $r \geq 0$, then the maximum possible degree of the Lie derivative (w.r.t. this system) of a polynomial p of maximum degree d is given by $d + r - 1$. Consequently, to search for a Darboux polynomial of maximum degree d, the maximum degree of the cofactor α in the equation $p' = \alpha p$ that one needs to consider is given by $r - 1$. To apply the method of undetermined coefficients, one requires a template $p_{\mathbf{a},d}$ for the Darboux polynomial and a separate template $\alpha_{\mathbf{b},r-1}$ for the cofactor. The equation to be solved is the following:

$$(p_{\mathbf{a},d})' - \alpha_{\mathbf{b},r-1} p_{\mathbf{a},d} = 0 \ .$$

By expanding the polynomial on the left-hand side and equating each of its monomial coefficients to 0, one obtains a system of equations in the symbolic parameters $\mathbf{a}, \mathbf{b}$; however, while this system is linear in the parameter variables $\mathbf{a}$ and $\mathbf{b}$ considered separately, it is a *non-linear system of equations* in $\mathbf{a}, \mathbf{b}$ simultaneously. In practice, solving such a non-linear system is far more computationally expensive than solving the linear systems for polynomial first integrals; the naïve method of undetermined coefficients does not provide a practically appealing solution for Darboux polynomial generation.

[15] In this example the first integrals are polynomial functions, but in general algebraic first integrals need *not* be polynomial: e.g. they may be rational functions, as we shall see in Sect. 4.4.3.

[16] In fact, for this particular example this closed curve represents the *periodic orbit* (see e.g. [13]) of the system through the point $\mathbf{x}_0$.

Fortunately, automatic generation of Darboux polynomials is an active area of research, owing largely to their importance as a crucial component in the *Prelle-Singer method* [67] for computing elementary closed-form solutions to ODEs. In order to implement the Prelle-Singer method, more sophisticated algorithms for Darboux polynomial generation have been developed in the computer algebra community, e.g. two algorithms were reported by Man [47]. Indeed, in our experiments we have found the algorithms `ps_1` and `new_ps_1` in Man [47] to be much more practical and implement them in Pegasus.

Remark 6 We remark also that several algorithms for generating (what are essentially) Darboux polynomials have more recently been developed within the verification community [39,68,76]. However, our experience with some of these procedures has been less positive. The method in [68] was in practice found to be very inefficient and *incomplete*, i.e. unable in general to find all the Darboux polynomials matching a given polynomial template; the technique described in [39] is significantly faster but is likewise incomplete.

⚠ *Determining whether an arbitrary system of polynomial ODEs possesses a Darboux polynomial (and finding a bound on its degree if it does) remains an open problem* [98, §4.1].

4.4.3 Rational first integrals

Beyond polynomial functions, a much larger class of algebraic conserved quantities is that of *rational first integrals*; these are first integrals represented by *rational functions*, i.e. functions of the form $\frac{a}{b}$, where a, b are polynomials and $b \neq 0$. Searching for this kind of first integral is (unsurprisingly) more difficult than is the case with polynomials; however, it is made possible by exploiting an idea from the seminal work of Darboux (see e.g. Schlomiuk [77]): multiple Darboux polynomials can be combined to construct a rational first integral.

Theorem 1 *Let $p_1, p_2, \ldots, p_k$ be Darboux polynomials for the system $\mathbf{x}' = f(\mathbf{x})$, with $p_i' = \alpha_i p_i$, where α_i is some polynomial cofactor for each $i = 1, \ldots, k$. If*

$$\lambda_1\alpha_1 + \lambda_2\alpha_2 + \cdots + \lambda_k\alpha_k = 0 \tag{3}$$

has a non-trivial integer solution, i.e. $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_k) \in \mathbb{Z}^k \setminus \{\mathbf{0}\}$, then the system has a rational first integral $r_{\boldsymbol{\lambda}} \in \mathbb{R}(\mathbf{x})$ given by the product

$$r_{\boldsymbol{\lambda}} = p_1^{\lambda_1} p_2^{\lambda_2} \cdots p_k^{\lambda_k}.$$

Proof By applying the product rule to compute the Lie derivative $r_{\boldsymbol{\lambda}}'$, we get

$$\begin{aligned}(p_1^{\lambda_1} p_2^{\lambda_2} \cdots p_k^{\lambda_k})' &= \lambda_1 p_1^{\lambda_1-1} p_1'(p_2^{\lambda_2} \cdots p_k^{\lambda_k}) + \cdots + \lambda_k p_k^{\lambda_k-1} p_k'(p_1^{\lambda_1} \cdots p_{k-1}^{\lambda_{k-1}}) \\ &= \lambda_1 p_1^{\lambda_1-1}\alpha_1 p_1(p_2^{\lambda_2} \cdots p_k^{\lambda_k}) + \cdots + \lambda_k p_k^{\lambda_k-1}\alpha_k p_k(p_1^{\lambda_1} \cdots p_{k-1}^{\lambda_{k-1}}) \\ &= (\lambda_1\alpha_1 + \lambda_2\alpha_2 + \cdots + \lambda_k\alpha_k)(p_1^{\lambda_1} p_2^{\lambda_2} \cdots p_k^{\lambda_k}).\end{aligned}$$

From equation (3) it follows that $r_{\boldsymbol{\lambda}}' = 0$ and $r_{\boldsymbol{\lambda}}$ is therefore a first integral. □

Remark 7 Obviously, if the solution to (3) is such that $\boldsymbol{\lambda} \in \mathbb{Z}^k_{\geq 0}$, then the first integral is polynomial; at least one negative component in $\boldsymbol{\lambda}$ is therefore required in order to construct a non-polynomial rational first integral. We also note that one may search for rational solutions to (3), i.e. $\boldsymbol{\lambda} \in \mathbb{Q}^k$, which will in general result in first integrals featuring radicals. Any such first integral can be turned into a rational first integral by raising it to an integer power

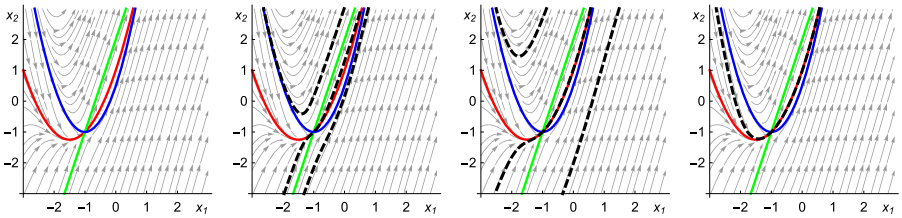


Fig. 10 Rational first integral $r_{\boldsymbol{\lambda}}$ constructed from three Darboux polynomials. Zero sets of the three Darboux polynomials shown in solid green, blue and red. Invariant level sets of the rational first integral shown in dashed black for values $r_\lambda = \frac{1}{10}, 1, -2$, respectively

corresponding to the least common multiple of the denominators of the rational numbers $\lambda_1, \ldots, \lambda_k$. In general, $\lambda_1, \ldots, \lambda_k$ need not be rational or even real numbers in order for the construction given in Theorem 1 to work; however, irrational solutions lead to first integrals that are not rational functions.

In light of the above theorem, a straightforward procedure for generating rational first integrals (which has previously been suggested by Man [48]) involves (i) generating Darboux polynomials $p_1, p_2 \ldots, p_k$ for the system $\mathbf{x}' = f(\mathbf{x})$, e.g. using an implementation of Man's algorithms [47], and (ii) finding integer (or rational) solutions to the linear system of equations (3) in Theorem 1. If the coefficients of the cofactors $\alpha_1, \alpha_2, \ldots, \alpha_k$ in equation (3) are all rational numbers, the problem reduces to solving a system of linear Diophantine equations, for which there exist polynomial-time algorithms. If a rational first integral $r_{\boldsymbol{\lambda}} = \frac{a}{b}$ is found, then $\frac{a}{b} = l$ defines an invariant hypersurface for any choice of $l \in \mathbb{R}$, assuming $b \neq 0$; rewriting this, we get that $a - lb = 0$ is invariant for any $l \in \mathbb{R}$ (when $b \neq 0$).

Example 4 Consider the following non-linear system of ODEs [22]:

$$x_1' = 6x_1^4 + 27x_1^3 - 9x_1^2x_2 + 42x_1^2 - 24x_1x_2 + 21x_1 + 4x_2^2 - 7x_2 + 4\,,$$
$$x_2' = 18x_1^4 + 99x_1^3 - 39x_1^2x_2 + 150x_1^2 + 2x_1x_2^2 - 80x_1x_2 + 71x_1 + 12x_2^2 - 21x_2 + 12\,.$$

Using our implementation of Man's algorithm [47], we obtain the following list of Darboux polynomials in under one second of computation time:

$$(p_1, p_2, p_3) = \left(x_1 - \frac{x_2}{3} + \frac{2}{3},\ x_1^2 + 2x_1 - \frac{2x_2}{3} + \frac{1}{3},\ x_1^2 + 3x_1 - x_2 + 1 \right)\,.$$

Solving equation (3) in Theorem 1, we obtain the solution $(\lambda_1, \lambda_2, \lambda_3) = (2, 1, -1)$, from which we obtain the rational first integral (illustrated in Fig. 10)

$$r_{\boldsymbol{\lambda}} = p_1^2 p_2^1 p_3^{-1} = \frac{(x_1 - \frac{x_2}{3} + \frac{2}{3})^2 (x_1^2 + 2x_1 - \frac{2x_2}{3} + \frac{1}{3})}{x_1^2 + 3x_1 - x_2 + 1}\,.$$

Remark 8 Before attempting to search for algebraic first integrals (whether polynomials or rational functions) it is helpful to have static criteria that determine whether such first integrals can arise in a given system of ODEs. Criteria for non-existence of various kinds of first integrals have been studied by numerous authors (notably by Poincaré [98, §7.2]) and typically make use of the linearization $\mathbf{x}' = A\mathbf{x}$ of the system $\mathbf{x}' = f(\mathbf{x})$ around a point of equilibrium

(i.e. a point $\mathbf{x}_*$ where $f(\mathbf{x}_*) = \mathbf{0}$). In particular, a sufficient criterion for non-existence of rational first integrals in non-linear systems of ODEs is given by Shi [78, Theorem 1]; it requires that the eigenvalues $\lambda_1, \ldots, \lambda_n$ of the matrix A are such that $k_1\lambda_1 + \cdots + k_n\lambda_n = 0$ does not have a non-trivial integer solution $(k_1, \ldots, k_n) \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$. A similar criterion, which furthermore accounts for repeated eigenvalues, is given by Goriely [31, Ch. 5, Prop. 5.5].

Combining Darboux Polynomials and Rational First Integrals. As a first hint of its flexibility for combining invariant generation methods, Pegasus implements rational first integral generation by combining several ideas described thus far in Sect. 4 as follows. This flexibility is further exploited in the discussion of *strategies* in Sect. 5.

1. Compute a list of Darboux polynomials $p_1, \ldots, p_k$ of some maximum polynomial degree d using generation methods from Sect. 4.4.2.
2. Abstract the state space into sign invariant cells using those polynomials, e.g., $S_1 \equiv p_1 < 0 \wedge p_2 = 0$, $S_2 \equiv p_1 < 0 \wedge p_2 > 0$, $S_3 \equiv p_1 < 0 \wedge p_2 < 0$, etc., as described in Sect. 4.1. Notably, the resulting abstraction has no transitions between its discrete states, as illustrated in Fig. 8.
3. Prune away those invariant cells that do not intersect the initial set of states, e.g., delete S_1 if $\mathit{Init} \cap S_1 = \emptyset$ since S_1 is then unreachable. Similarly, prune away cells that do not intersect the unsafe set, e.g., delete S_2 if $\mathit{Unsafe} \cap S_2 = \emptyset$ because no initial states in S_2 can reach the unsafe set.
4. The remaining unpruned *conflict cells*, say S_3, define new invariant generation *subproblems*, where the original evolution domain constraint Q is restricted to $Q \wedge S_3$. Each of the Darboux polynomials are sign-invariant in these cells; moreover, those Darboux polynomials that are sign-definite (either strictly positive or negative) in each cell, e.g. p_1, p_2 with evolution domain constraint $p_1 < 0 \wedge p_2 > 0$ for S_3, can be used to compute rational first integrals $r_{\boldsymbol{\lambda}}$ (following Theorem 1). The denominator of $r_{\boldsymbol{\lambda}}$ is guaranteed to be a product of (powers of) sign-definite polynomials so these rational functions are always defined within each conflict cell.
5. Using their respective rational first integrals r_λ, refine each conflict cell by maximizing and minimizing the values of $r_{\boldsymbol{\lambda}}$ to obtain invariant sub-level sets $k_{\min} \leq r_{\boldsymbol{\lambda}} \wedge r_{\boldsymbol{\lambda}} \leq k_{\max}$ over the initial set (restricted to that cell), as described in Sect. 4.4.1.
6. If conflict cells remain, increase the polynomial degree d and go to step *1*.

Rational First Integrals of Linear Systems. In the case of linear systems of ODEs $\mathbf{x}' = A\mathbf{x}$, more efficient methods exist that allow us to *directly construct* rational first integrals from the eigenvalues and eigenvectors of the system matrix A. These explicit constructions are described, e.g. in the work of Gorbuzov & Pranevich [30] and Falconi & Llibre [21]; in Pegasus, we implement and deploy the former techniques [30].

It is instructive to compare the results obtained by Lafferriere, Pappas and Yovine [42] (which state that semi-algebraic reachable sets of linear ODEs $\mathbf{x}' = A\mathbf{x}$ can be constructed from semi-algebraic initial sets in cases when A is diagonalizable and all of its eigenvalues are rational) to essentially analogous results independently obtained in the study of integrability of linear systems. For instance, [30, Property 1.1] gives a sufficient condition for algebraic integrability which states that a linear system $\mathbf{x}' = A\mathbf{x}$ has a *basis* of rational first integrals (i.e. is algebraically integrable) if all the eigenvalues of A are rational and of multiplicity 1. Indeed, such a basis of rational first integrals enables one to construct reachable sets described by polynomials.

4.4.4 Barrier certificates

The method of *barrier certificates* is a popular Lyapunov-like technique for safety verification of continuous and hybrid systems [66]. Barrier certificates are differentiable functions p that 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 $p > 0$). In order to ensure continuous invariance of the region defined by $p \leq 0$, the Lie derivative p' of the barrier certificate needs to satisfy certain criteria; differences in these criteria give rise to a number of variations of barrier certificates in the literature. The original work by Prajna and Jadbabaie [66] introduced *convex barrier certificates*, which employ the differential inequality $p' \leq 0$ to guarantee invariance of $p \leq 0$ under the flow of the system. Later work by Kong et al. [40] introduced so-called *exponential-type barrier certificates*, which provide a generalization employing the differential inequality $p' \leq \lambda p$, where $\lambda \in \mathbb{R}$; this was generalized further yet in the work of Dai et al. [16], who introduced *general barrier certificates* employing the differential inequality $p' \leq \omega(p)$, where ω is a specifically crafted scalar *function* to guarantee invariance of $p \leq 0$. All of the above developments are fundamentally based on the classical notion of *comparison systems* [71, Ch II, §3, Ch. IX] in the theory of ODEs. A unified understanding of these generalizations is described in prior work [83], which introduces a further generalization of the barrier certificate framework: *vector barrier certificates*, employing multidimensional comparison systems in a way analogous to vector Lyapunov functions introduced by Bellman [5].

Barrier certificates are practically interesting because one may apply the method of undetermined coefficients to automatically search for them using tractable techniques: either sum-of-squares programming (SOS) [66] or linear programming (LP) [95]. Pegasus is able to search for convex [66], exponential-type [40], and vector barrier certificates [83] using both SOS and LP techniques. However, the resulting barrier certificates often suffer from numerical inaccuracies arising from the use of semidefinite solvers and interior point methods [72]. 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. An example of a barrier certificate generation technique implemented in Pegasus, and an illustration of its numerical issues is given next.

Example 5 Consider the safety verification problem illustrated in Fig. 11 (left). The task is to generate an invariant showing that ODE solutions starting within the initial set *Init* (in green) do not enter the unsafe set *Unsafe* (in red). A candidate continuous invariant $p \leq 0$ (shown in blue in Fig. 11, left) is found using numerical barrier certificate generation techniques.

The (exponential-type) barrier certificate p is generated from a polynomial template $p_{\mathbf{a},d}$ of degree d over variables x, y, by solving (and then substituting) for appropriate concrete values of the template coefficients $\mathbf{a}$. For clarity below, the notation $p_{\mathbf{a},d}$ is used in steps where the generation algorithm produces constraints on the coefficients $\mathbf{a}$, while p always refers to the final, generated barrier certificate. Logically, it suffices to find real values for $\mathbf{a}$ so that the following formulas are simultaneously valid:

$$Init \rightarrow p_{\mathbf{a},d} \leq 0\,, \tag{4}$$

$$Unsafe \rightarrow p_{\mathbf{a},d} > 0\,, \tag{5}$$

$$(p_{\mathbf{a},d})' \leq \lambda p_{\mathbf{a},d}\ . \tag{6}$$

Constraints (4) and (5) ensure that the generated barrier separates the initial set from the unsafe set, e.g., in Fig. 11 (left) the green initial region is wholly contained in the blue candidate invariant region $p \leq 0$, while the red unsafe region lies entirely outside. Constraint (6)

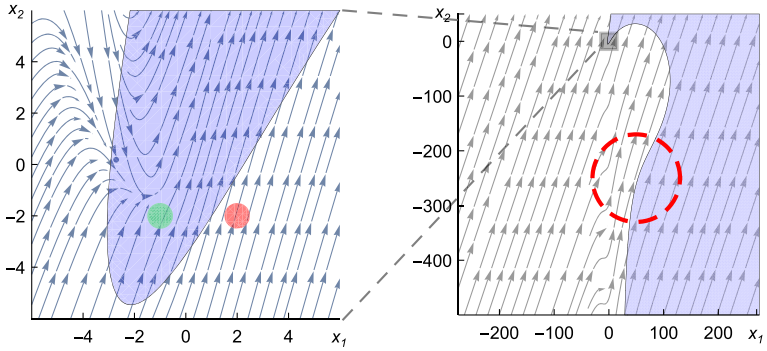


Fig. 11 (Left) A candidate invariant generated using numerical barrier certificates (in blue) for the safety verification problem of showing that solutions from the green initial state never reach the red unsafe states. (Right) A zoomed out view of the safety verification problem, showing that the candidate invariant is, in fact, *not* an invariant of the ODE because some states can exit the invariant (highlighted with a dashed red circle)

ensures that the sub-level set $p \leq 0$ is a continuous invariant, intuitively, the vector field points "inwards" along the boundary of $p \leq 0$ (blue region in Fig. 11), so it is impossible to flow from within $p \leq 0$ to $p > 0$. A more general version of these constraints, and a soundness proof, is available elsewhere [40].

Sum-of-squares (SOS) programming [53] provides a tractable way of solving for the coefficients $\mathbf{a}$. Suppose that *Init*, *Unsafe* are described with polynomial inequalities $Init \equiv \bigwedge_{i=1}^{a} I_i \geq 0$, $Unsafe \equiv \bigwedge_{i=1}^{b} U_i \geq 0$. Inequalities (4)–(6) are respectively implied by the following SOS inequalities, where $\varepsilon > 0$ is a small positive constant and $\sigma_{I_i}, \sigma_{U_i}$ are template SOS polynomials [53]:

$$-p_{\mathbf{a},d} - \sum_{i=1}^{a} \sigma_{I_i} I_i \geq 0\,, \tag{7}$$

$$p_{\mathbf{a},d} - \sum_{i=1}^{b} \sigma_{U_i} U_i - \varepsilon \geq 0\,, \tag{8}$$

$$\lambda p_{\mathbf{a},d} - (p_{\mathbf{a},d})' \geq 0\ . \tag{9}$$

Sum-of-squares solvers, such as SOSTOOLS [53], witness the inequalities (7)–(9) by finding an SOS representation for their left-hand side. For example, a set of polynomials $g_1, \ldots, g_n$ satisfying the polynomial identity $-p_{\mathbf{a},d} - \sum_{i=1}^{a} \sigma_{I_i} I_i = \sum_{i=1}^{n} g_i^2$ proves (7) because the RHS of this inequality is a sum-of-squares, which is non-negative. These polynomial identities are found efficiently by semidefinite programming [55], which is also where *numerical* solvers are used. In practice, Pegasus loops through a range of values for the parameters d, λ, ε as well as the degrees of the SOS polynomials $\sigma_{I_i}, \sigma_{U_i}$ and attempts to solve these constraints for each concrete choice of parameters.

While efficient, the use of numerical solvers has its drawbacks, e.g. because the generated coefficients $\mathbf{a}$ need not truly satisfy all the required constraints. This is why Pegasus (and KeYmaera X) treats the generated barrier certificate p only as a *candidate* invariant and performs additional arithmetical checks to ensure that the constraints are truly met. As a cautionary example, Fig. 11 (left) rather misleadingly suggests that $p \leq 0$ is an invariant within its small plot domain. Indeed, Fig. 11 (right) is a zoomed out version of the same plot which shows that the constraint (6) fails to hold for larger values of x, y.

Linear programming (LP) was employed as an alternative to sum-of-squares programming by Sankaranarayanan et al. [75] to generate Lyapunov functions, and later applied by Yang et al. [95] to similarly generate barrier certificates. The main idea behind this approach is to employ a *linear relaxation*, whereby non-negativity of a polynomial p is witnessed, subject to non-negativity of (basis) polynomials $p_1, p_2, \ldots, p_k$, i.e. $p_1 \geq 0 \wedge p_2 \geq 0 \wedge \cdots \wedge p_k \geq 0 \rightarrow p \geq 0$ is reduced to the existence of non-negative Lagrangian multipliers $\lambda_1, \lambda_2, \ldots, \lambda_k$ such that $\lambda_1 p_1 + \lambda_2 p_2 + \cdots + \lambda_k p_k = p$.

In cases when the evolution domain constraint Q is described by a conjunction of polynomial inequalities $Q \equiv q_1 \geq 0 \wedge \cdots \wedge q_l \geq 0$ (e.g. in the case of hyperboxes or polyhedra), one may form all products $p_i = q_1^{\alpha_{1i}} \cdots q_l^{\alpha_{li}}$ up to some maximum total degree and use them to solve the linear relaxation for $p_1 \geq 0 \wedge \cdots \wedge p_k \geq 0 \rightarrow p_{\mathbf{a},d} \geq 0$ using linear programming, obtaining a polynomial which is non-negative on Q. The conditions for barrier certificates are encoded in an obvious way.

⚠ *In using convex optimization methods to search for barrier certificates, one is not concerned with optimizing the value of any particular objective function (the zero function suffices); one is rather interested in finding a feasible solution to a set of constraints. For LP, it is possible to use an SMT solver which supports the theory of linear real arithmetic* (`LRA`, *e.g., as supported by Z3) to search for* models *of formulas describing the constraints to obtain instantiations of the parameter variables in the template; however, in our experience, implementations of linear programming solvers (especially employing interior point algorithms) in Mathematica and MATLAB offer considerably better performance compared to Z3 (which implements the Dual Simplex algorithm* [20]).

5 Strategies for invariant generation

The implementation of primitive invariant generation methods from Sect. 4 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.

5.1 Differential saturation

The main invariant generation strategy Pegasus uses for general non-linear systems is based on a *differential saturation* procedure [61]. Briefly, the procedure loops through a prescribed *sequence* of invariant generation methods and *successively* attempts to strengthen the evolution domain constraint using invariants found by those methods until the desired safety condition is proved. [17] 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

[17] Pegasus partitions problems into subsystems according to variable dependencies in their differential equations [61]. 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.

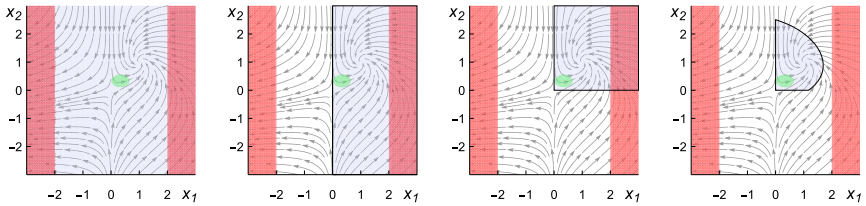


Fig. 12 Invariant synthesis using the differential saturation loop in Pegasus. The domain under consideration at each step is shaded in blue and annotated below each plot, with the polynomial $p = \frac{3}{8}x_1 + \frac{23}{56}x_1^2 - \frac{123}{56}x_2 + \frac{3}{14}x_1x_2 + \frac{29}{28}x_2^2 - 1$

strategy to avoid redundancy. Pegasus orders the methods by computational efficiency, e.g. it first searches for first integrals, followed by Darboux polynomials and barrier certificates. This sequencing allows slower methods to exploit invariants that are quickly generated by earlier methods.

Example 6 The synergy between individual methods exploited by differential saturation is illustrated in Fig. 12 for an example from our benchmarks.

Initially (leftmost plot Fig. 12a), 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 (Fig. 12b), Pegasus confines its search to the region $x_1 > 0$ using the generated Darboux polynomial x_1. In the third plot (Fig. 12c), using $x_1 > 0$, qualitative analysis finds the invariant $x_2 > 0$ (whose invariance depends on $x_1 > 0$) which further confines the evolution domain constraint. Finally (rightmost plot Fig. 12d), Pegasus finds a barrier certificate (of polynomial degree 2) that suffices to show the safety property within the strengthened evolution domain constraint (which, by construction, is invariant). The final invariant region contains several sharp corners and thus *cannot* be directly obtained as the sub-level set of a single polynomial barrier certificate. Instead, it incorporates a conjunction of invariants discovered earlier by other means.

Remark 9 Pegasus extracts *proof hints* from the internal reasoning sequence used in its differential saturation strategy, e.g., it tracks the order of construction of the invariants $x_1 > 0, x_2 > 0, \ldots$ from Example 6 and how they were individually proved. These hints are useful for deductive tools like KeYmaera X because they can be used to guide its proofs for the generated invariants in a corresponding, step-by-step manner, with the most appropriate verification technique for the invariant used at each step.

Given an input safety verification problem, it is *a priori* unknown which of the invariant generation methods used for differential saturation would succeed; and even for those that do succeed, it is difficult to predict the precise duration required. The overall strategy in Pegasus imposes carefully balanced timeouts, where each method called by differential saturation attempts to:

- detect their applicability efficiently to conserve time budgets for other methods if they are not applicable,
- keep track of intermediate results and report partial results (if applicable) when their individual timeouts are hit,

– efficiently check when they are done.

Pegasus uses configuration parameters to adjust timeouts and method behavior, e.g., maximum degree of barrier certificate templates. In addition, Pegasus supports configuration of the overall strategy behavior in terms of combining method results, how it handles method timeouts, and how it detects when the methods succeeded. In the current implementation, and in Sect. 6, we explore the following strategy configuration options:

(**C1**) Auto-Reduction: whether or not to filter redundant invariants when combining results
(**C2**) Heuristic Search: whether or not to apply qualitative analysis and other heuristic search methods
(**C3**) Budget Redistribution: strict method timeouts or redistribution of unused time budget to later methods
(**C4**) Subsystem Splitting: whether or not to analyze subsystems separately

Option (C1) allows Pegasus to find invariants of lower descriptive complexity, which may be more insightful for users and easier to prove in KeYmaera X. Options (C2)–(C4) allow expert users finer control over how Pegasus searches for invariants. For example, (C4) is useful when the input problem is known to consist of many subsystems of ODEs [61] that can be tackled separately. The trade-off between these options is qualitatively evaluated in Sect. 6.

5.2 Differential divide-and-conquer

The differential saturation strategy uses a melting pot of primitive invariant generation methods without (directly) adding more logical or mathematical considerations. The *differential divide-and-conquer* (DDC) proof rule [81] is an example logical technique that also fits well into the Pegasus framework.

Briefly, the rule says that if $p = 0$ is an invariant for both the forwards ODE $\mathbf{x}' = f(\mathbf{x})$ *and* the backwards ODE $\mathbf{x}' = -f(\mathbf{x})$, then the state space partitions into three invariant subspaces $p < 0$, $p = 0$, $p > 0$, and it suffices to consider the invariant generation sub-problems (restricted to each subspace) separately. All Darboux polynomials p (Sect. 4.4.2) meet the forwards-and-backwards invariance criteria and can be used to partition the state space. Indeed, this DDC strategy is already implicitly used in the invariant generation method for rational first integrals in Sect. 4.4.3, which partitions the state space using Darboux polynomials, and then generates rational first integrals on the resulting sub-problems. Pegasus generalizes this by looking for invariants on each sub-problem instead, i.e., by replacing steps 4 and 5 from the method described in Sect. 4.4.2 as follows:

4* For each unpruned *conflict cell* S, define a new invariant generation *sub-problem*, with the original evolution domain constraint Q restricted to $Q \wedge S$.
5* Call the differential saturation strategy (Sect. 5.1) to find an invariant on all newly generated sub-problems.

Example 7 The differential divide-and-conquer strategy is illustrated in Fig. 13 for a tweaked Example 6 with larger initial set and smaller unsafe set.

As before, initially (leftmost plot Fig 13a), 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). Pegasus partitions the problem into three sub-problems, shown in the subsequent plots, using the Darboux polynomial x_1; in those plots, only the part of the plane relevant to each sub-problem is drawn.In the third plot (Fig. 13c, the evolution

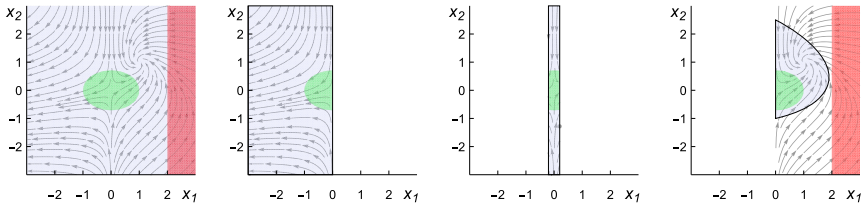


Fig. 13 Invariant synthesis using differential divide-and-conquer in Pegasus. The domain under consideration at each step is shaded in blue and annotated below each plot, with the polynomial $p = \frac{11}{25}x_1 + \frac{7}{100}x_1^2 - \frac{3}{5}x_2 + \frac{3}{25}x_1x_2 + \frac{2}{5}x_2^2 - 1$

domain constraint $x_1 = 0$ is slightly (but soundly) enlarged to $-0.2 \leq x_1 \leq 0.2$ for visibility in the illustration as it would otherwise be an infinitesimal line. In the second (evolution domain constraint $x_1 < 0$, Fig. 13b) and third (evolution domain constraint $x_1 = 0$, enlarged in Fig. 13c) plots, the sub-problems are proved trivially because they contain no unsafe states. In the rightmost plot (Fig. 13d, evolution domain constraint $x_1 > 0$), Pegasus finds a barrier certificate (in blue) that solves the sub-problem.

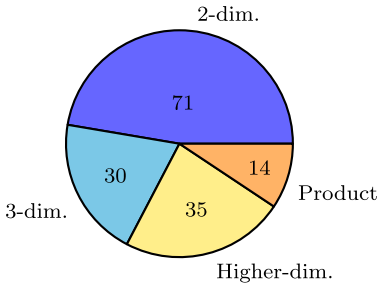
6 Evaluation

This section presents a qualitative evaluation of the invariant generation capabilities of Pegasus and its interaction with the ODE proving tactics of KeYmaera X. The insights obtained from these benchmarks provide useful default configuration options for Pegasus, e.g., those described in Sect. 5.

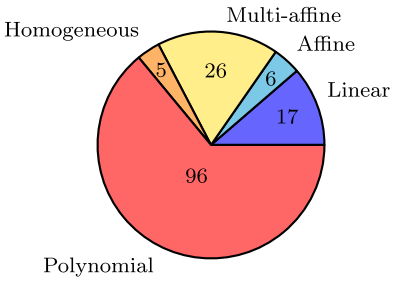
6.1 Benchmark suite

The benchmark suite consists of 150 continuous safety verification problems, with 90 earlier problems [84] and 60 new ones, all drawn from the literature [1,6,16,19,22,27,30,32,35,36,39,44,45,54,70,74,82,83,95–97]. Some are drawn from papers that present and discuss properties of a system of ODEs without explicitly providing initial and safe conditions; in such cases, we design our own initial and safe sets based on the provided discussion.

The suite consists of problems involving linear, affine, multi-affine, or (non-linear) polynomial ODEs over a range of dimensions: 71 two-dimensional systems, 30 three-dimensional systems, 35 higher-dimensional ($\geq$4, $\leq$16) systems, and 14 *product systems* that were formed by randomly combining pairs of two- and three-dimensional systems, see Fig. 14a, b. The problems have a range of topological and logical structures to test the applicability of various invariant generation methods. A summary of the topological structure of the problems is shown in Fig. 14c; the sets involved are either topologically bounded or unbounded (or None, when there is no evolution domain constraint), and either topologically open or closed (or neither). A summary of the logical structure of the problems is shown in Fig. 14d; the formulas involved are either described algebraically by an equation, or by an atomic inequality,



(a) Differential Equation Dimension



(b) Differential Equation Class

	Bounded			Unbounded			
Topology	Open	Closed	Neither	Open	Closed	Neither	None
Initial Set (Pre.)	15	76	13	20	16	10	-
Unsafe Set (Neg. Post.)	1	49	2	26	57	15	-
Evolution Domain	0	26	0	3	10	0	111

(c) Problem topology

Logical Structure	Algebraic	Atomic Inequality	Semi-algebraic	None
Precondition	29	44	77	-
Postcondition	5	74	71	-
Evolution Domain	1	5	33	111

(d) Problem logical structure

Fig. 14 Benchmark suite classification among 150 benchmarks

or, more generally, by a semi-algebraic formula involving conjunctions and disjunctions of equations and inequalities. The experiment was run on commodity hardware.[18]

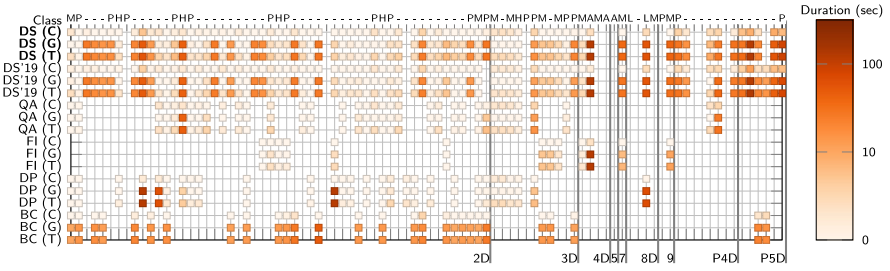
6.2 Differential saturation performance

We analyze the differential saturation strategy compared to each invariant generation method in isolation, measuring the duration of invariant generation, duration of checking the generated invariants, and the total proof duration. We analyze the effect of exposing proof hints with the generated invariants, and the effect of strategy configuration options (C1)–(C4) from Sect. 5.

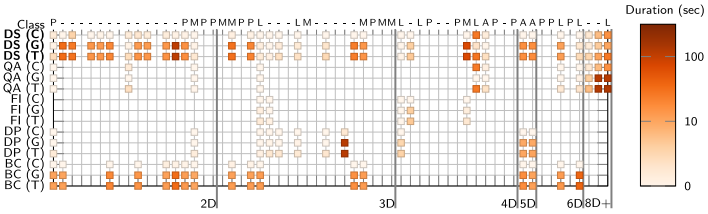
6.2.1 Differential saturation versus individual generation methods

The results comparing differential saturation against individual methods for each benchmark problem are shown in Fig. 15. Several experimental insights can be drawn from these results: (i) different invariant generation methods generally solve different subsets of the problems, (ii) invariant generation almost always dominates total proof duration although invariant checking becomes more expensive as problem dimension increases, (iii) when

[18] MacBook Pro 2019 with 2.6GHz Intel Core i7 (model 9750H) and 32GB memory (2667MHz DDR4 SDRAM), Mathematica 12.1 and MATLAB 2019b with SOSTOOLS 3.03.



(a) 90 benchmark problems from FM 2019 conference version [84]

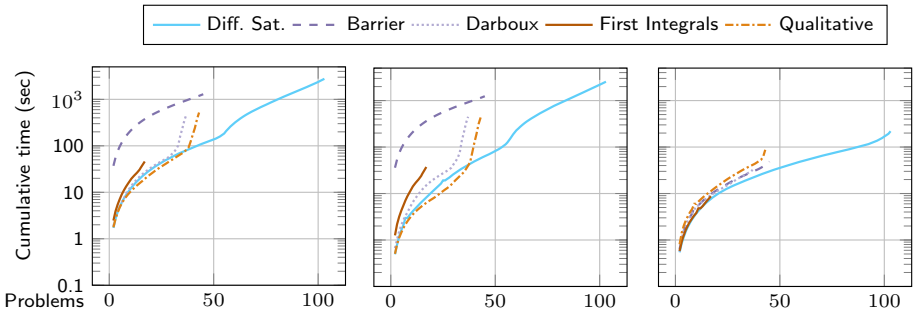


(b) 60 additional benchmark problems

Fig. 15 Comparison of invariant generation methods. Each column represents one benchmark problem and the color 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). Results for the earlier implementation [84] (with new hardware, see Footnote 18) are also shown for comparison (DS'19). The ODE classification for each problem is annotated at the top: homogeneous polynomial (H), polynomial (P), linear (L), affine (A), multi-affine (M), dashes indicate same class as the enclosing labels

multiple methods 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 16 additional problems (of which 7 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. (v) Finally, the performance of Pegasus (with default configuration) has remained relatively stable compared to its earlier version [84].

To evaluate the effectiveness of combining methods by differential saturation, Fig. 16 plots the *accumulated* duration for solving the fastest n out of 150 benchmark problems. The main insights are: (i) differential saturation solves the largest number of problems per accumulated time, i.e., despite sequentially executing invariant generation methods, it often succeeds in trying out the most efficient method first and fails fast when earlier methods are unsuitable; however, qualitative analysis (in isolation) generates some invariants faster when the heuristics it employs for guessing invariant candidates are successful, (ii) cumulatively, invariant generation duration dominates invariant checking duration (note logarithmic scaling of the time axis in Fig. 16); this effect is especially pronounced for barrier certificates, but can also be observed in all other methods when solving more expensive (harder) problems, (iii) first integrals are least expensive to check when they solve problems, (iv) qualitative analysis is less expensive for generation than other methods, but is most expensive for checking



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

Fig. 16 Cumulative logarithmic time (in seconds) taken to solve the fastest n problems (more problems solved and flatter is better; accumulated generation and checking duration of Diff. Sat. compared at 50/75/100 fastest problems)

because the invariants it generates often have high descriptive complexity and may not have simple invariance justifications.

6.2.2 Differential saturation configuration options

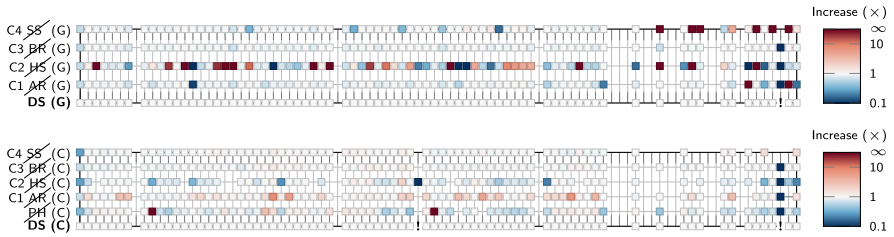
Next, we explore the effect of configuration options on the invariant generation and subsequent checking duration by disabling features of the differential saturation procedure. Specifically, we executed differential saturation with:

C1 ~~AR~~ No Auto-Reduction, which is expected to speed up generation but may cause redundant cuts or unnecessarily complicated invariants.
C2 ~~HS~~ No Heuristic Search, which is expected to produce more principled invariants and more specific proof hints but solve fewer problems.
C3 ~~BR~~ No Budget Redistribution, which is expected to result in a more predictable generation duration but solve fewer problems.
C4 ~~SS~~ No Subsystem Splitting, which is expected to result in faster performance on problems without clear subsystems, but solve fewer problems overall (e.g., the product problems should benefit from C4).
~~PH~~ No Proof Hints, which is expected to slow down invariant checking but have no effect on invariant generation.

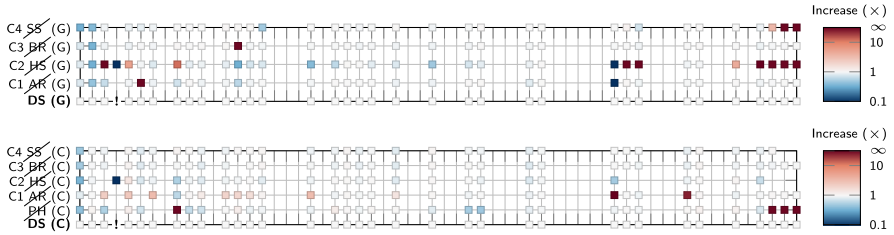
Figure 17 shows the benefits and drawbacks of each configuration option on the suite of benchmark problems, while Fig. 18 summarizes the cumulative effect of configuration options. Since these configuration options are tuning parameters that offer fine-grained control over differential saturation for Pegasus, their cumulative effect over all 150 problems is small, see Fig. 18.

Except for Heuristic Search (C2), disabling features results in similar (or slightly faster) generation duration for most problems, but at the expense of not solving others, see Figs. 17a and 17b (top). On three particular problems, disabling features helped Pegasus to solve the problem within the given time budget. Overall, the configuration options have little net effect on most problems but can make a difference on select problems:

– No Proof Hints (~~PH~~): Several problems check slightly faster *without* following the proof hints, which indicates that KeYmaera X's checking procedure is sometimes able to find



(a) Invariant generation (top) and checking (bottom) duration in multiples of differential saturation (90 benchmark problems from FM 2019 conference version [84])



(b) Invariant generation (top) and checking (bottom) duration in multiples of differential saturation (60 additional benchmark problems)

Fig. 17 Influence of configuration options: no Auto-Reduction (**C1** ~~AR~~), no Heuristic Search (**C2** ~~HS~~), no Budget Redistribution (**C3** ~~BR~~), no Subsystem Splitting (**C4** ~~SS~~), and no Proof Hints(~~PH~~) . A **!** mark indicates that the default Differential Saturation (DS) configuration failed to generate or check that problem, while one (or more) of the other configuration options succeeded

more efficient proofs than the hints. However, there are also problems that check slightly slower and several problems that fail to check without proof hints. Conclusion: proof hints can be extremely helpful; they should be kept wherever possible, especially since they are inexpensive to produce in Pegasus. KeYmaera X could try its default checking procedure first and fallback to hints if the default fails.

– No Auto-Reduction (**C1** ~~AR~~): significant increase in proof checking duration on several examples, but decrease in generation duration on several examples as well. Conclusion: C1 auto-reduction is useful for checking but at the expense of generation duration; it should be provided as an optional post-processing step for users interested in more succinct invariants.
– No Heuristic Search (**C2** ~~HS~~): variable severe impact (both positive and negative) on generation duration across examples, but fails to generate invariants for several examples. However, checking duration is generally improved for principled invariants generated without heuristics. Notably, two problems were successfully solved *solely* by C2 out of all other configuration options. Conclusion: C2 should be a configurable option for users, but should typically be enabled when the ultimate goal is to solve a given problem and invariant generation time is not a significant constraint.
– No Budget Redistribution (**C3** ~~BR~~): minor impact on both generation and checking duration, except failing to solve one problem. Conclusion: C3 is not very impactful, but could be left enabled by default as a failsafe.
– No Subsystem Splitting (**C4** ~~SS~~): minor impact on both generation and checking duration for solved problems, but solves fewer problems (mostly product system and higher-

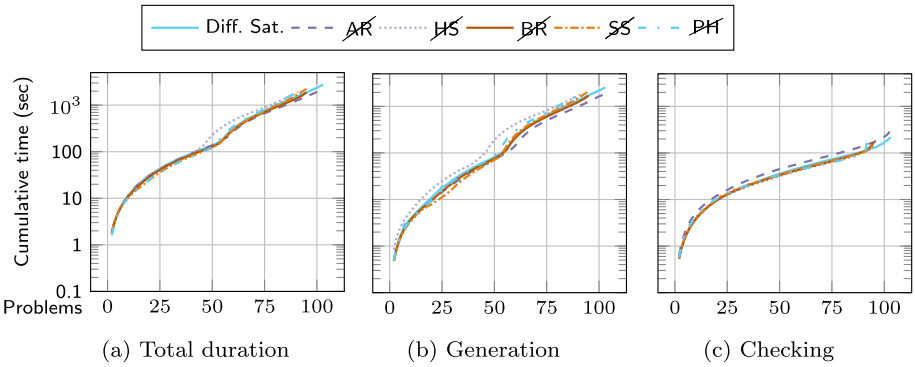


Fig. 18 Configuration options: cumulative logarithmic time (in seconds) taken to solve the fastest n problems (more problems solved and flatter is better)

dimensional problems). Conclusion: C4 is a useful technique in invariant generation and should typically be enabled.

7 Related work

Techniques developed for *qualitative simulation* have been applied to prove temporal properties of continuous systems by Shults and Kuipers [79], as well as Loeser, Iwasaki and Fikes [46]. Zhao [99] 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 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. [2]. The case of systems whose continuous motion is governed by non-linear ODEs was studied in the work of Tiwari and Khanna [88,90]. Tiwari studied reachability of linear systems [87], using information from real eigenvectors and ideas from qualitative abstraction to generate invariants. Zaki *et al.* [97] were the first to apply Darboux polynomials to verification of continuous systems using discrete abstraction. Numerous works employ barrier certificates for verification [16,40,66,83,95]. Since we implement many of the above techniques as methods for invariant generation in Pegasus, our work draws heavily upon ideas developed previously in the verification and hybrid systems communities. Previous work [81] 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 is quite distinct from the use of trusted oracles in the work of Wang et al. [92] (for the HHL prover) and, notably, provides a *sound* framework for reasoning with continuous invariants that is significantly less exposed to soundness issues in external tools.

A *complete* semi-algorithm for computing algebraic invariants (described by zero sets of polynomial functions) for polynomial systems of ODEs was developed by Ghorbal and Platzer [27]. An interesting development along very similar lines was also recently pursued by Boreale [11], whose method makes use of the algebraic nature of the precondition (initial set) in the verification problem in order to speed up the algebraic invariant generation. Both of these (semi-)algorithms involve enumeration of polynomial templates; the biggest practical

difficulty stems from the computational cost of minimizing the rank of symbolic matrices [27], and computing the generators of *real radical ideals* [11], both of which are difficult problems with the latter having few algorithms with robust implementations currently in existence.[19] In the future, we hope to extend Pegasus with an implementation of these techniques.

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 improve user experience. Better proof automation will certainly also be useful in future applications of provably correct runtime monitoring frameworks, such as ModelPlex [50], as well as frameworks for generating verified controller executables, such as VeriPhy [10]. 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 for diagonalizable classes of linear systems with tame eigenvalues [26,42], as well as more generally [1]. The complexity of invariants obtained using these methods may not always make them practical, but they would provide a valuable fallback when 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. For pragmatic reasons, KeYmaera X currently uses Mathematica's implementation of real quantifier elimination to check validity of first-order real arithmetic formulas. Removing this reliance by efficiently building fully formal proofs of real arithmetic formulas within dL (e.g. through exhibiting appropriate witnesses or using proof-producing procedures; see [63] for an overview) is an important task for the future.

Other important topics not addressed in this article concern *stability* and *robustness* of continuous invariants [29,33,38,41]. These notions are important in ensuring that the generated invariants are reflective of the real world, and are not merely by-products of mathematical idealization.

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 may require expertise and ingenuity, just like manually selecting support function templates for reachability tools [23], and presents a major practical hurdle in the way of wider industrial adoption of this technology. In this article, 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

[19] Although an *incomplete* invariant generation procedure could still employ inexpensive ad-hoc methods to compute generators of real radical ideals; likewise, generators of (complex) radical ideals can be used instead in a sound but incomplete algebraic invariant generation algorithm [11, § 5].

first large-scale effort in 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 guest editor for handling this article, the anonymous reviewers for providing value feedback, and FM 2019 for the special issue invitation.

References

1. Almagor S, Kelmendi E, Ouaknine J, Worrell J (2020) Invariants for continuous linear dynamical systems. In: ICALP, *LIPIcs*, vol 168, pp 107:1–107:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik. https://doi.org/10.4230/LIPIcs.ICALP.2020.107
2. Alur R, Henzinger TA, Lafferriere G, Pappas GJ (2000) Discrete abstractions of hybrid systems. Proc IEEE 88(7):971–984. https://doi.org/10.1109/5.871304
3. Arrowsmith D, Place CM (1992) Dynamical systems: differential equations, maps, and chaotic behaviour, vol 5. CRC Press, Boca Raton
4. Beckert B, Giese M, Hähnle R, Klebanov V, Rümmer P, Schlager S, Schmitt PH (2007) The KeY system 1.0 (deduction component). In: Pfenning F (ed) CADE, *LNCS*, vol 4603, pp 379–384. Springer. https://doi.org/10.1007/978-3-540-73595-3_26
5. Bellman R (1962) Vector Lyapunov functions. SIAM J Control Optim 1(1):32–34. https://doi.org/10.1137/0301003
6. Ben Sassi MA, Girard A, Sankaranarayanan S (2014) Iterative computation of polyhedral invariants sets for polynomial dynamical systems. In: CDC, pp 6348–6353. IEEE. https://doi.org/10.1109/CDC.2014.7040384
7. Bogomolov S, Giacobbe M, Henzinger TA, Kong H (2017) Conic abstractions for hybrid systems. In: Abate A, Geeraerts G (eds) FORMATS, *LNCS*, vol 10419, pp 116–132. Springer. https://doi.org/10.1007/978-3-319-65765-3_7
8. Böhme S, Weber T (2010) Fast LCF-style proof reconstruction for Z3. In: Kaufmann M, Paulson LC (eds) ITP, *LNCS*, vol 6172, pp 179–194. Springer. https://doi.org/10.1007/978-3-642-14052-5_14
9. Bohrer B, Fernández M, Platzer A (2019) dL_ι: Definite descriptions in differential dynamic logic. In: Fontaine P (ed) CADE, *LNCS*, vol 11716, pp 94–110. Springer. https://doi.org/10.1007/978-3-030-29436-6_6
10. Bohrer B, Tan YK, Mitsch S, Myreen MO, Platzer A (2018) VeriPhy: verified controller executables from verified cyber-physical system models. In: Foster JS, Grossman D (eds) PLDI. ACM, New York, pp 617–630. https://doi.org/10.1145/3192366.3192406
11. Boreale M (2020) Complete algorithms for algebraic strongest postconditions and weakest preconditions in polynomial ODEs. Science of Computer Programming 193. https://doi.org/10.1016/j.scico.2020.102441
12. Chen M, Han X, Tang T, Wang S, Yang M, Zhan N, Zhao H, Zou L (2017) MARS: a toolchain for modelling, analysis and verification of hybrid systems. In: Hinchey MG, Bowen JP, Olderog E (eds) Provably correct systems, NASA monographs in systems and software engineering. Springer, Berlin, pp 39–58. https://doi.org/10.1007/978-3-319-48628-4_3

13. Chicone C (2006) Ordinary differential equations with applications, 2nd edn. Springer, New York. https://doi.org/10.1007/0-387-35794-7
14. Collins GE (1975) Quantifier elimination for real closed fields by cylindrical algebraic decompostion, *LNCS*, vol 33, pp 134–183. Springer. https://doi.org/10.1007/3-540-07407-4_17
15. Cox DA, Little J, O'Shea D (2015) Ideals, varieties, and algorithms, 4th edn. Springer, Berlin. https://doi.org/10.1007/978-3-319-16721-3
16. Dai L, Gan T, Xia B, Zhan N (2017) Barrier certificates revisited. J Symb Comput 80:62–86. https://doi.org/10.1016/j.jsc.2016.07.010
17. Darboux JG (1878) Mémoire sur les équations différentielles algébriques du premier ordre et du premier degré. Bull Sci Math 2(1):151–200
18. Denman W, Muñoz CA (2014) Automated real proving in PVS via MetiTarski. In: Jones CB, Pihlajasaari P, Sun J (eds) FM, *LNCS*, vol 8442, pp 194–199. Springer. https://doi.org/10.1007/978-3-319-06410-9_14
19. Djaballah A, Chapoutot A, Kieffer M, Bouissou O (2017) Construction of parametric barrier functions for dynamical systems using interval analysis. Automatica 78:287–296. https://doi.org/10.1016/j.automatica.2016.12.013
20. Dutertre B, de Moura LM (2006) A fast linear-arithmetic solver for DPLL(T). In: Ball T, Jones RB (eds) CAV, *LNCS*, vol 4144, pp 81–94. Springer. https://doi.org/10.1007/11817963_11
21. Falconi M, Llibre J (2004) $n-1$ independent first integrals for linear differential systems in $\mathbb{R}^n$ and $\mathbb{C}^n$. Qual Theory Dyn Syst 4(2):233–254. https://doi.org/10.1007/BF02970860
22. Ferragut A, Giacomini H (2010) A new algorithm for finding rational first integrals of polynomial vector fields. Qual Theory Dyn Syst 9(1–2):89–99. https://doi.org/10.1007/s12346-010-0021-x
23. Frehse G, Le Guernic C, Donzé A, Cotton S, Ray R, Lebeltel O, Ripado R, Girard A, Dang T, Maler O (2011) SpaceEx: scalable verification of hybrid systems. In: Gopalakrishnan G, Qadeer S (eds) CAV, *LNCS*, vol 6806, pp 379–395. Springer. https://doi.org/10.1007/978-3-642-22110-1_30
24. Fulton N, Mitsch S, Bohrer B, Platzer A (2017) Bellerophon: tactical theorem proving for hybrid systems. In: Ayala-Rincón M, Muñoz CA (eds) ITP, *LNCS*, vol 10499, pp 207–224. Springer. https://doi.org/10.1007/978-3-319-66107-0_14
25. Fulton N, Mitsch S, Quesel J, Völp M, Platzer A (2015) KeYmaera X: an axiomatic tactical theorem prover for hybrid systems. In: Felty AP, Middeldorp A (eds) CADE, *LNCS*, vol 9195, pp 527–538. Springer. https://doi.org/10.1007/978-3-319-21401-6_36
26. Gan T, Chen M, Li Y, Xia B, Zhan N (2018) Reachability analysis for solvable dynamical systems. IEEE Trans Autom Control 63(7):2003–2018. https://doi.org/10.1109/TAC.2017.2763785
27. Ghorbal K, Platzer A (2014) Characterizing algebraic invariants by differential radical invariants. In: Ábrahám E, Havelund K (eds) TACAS, *LNCS*, vol 8413, pp 279–294. Springer. https://doi.org/10.1007/978-3-642-54862-8_19
28. Ghorbal K, Sogokon A, Platzer A (2017) A hierarchy of proof rules for checking positive invariance of algebraic and semi-algebraic sets. Comput Lang Syst Struct 47(1):19–43. https://doi.org/10.1016/j.cl.2015.11.003
29. Goebel R, Hespanha J, Teel AR, Cai C, Sanfelice R (2004) Hybrid systems: generalized solutions and robust stability. In: NOLCOS, vol 37, pp 1–12. Stuttgart, Germany. https://doi.org/10.1016/S1474-6670(17)31194-1
30. Gorbuzov VN, Pranevich AF (2012) First integrals of ordinary linear differential systems. CoRR arXiv:1201.4141
31. Goriely A (2001) Integrability and nonintegrability of dynamical systems. World Scientific. https://doi.org/10.1142/3846
32. Gulwani S, Tiwari A (2008) Constraint-based approach for analysis of hybrid systems. In: Gupta A, Malik S (eds) CAV, *LNCS*, vol 5123, pp 190–203. Springer. https://doi.org/10.1007/978-3-540-70545-1_18
33. Haddad WM, Chellaboina V (2008) Nonlinear dynamical systems and control: a Lyapunov-based approach. Princeton University Press, Princeton
34. Herbrand J (1930) Recherches sur la théorie de la démonstration. Université de Paris, Faculté des Sciences, Doctorat d'état
35. Immler F, Althoff M, Chen X, Fan C, Frehse G, Kochdumper N, Li Y, Mitra S, Tomar MS, Zamani M (2018) ARCH-COMP18 category report: continuous and hybrid systems with nonlinear dynamics. In: Frehse G, Althoff M, Bogomolov S, Johnson TT (eds) ARCH, EPiC series in computing, vol 54. EasyChair, pp 53–70
36. Kapinski J, Deshmukh JV, Sankaranarayanan S, Arechiga N (2014) Simulation-guided Lyapunov analysis for hybrid dynamical systems. In: Fränzle M, Lygeros J (eds) HSCC. ACM, New York, pp 133–142. https://doi.org/10.1145/2562059.2562139

37. Kasner E (1925) Solutions of the Einstein equations involving functions of only one variable. Trans Am Math Soc 27(2):155–162. https://doi.org/10.1090/S0002-9947-1925-1501305-1
38. Khalil HK (1992) Nonlinear systems. Macmillan Publishing Company, New York
39. Kong H, Bogomolov S, Schilling C, Jiang Y, Henzinger TA (2017) Safety verification of nonlinear hybrid systems based on invariant clusters. In: Frehse G, Mitra S (eds) HSCC. ACM, New York, pp 163–172. https://doi.org/10.1145/3049797.3049814
40. Kong H, He F, Song X, Hung WNN, Gu M (2013) 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. https://doi.org/10.1007/978-3-642-39799-8_17
41. Kong S, Gao S, Chen W, Clarke EM (2015) dReach: δ-reachability analysis for hybrid systems. In: Baier C, Tinelli C (eds) TACAS, *LNCS*, vol 9035, pp 200–205. Springer. https://doi.org/10.1007/978-3-662-46681-0_15
42. Lafferriere G, Pappas GJ, Yovine S (2001) Symbolic reachability computation for families of linear vector fields. J Symb Comput 32(3):231–253. https://doi.org/10.1006/jsco.2001.0472
43. Liu J, Lv J, Quan Z, Zhan N, Zhao H, Zhou C, Zou L (2010) A calculus for hybrid CSP. In: Ueda K (ed) APLAS, *LNCS*, vol 6461, pp 1–15. Springer. https://doi.org/10.1007/978-3-642-17164-2_1
44. Liu J, Zhan N, Zhao H (2011) Computing semi-algebraic invariants for polynomial dynamical systems. In: Chakraborty S, Jerraya A, Baruah SK, Fischmeister S (eds) EMSOFT. ACM, New York, pp 97–106. https://doi.org/10.1145/2038642.2038659
45. Llibre J, Zhang X (2002) Invariant algebraic surfaces of the Lorenz system. J Math Phys 43(3):1622–1645. https://doi.org/10.1063/1.1435078
46. Loeser T, Iwasaki Y, Fikes R (1998) Safety verification proofs for physical systems. In: Proc. of the 12th intl. workshop on qualitative reasoning, pp 88–95
47. Man Y (1993) Computing closed form solutions of first order ODEs using the Prelle–Singer procedure. J Symb Comput 16(5):423–443. https://doi.org/10.1006/jsco.1993.1057
48. Man Y (1994) First integrals of autonomous systems of differential equations and the Prelle–Singer procedure. J Phys A Math Gen 27(10):L329–L332. https://doi.org/10.1088/0305-4470/27/10/005
49. Mishra B (1993) Algorithmic algebra. Springer, Berlin. https://doi.org/10.1007/978-1-4612-4344-1
50. Mitsch S, Platzer A (2016) ModelPlex: verified runtime validation of verified cyber-physical system models. Formal Methods Syst Des 49(1–2):33–74. https://doi.org/10.1007/s10703-016-0241-z
51. Mitsch S, Platzer A (2020) A retrospective on developing hybrid systems provers in the KeYmaera family: a tale of three provers. In: Ahrendt W, Bubel R, Beckert B, Hähnle R, Ulbrich M (eds) Deductive verification: the state of the future, LNCS. Springer, Berlin
52. Olver PJ (2000) Applications of Lie groups to differential equations, graduate texts in mathematics, vol 107, 2nd edn. Springer. https://doi.org/10.1007/978-1-4684-0274-2
53. Papachristodoulou A, Anderson J, Valmorbida G, Prajna S, Seiler P, Parrilo PA (2013) SOSTOOLS version 3.00 sum of squares optimization toolbox for MATLAB. CoRR arXiv:1310.4716
54. Papachristodoulou A, Prajna S (2002) On the construction of Lyapunov functions using the sum of squares decomposition. In: CDC, vol 3, pp 3482–3487. https://doi.org/10.1109/CDC.2002.1184414
55. Parrilo PA (2000) Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization. Ph.D. thesis, California Institute of Technology. https://doi.org/10.7907/2K6Y-CH43
56. Platzer A (2008) Differential dynamic logic for hybrid systems. J Autom Reason 41(2):143–189. https://doi.org/10.1007/s10817-008-9103-8
57. Platzer A (2012) The complete proof theory of hybrid systems. In: LICS, pp 541–550. IEEE Computer Society. https://doi.org/10.1109/LICS.2012.64
58. Platzer A (2012) A differential operator approach to equational differential invariants—(invited paper). In: Beringer L, Felty AP (eds) ITP, LNCS, vol 7406, pp 28–48. Springer. https://doi.org/10.1007/978-3-642-32347-8_3
59. Platzer A (2012) Logics of dynamical systems. In: LICS, pp 13–24. IEEE Computer Society. https://doi.org/10.1109/LICS.2012.13
60. Platzer A (2017) A complete uniform substitution calculus for differential dynamic logic. J Autom Reason 59(2):219–265. https://doi.org/10.1007/s10817-016-9385-1
61. Platzer A, Clarke EM (2009) Computing differential invariants of hybrid systems as fixedpoints. Formal Methods Syst Des 35(1):98–120. https://doi.org/10.1007/s10703-009-0079-8
62. Platzer A, Quesel J (2008) 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. https://doi.org/10.1007/978-3-540-71070-7_15
63. Platzer A, Quesel J, Rümmer P (2009) Real world verification. In: Schmidt RA (ed) CADE, *LNCS*, vol 5663, pp 485–501. Springer. https://doi.org/10.1007/978-3-642-02959-2_35

64. Platzer A, Tan YK (2020) Differential equation invariance axiomatization. J ACM 67:1. https://doi.org/10.1145/3380825
65. Pontryagin LS (1962) Ordinary differential equations. Pergamon Press, Oxford. https://doi.org/10.1016/C2013-0-01692-1
66. Prajna S, Jadbabaie A (2004) Safety verification of hybrid systems using barrier certificates. In: Alur R, Pappas GJ (eds) HSCC, *LNCS*, vol 2993, pp 477–492. Springer. https://doi.org/10.1007/978-3-540-24743-2_32
67. Prelle MJ, Singer MF (1983) Elementary first integrals of differential equations. Trans Am Math Soc 279(1):215–229. https://doi.org/10.1090/S0002-9947-1983-0704611-X
68. Rebiha R, Moura AV, Matringe N (2015) Generating invariants for non-linear hybrid systems. Theor Comput Sci 594:180–200. https://doi.org/10.1016/j.tcs.2015.06.018
69. Renegar J (1990) Recent progress on the complexity of the decision problem for the reals. In: Goodman JE, Pollack R, Steiger W (eds) Discrete and computational geometry: papers from the DIMACS special year, vol 6. DIMACS/AMS, New York, pp 287–308. https://doi.org/10.1007/978-3-7091-9459-1_11
70. Rodríguez-Carbonell E, Tiwari A (2005) Generating polynomial invariants for hybrid systems. In: Morari M, Thiele L (eds) HSCC, *LNCS*, vol 3414, pp 590–605. Springer. https://doi.org/10.1007/978-3-540-31954-2_38
71. Rouche N, Habets P, Laloy M (1977) Stability theory by Liapunov's direct method, Appl. Math. Sci., vol 22. Springer. https://doi.org/10.1007/978-1-4684-9362-7
72. Roux P, Voronin Y, Sankaranarayanan S (2018) Validating numerical semidefinite programming solvers for polynomial invariants. Form Methods Syst Des 53(2):286–312. https://doi.org/10.1007/s10703-017-0302-y
73. Roy MF (1996) Basic algorithms in real algebraic geometry and their complexity: from Sturm's theorem to the existential theory of reals. De Gruyter Expos Math 23:1–67. https://doi.org/10.1515/9783110811117
74. Sankaranarayanan S (2010) Automatic invariant generation for hybrid systems using ideal fixed points. In: Johansson KH, Yi W (eds) HSCC. ACM, New York, pp 221–230
75. Sankaranarayanan S, Chen X, Ábrahám E (2013) Lyapunov function synthesis using Handelman representations. In: NOLCOS, pp 576–581. https://doi.org/10.3182/20130904-3-FR-2041.00198
76. Sankaranarayanan S, Sipma HB, Manna Z (2008) Constructing invariants for hybrid systems. Form Methods Syst Des 32(1):25–55. https://doi.org/10.1007/s10703-007-0046-1
77. Schlomiuk D (1993) Algebraic and geometric aspects of the theory of polynomial vector fields. In: NATO ASI series, vol 408, pp 429–467. Springer, Netherlands. https://doi.org/10.1007/978-94-015-8238-4_10
78. Shi S (2007) On the nonexistence of rational first integrals for nonlinear systems and semiquasihomogeneous systems. J Math Anal Appl 335(1):125–134. https://doi.org/10.1016/j.jmaa.2007.01.060
79. Shults B, Kuipers B (1997) Proving properties of continuous systems: qualitative simulation and temporal logic. Artif Intell 92(1–2):91–129. https://doi.org/10.1016/S0004-3702(96)00050-1
80. Slotine JJE, Li W (1991) Applied nonlinear control. Prentice-Hall Inc., Upper Saddle River
81. Sogokon A, Ghorbal K, Jackson PB, Platzer A (2016) A method for invariant generation for polynomial continuous systems. In: Jobstmann B, Leino KRM (eds) VMCAI, *LNCS*, vol 9583, pp 268–288. Springer. https://doi.org/10.1007/978-3-662-49122-5_13
82. Sogokon A, Ghorbal K, Johnson TT (2016) Non-linear continuous systems for safety verification. In: Frehse G, Althoff M (eds) ARCH, EPiC series in computing, vol 43. EasyChair, pp 42–51
83. Sogokon A, Ghorbal K, Tan YK, Platzer A (2018) Vector barrier certificates and comparison systems. In: Havelund K, Peleska J, Roscoe B, de Vink EP (eds) FM, *LNCS*, vol 10951, pp 418–437. Springer. https://doi.org/10.1007/978-3-319-95582-7_25
84. Sogokon A, Mitsch S, Tan YK, Cordwell K, Platzer A (2019) Pegasus: a framework for sound continuous invariant generation. In: ter Beek MH, McIver A, Oliveira JN (eds) FM, *LNCS*, vol 11800, pp 138–157. Springer. https://doi.org/10.1007/978-3-030-30942-8_10
85. Strogatz SH (2001) Nonlinear dynamics and chaos. Studies in nonlinearity. Westview Press, Boulder
86. Sturm T, Tiwari A (2011) Verification and synthesis using real quantifier elimination. In: Schost É, Emiris IZ (eds) ISSAC, pp 329–336. ACM. https://doi.org/10.1145/1993886.1993935
87. Tiwari A (2003) Approximate reachability for linear systems. In: Maler O, Pnueli A (eds) HSCC, *LNCS*, vol 2623, pp 514–525. Springer. https://doi.org/10.1007/3-540-36580-X_37
88. Tiwari A (2008) Abstractions for hybrid systems. Form Methods Syst Des 32(1):57–83. https://doi.org/10.1007/s10703-007-0044-3
89. Tiwari A (2008) Generating box invariants. In: Egerstedt M, Mishra B (eds) HSCC, *LNCS*, vol 4981, pp 658–661. Springer. https://doi.org/10.1007/978-3-540-78929-1_58
90. Tiwari A, Khanna G (2002) Series of abstractions for hybrid automata. In: Tomlin C, Greenstreet MR (eds) HSCC, *LNCS*, vol 2289, pp 465–478. Springer. https://doi.org/10.1007/3-540-45873-5_36

91. Tiwari A, Khanna G (2004) Nonlinear systems: approximating reach sets. In: Alur R, Pappas GJ (eds) HSCC, *LNCS*, vol 2993, pp 600–614. Springer. https://doi.org/10.1007/978-3-540-24743-2_40
92. Wang S, Zhan N, Zou L (2015) An improved HHL prover: an interactive theorem prover for hybrid systems. In Butler MJ, Conchon S, Zaïdi F (eds) ICFEM, *LNCS*, vol 9407, pp 382–399. Springer. https://doi.org/10.1007/978-3-319-25423-4_25
93. Weber T (2006) Integrating a SAT solver with an LCF-style theorem prover. Electr Notes Theor Comput Sci 144(2):67–78. https://doi.org/10.1016/j.entcs.2005.12.007
94. Weber T (2011) SMT solvers: new oracles for the HOL theorem prover. STTT 13(5):419–429. https://doi.org/10.1007/s10009-011-0188-8
95. Yang Z, Huang C, Chen X, Lin W, Liu Z (2016) A linear programming relaxation based approach for generating barrier certificates of hybrid systems. In: Fitzgerald JS, Heitmeyer CL, Gnesi S, Philippou A (eds) FM, *LNCS*, vol 9995, pp 721–738. https://doi.org/10.1007/978-3-319-48989-6_44
96. Yang Z, Wu M, Lin W (2020) An efficient framework for barrier certificate generation of uncertain nonlinear hybrid systems. Nonlinear Anal Hybrid Syst 36:100837. https://doi.org/10.1016/j.nahs.2019.100837
97. Zaki MH, Denman W, Tahar S, Bois G (2009) Integrating abstraction techniques for formal verification of analog designs. J Aerosp Comput Inf Commun 6(5):373–392. https://doi.org/10.2514/1.44289
98. Zhang X (2017) Integrability of dynamical systems: algebra and analysis. Developments in Mathematics, vol 47. Springer. https://doi.org/10.1007/978-981-10-4226-3
99. Zhao F (1994) Extracting and representing qualitative behaviors of complex systems in phase space. Artif Intell 69(1–2):51–92. https://doi.org/10.1016/0004-3702(94)90078-7