Bellerophon: Tactical Theorem Proving for Hybrid Systems*

Nathan Fulton, Stefan Mitsch, Rose Bohrer, and André Platzer

Computer Science Department, Carnegie Mellon University, Pittsburgh PA, USA
{nathanfu,smitsch,aplatzer}@cs.cmu.edu, rose.bohrer.cs@gmail.com

Abstract. Hybrid systems combine discrete and continuous dynamics, which makes them attractive as models for systems that combine computer control with physical motion. Verification is undecidable for hybrid systems and challenging for many models and properties of practical interest. Thus, human interaction and insight are essential for verification. Interactive theorem provers seek to increase user productivity by allowing them to focus on those insights. We present a tactics language and library for hybrid systems verification, named Bellerophon, that provides a way to convey insights by programming hybrid systems proofs.
We demonstrate that in focusing on the important domain of hybrid systems verification, Bellerophon emerges with unique automation that provides a productive proving experience for hybrid systems from a small foundational prover core in the KeYmaera X prover. Among the automation that emerges are tactics for decomposing hybrid systems, discovering and establishing invariants of nonlinear continuous systems, arithmetic simplifications to maximize the benefit of automated solvers and general-purpose heuristic proof search. Our presentation begins with syntax and semantics for the Bellerophon tactic combinator language, culminating in an example verification effort exploiting Bellerophon's support for invariant and arithmetic reasoning for a non-solvable system.

1 Introduction

Cyber-Physical Systems combine computer control with physical dynamics in ways that are often safety-critical. Reasoning about safety properties of Cyber-Physical Systems requires analyzing the system's discrete and continuous dynamics together in a *hybrid system* [2,13]. For example, establishing safety of an adaptive cruise controller in a car requires reasoning about the computations of the controller together with the resulting physical motion of the car.

Theorem proving is an attractive technique for verifying correctness properties of hybrid systems because it is applicable to a large class of hybrid systems [25]. Verification for hybrid systems is not semidecidable, thus requiring

* This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246 and NSF CNS-1446712. This research was sponsored by the AFOSR under grant number FA9550-16-1-0288. This research was supported as part of the Future of Life Institute (futureoflife.org) FLI-RFP-AI1 program, grant #2015-143867.

Table 1. Comparison to related verification tools and provers

Tool	Small Core	HS Library	HS Auto	Scriptable	External Tools
KeYmaera X	Yes	Yes	Yes	Yes	SC
SpaceEx	No	No	Yes	No	SC
Isabelle,Coq,HOL	Yes	No	No	Yes	No
KeYmaera 3	No	Yes	Yes	SC	SC

human assistance along two major dimensions. First, general-case hybrid systems proving requires identifying invariants of loops and differential equations, which is undecidable in both theory and practice. Second, the remaining verification tasks consist of first-order logic over the reals with polynomial terms. Decision procedures exist which are complete in theory [7], but are only complete in practice if a human provides additional guidance. Because both these dimensions are essential to hybrid systems proving, innovating along these dimensions benefits a wide array of hybrid systems verification tasks.

We argue that trustworthy and productive hybrid systems theorem proving requires: *1)* a small foundational core; *2)* a library of high-level primitives automating common deductions (e.g., computing Lie Derivatives, computing and proving solutions of ODEs, propagating quantities across dynamics in which they do not change, automated application of invariant candidates, and conservation/symmetry arguments); and *3)* scriptable heuristic search automation.

Even though these ingredients can be found scattered across a multitude of theorem provers, their combination to a tactical theorem proving technique for hybrid systems is non-obvious. Table 1 compares several tools along the dimensions that we identify as crucial to productive hybrid systems verification (SC indicates a soundness-critical dependency on user-defined tactics or on an external implementation of a more scalable arithmetic decision procedure).

General purpose theorem provers, such as Coq [20] and Isabelle [23], have small foundational cores and tactic languages, but their tactic languages and automation are not tailored to the needs of hybrid systems. This paper addresses the problem of getting from a strong mathematical foundation of hybrid systems [27] to a productive hybrid systems theorem proving tool. Reachability analysis tools, e.g. SpaceEx [11], provide automated hybrid systems verification for *linear* hybrid systems, but at the expense of a large trusted codebase and limited ways of helping when automation fails, which is inevitable due to the undecidability of the problem. KeYmaera's [29] user-defined rules are no adequate solution because they enlarge the trusted codebase and are difficult to get right.

KeYmaera X [12] is structured from the very beginning to maintain a small and trustworthy core, upon which this paper builds the Bellerophon tactic language. Using these logical foundations [27], we develop a set of automated deduction procedures. These procedures manifest themselves as a *library of hybrid systems primitives* in which complex hybrid systems can be interactively verified. Finally, heuristic automation tactics written in Bellerophon automatically apply these primitives to provide automation of hybrid systems reachability analysis.

Table 2. Hybrid Programs

Program Statement	**Meaning**
$\alpha;\beta$	Sequentially composes α and β.
$\alpha \cup \beta$	Executes either α or β.
α^*	Repeats α zero or more times.
$x := \theta$	Evaluates term θ and assigns result to x.
$x := *$	Assigns an arbitrary real value to x.
$\{x_1' = \theta_1, ..., x_n' = \theta_n \& F\}$	Continuous evolution within F along this ODE.
$?F$	Aborts if formula F is not true.

Contributions. This paper demonstrates how to combine a small foundational core [27], reusable automated deductions, and problem-specific proof-search tactics into a tactical theorem prover for hybrid systems. It presents Bellerophon, a hybrid systems tactics language and library implemented in the theorem prover KeYmaera X [12]. Bellerophon includes a tactics library which provides the decision procedures and heuristics necessary for a productive interactive hybrid systems proving environment. We first demonstrate the interactive verification benefits of Bellerophon through interactive verification of a simple hybrid system, which is optimized to showcase a maximum of features in a minimal example. In the process, we also discuss significant components of the Bellerophon standard library that enable such tactical theorem proving. We then present two examples of proof search procedures implemented in Bellerophon, demonstrating Bellerophon's suitability for implementing reusable proof search heuristics for hybrid systems. Along the way, we demonstrate how the language features of Bellerophon support manual proofs and proof search automation.

2 Background

This section reviews hybrid programs, a programming language for hybrid systems; differential dynamic logic ($\mathsf{d}\mathcal{L}$) [24–27] for specifying reachability properties about hybrid programs; and the theorem prover KeYmaera X for $\mathsf{d}\mathcal{L}$ [12].

Hybrid (dynamical) systems [2, 26] are mathematical models for the interaction between discrete and continuous dynamics, and hybrid programs [24–27] their programming language. The syntax and informal semantics is in Table 2.

The following hybrid program outlines a simple model of a skydiver who deploys a parachute to land at a safe speed. Here, we illustrate the rough program structure to become acquainted with the syntax. We will fill in the necessary details (e.g., when to deploy the parachute exactly) for a proof later in Section 4.

Example 1 (Skydiver model). $\Big(\underbrace{(?\mathit{Dive} \cup r := p)}_{\mathit{ctrl}} \; ; \; \underbrace{\{x' = v,\ v' = f(v,g,r)\}}_{\mathit{plant\ (continuous\ dynamics)}} \Big)^*$

[0] A continuous evolution along the differential equation system $x_i' = \theta_i$ for an arbitrary duration within the region described by formula F. The $\&F$ is optional so that e.g., $\{x' = \theta\}$ is equivalent to $\{x' = \theta \& \mathit{true}\}$.

Example 1 describes a skydiver whose *ctrl* chooses nondeterministically to continue diving if the formula *Dive* indicates it is still safe to do so, or to deploy the parachute ($r := p$). The skydiver's altitude x then follows a differential equation, where the velocity v non-linearly depends on v itself, gravity g and drag coefficient r. This process may repeat arbitrarily many times (indicated by the repetition operator *). Because there is no evolution domain constraint on *plant*, each continuous evolution has any arbitrary non-negative duration $e \in \mathbb{R}$.

Differential dynamic logic ($\mathsf{d}\mathcal{L}$) [24–27] is a first-order multimodal logic for specifying and proving properties of hybrid programs. Each hybrid program α has modal operators $[\alpha]$ and $\langle\alpha\rangle$, which express reachability properties of program α. The formula $[\alpha]\phi$ expresses that the formula ϕ is true in all states reachable by the hybrid program α. Similarly, $\langle\alpha\rangle\phi$ expresses that the formula ϕ is true after some execution of α. The $\mathsf{d}\mathcal{L}$ formulas are generated by the grammar

$$\phi ::= \theta_1 \backsim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \phi \leftrightarrow \psi \mid \forall x\, \phi \mid \exists x\, \phi \mid [\alpha]\phi \mid \langle\alpha\rangle\phi$$

where ϕ and ψ are formulas, α ranges over the hybrid programs of Table 2, and $\backsim$ is a comparison operator $=, \neq, \geq, >, \leq, <$, and θ is a term of real arithmetic.

Model 1 (Safety specification for the skydiver)

$$\underbrace{x \geq 0 \wedge \ldots}_{\textit{initial condition}} \rightarrow [\big(\underbrace{(?\mathit{Dive} \cup r := p)}_{\textit{ctrl}} ;\ \underbrace{\{x' = v, v' = f(v,g,r)\}}_{\textit{plant}}\big)^*] \underbrace{(x{=}0 \rightarrow |v|{\leq}|m|)}_{\textit{post cond.}}$$

The formula above expresses that if the skydiver, among other things, starts diving at some non-negative altitude x, then it is always the case that if they touch ground ($x = 0$) they do so softly with a safe descending speed ($|v|{\leq}|m|$, because both v and m are always negative).

KeYmaera X Bellerophon is part of KeYmaera X, an axiomatic theorem prover for $\mathsf{d}\mathcal{L}$ [12]. Its uniform substitution mechanism [27] enables a trusted core of only about 1,700 lines of Scala. This is far smaller than other hybrid systems verification tools and compares favorably even with many other LCF-style provers. While verified real arithmetic solving is possible via witnesses [30], KeYmaera X uses external real arithmetic solvers in practice for their superior performance.

3 The Bellerophon Tactic Language

Bellerophon is a programming language and standard library for automating proof constructions and proof search operations of the KeYmaera X core. As in other LCF-style provers, Bellerophon is not soundness-critical. This frees us to provide courageous reasoning strategies that enable users to perform high-level proofs about hybrid systems while still benefiting from the high degree of trustworthiness that comes from a small soundness-critical core and the cross-verification of $\mathsf{d}\mathcal{L}$ in Isabelle and Coq [4]. A basic use of Bellerophon is to recover a convenient sequent calculus for $\mathsf{d}\mathcal{L}$ [24] from the simpler Hilbert calculus-based core [27] of KeYmaera X. This demonstrates that Bellerophon is expressive enough to implement the automation capabilities of the predecessor prover

KeYmaera [29] from a smaller set of primitives. Beyond that, Bellerophon is used, e.g. for programming both individual proofs and custom proof search procedures.

This section presents the basic constructs of the Bellerophon language. Readers familiar with tactic languages for interactive theorem provers (e.g., [20]) will find many constructs familiar, but should pay particular attention to the discussion of Bellerophon's standard library. For usability, traceability and educational purposes, Bellerophon tactics can be written in a hierarchical structure that maps to the graphical tree structure of the resulting dℒ sequent proof [21].

This dℒ proof motivates the constructs of our language and standard library.

Proof 1 (Skydiver sequent proof sketch) The proof starts from the initial conjecture (Model 1) at the bottom, phrased as a *sequent*. Each sequent has the shape *assumptions* ⊢ *obligations*, which means from the assumptions left of the turnstile ⊢, we have to prove any formula on the right. Horizontal lines indicate that the sequent below the horizontal line is proved when the sequent above the horizontal line is proved, justified by the tactic that is annotated left of the horizontal bar (the corresponding operator is highlighted in boldface and red). For example, the first step `prop` makes all conjuncts left of an implication available as assumptions, so the goal $x \geq 0 \land B \rightarrow C$ below the line becomes $x \geq 0, B \vdash C$ above the line. When proof rules (e.g., `andR`) result in multiple subgoals, each subgoal continues in a separate branch and all need to be proved.

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dots}{\Gamma \vdash \mathit{Dive} \rightarrow [\mathit{ode}](x{=}0 \rightarrow |v|{\leq}|m|)}}{\Gamma \vdash [\mathbf{?\mathit{Dive}}][\cdots](x{=}0 \rightarrow |v|{\leq}|m|)}\,\text{testb} \quad \dfrac{\dots}{\Gamma \vdash [\mathit{ode}(p)](x{=}0 \rightarrow |v|{\leq}|m|)}\,\text{assignb}}{\Gamma \vdash [?\mathit{Dive}][\cdots](x{=}0 \rightarrow |v|{\leq}|m|) \land [r := p][\cdots](x{=}0 \rightarrow |v|{\leq}|m|)}\,\text{andR}}{\Gamma \vdash [\{?\mathit{Dive} \cup r := p\}][\{p' = v, v' = f(v,g,r)\}](x{=}0 \rightarrow |v|{\leq}|m|)}\,\text{choiceb}}{x \geq 0, \dots \vdash [\{?\mathit{Dive} \cup r := p\};\{x' = v, v' = f(v,g,r)\}](x{=}0 \rightarrow |v|{\leq}|m|)}\,\text{composeb}}{\vdash x \geq 0 \land \dots \rightarrow [\{?\mathit{Dive} \cup r := p\};\{x' = v, v' = f(v,g,r)\}](x{=}0 \rightarrow |v|{\leq}|m|)}\,\text{prop}
$$

Each of the steps in the sequent proof above is a built-in tactic:

`prop` Exhaustively applies propositional proof rules in the sequent calculus.

`composeb` Splits sequential composition $[\alpha;\beta]P$ into nested modalities $[\alpha][\beta]P$.

`choiceb` Splits choice $[\alpha \cup \beta]P$ into a conjunction of subsystems $[\alpha]P \land [\beta]P$.

`andR`, `implyR`, `existsL`, `...` are the right conjunction rule (∧R), the right implication rule (→R) and left existential rule (∃L) as usual in sequent calculus. Throughout the paper, we will make use of standard propositional sequent calculus tactics that follow this naming convention.

`testb` Makes test condition $[?Q]P$ available as assumption $Q \rightarrow P$.

`assignb` Makes effect of assignment $[x := t]P(x)$ available as update to $P(t)$ or as assumption $x = t$ with proper renaming of other occurrences of x.

Bellerophon programs, called tactics, are functions mapping lists of sequents to (lists of[1]) sequents. Built-in tactics (ranged over by τ) are implemented in

[1] Tactics may map a single sequent to a list of sequents; the simplest example of such a tactic `andR` corresponds to the proof rule ∧R, which maps a single sequent $\Gamma \vdash A \land B, \Delta$ to the list of subgoals $\Gamma \vdash A, \Delta$ and $\Gamma \vdash B, \Delta$.

Table 3. Meaning of tactic combinators

Language Primitive	Operational Meaning
$e ::= \tau$	Built-in tactic
$\mid\ e(\overline{v})$	Applies a tactic e to a (list of) positions or formulas
$\mid\ e_1$ `;` e_2	Sequential Composition: Applies e_2 on the output of e_1
$\mid\ e_1 \mid e_2$	Either Composition: Applies e_2 if applying e_1 fails
$\mid\ e^*$	Saturating Repetition: Repeatedly applies e as long as it is applicable (diverging if it stays applicable indefinitely)
$\mid\ ?(e)$	Optional: Applies e if e does not result in an error
$\mid\ <(e_1, e_2, \ldots e_n)$	Applies e_1 to the first of n subgoals, e_2 to the second, etc.
$\mid$ `abbrv` $P(\overline{x}) = \phi$ `in` e	Replaces all occurrences of ϕ with $P(\overline{x})$ in the current subgoal, and then applies e. After e, remaining occurrences of $P(\overline{x})$ are uniformly substituted back to ϕ

Scala. Proof developers can combine existing tactics using the constructs described in Table 3. Built-in programs are implemented as a sequence of operations on a data structure that can only be created or modified by the soundness-critical core of KeYmaera X, thereby ensuring soundness of built-in tactics.

Built-in Tactics. Bellerophon is both a stand-alone language and a domain-specific language embedded in the Scala programming language. Built-in tactics directly manipulate the KeYmaera X core to transform formulas in a validity-preserving manner. Bellerophon programmers can construct new tactics either by writing new built-in tactics in Scala, or else by combining pre-existing tactics using the combinators described in Table 3. KeYmaera X ships with a large library of tactics for proof construction and proof search. Some built-in tactics – the propositional rules and `choiceb` for example – are straight-forward applications of the axioms in [27]. Others provide a significant amount of automation on top of the axiomatic foundations. For example, `prop` combines propositional sequent calculus rules to an automated proof search procedure that often performs numerous simpler proof steps automatically.

Parameters. Most tactics are parameterized by formulas, locators, or both. Formula parameters are provided whenever the behavior of a tactic is dependent upon a particular formula; for example, the loop and differential induction tactics take an invariant formula as parameter. Locators specify where in a sequent a tactic should be applied. The simplest form of locator is an explicit position. Negative positions refer to formulas to the left of the turnstile ($\vdash$) and positive positions refer to formulas to the right of the turnstile,[2] e.g., $-1 : A,\ -2 : B,\ -3 : C \ \vdash\ 1 : D,\ 2 : E$ with annotated formula positions. In addition to explicit positions, Bellerophon provides indirect locators: *(i)* $e($`R`$)$ applies e to the first applicable position[3] in the succedent; *(ii)* $e($`Rlast`$)$ applies e to the last position in the succedent. $e($`L`$)$ and $e($`Llast`$)$ behave accordingly in the antecedent.

[2] The addressing scheme extends to subformulas and subterms in a straight-forward way. Interested readers may refer to the Bellerophon documentation for details.

[3] Tactic e is applicable at a position *pos* if $e(pos)$ does not result in an error.

Basic Combinators. Tactics are executed sequentially using the ; combinator. In $e; f$, the left tactic e is executed on the current subgoal and then the right tactic f is executed on the result of the left tactic's execution. The | combinator attempts multiple tactics – moving from left to right through a list of alternatives. The $*$ combinator in e^* repeats the tactic e as long as it is applicable. Many proof search procedures are expressible as a repetition of choices.
Branching. Proof search often results in branching. For example, a canonical proof of the induction step of Model 1 decomposes into two cases: a diving case corresponding to the control decision ?*Dive* and a deployed parachute case corresponding to the control decision $r := p$. Proof 1 from above in the dL sequent calculus visually emphasizes the branching structure, which can be helpful for structuring tactics too. The $<$ combinator expresses how a proof decomposes into cases. An explicit tactic directly performing Proof 1 without any search is:

Listing 1.1. A Structured Bellerophon Tactic for a Branching Proof

```
prop ; composeb(1) ; choiceb(1) ; andR(1) ; <(
  testb(1) ; ...,          /* tactic for left  branch of andR */
  assignb(1) ; ...         /* tactic for right branch of andR */
)
```

Equivalently, the proof search tactic `unfold` automates proofs such as Listing 1.1 by applying all propositional and dynamical axioms until encountering a loop program or a differential equation, where cleverness might be needed.

4 Demonstration of Tactical Hybrid Systems Proving

In this section, we demonstrate that the Bellerophon standard library's techniques for invariance properties, conservation properties, and real arithmetic simplifications, as implemented in KeYmaera X, make it a convenient mechanism for interactively verifying hybrid systems. The proof developed in this section is at `http://web.keymaeraX.org/show/itp17/skydiver.kya`

Model 2 fills in the details of the skydiver model, which guarantees landing at a safe speed if the parachute opens early enough.

Model 2 (Safety specification for the skydiver model)

$$x \geq 0 \land g > 0 \land 0 < a = r < p \land -\sqrt{\frac{g}{p}} < v < 0 \land m < -\sqrt{\frac{g}{p}} \land T \geq 0 \qquad (\textit{init})$$

$$\rightarrow [\{\ (?\underbrace{\left(r = a \land v - g \cdot T > -\sqrt{\frac{g}{p}}\right)}_{\textit{Dive}}\ \cup\ r := p); \qquad (\textit{ctrl})$$

$$t := 0;\ \{x' = v,\ v' = r \cdot v^2 - g\ \&\ x \geq 0 \land v < 0 \land t \leq T\} \qquad (\textit{plant})$$

$$\}^*](x = 0 \rightarrow |v| \leq |m|) \qquad (\textit{post cond.})$$

Opening the parachute is a discrete control decision. The diver's physics are modeled as an ODE, accounting for both gravity and drag, which changes

when the parachute opens. This example is carefully crafted to demonstrate many of the challenges in hybrid systems reasoning while retaining relatively simple dynamics. Qualitative changes happen to the continuous dynamics after a discrete state transition, the dynamics are non-linear, and the property of interest is not directly inductive.

We model a gravitational force ($g > 0$), a drag coefficient (r) which depends on whether the parachute is closed (air a) vs. open (parachute p), the skydiver's altitude $x \geq 0$ and velocity $v < 0$. The time between control decisions is bounded by the skydiver's reaction time T. We also assume that the diver does not pass through the earth $x \geq 0$ and (to streamline this presentation) that $v < 0$.

The controller contains two options for our skydiver. The left choice lets a closed parachute ($r = a$) stay closed if the speed after one control cycle is definitely safe, computed by over-approximating as if gravity were the only force ($v - g \cdot T > -\sqrt{\frac{g}{p}}$). The right control choice opens the parachute, after which it stays open (as $r \neq a$). For simplicity, we say the parachute opens instantantly.

The safety theorem says when the skydiver hits the ground, the velocity is at most a specified safe landing speed $|v| \leq |m|, v < 0$. We assume the parachute is initially closed ($r = a$), the speed initially safe ($v > -\sqrt{\frac{g}{p}}$), and the safe landing speed faster than the limit speed of the parachute ($m < -\sqrt{\frac{g}{p}}$).

Loop Invariants Verifying a system loop begins with identifying a loop invariant J that is true initially, implies the post-condition and is preserved by the controller. Each formula of the initial condition in Model 2 is invariant except $r = a$; therefore, we will proceed with the following invariant J:

$$\underbrace{(x \geq 0 \wedge v < 0)}_{ev.dom.} \wedge \underbrace{\left(g > 0 \wedge 0 < a < p \wedge T \geq 0 \wedge m < -\sqrt{\frac{g}{p}}\right)}_{diff.\ inductive} \wedge \underbrace{v > -\sqrt{\frac{g}{p}}}_{hard} \quad (1)$$

Note that J holds initially and implies formula $|v| \leq |m|$ because $v > -\sqrt{\frac{g}{p}} > m$. These facts prove automatically. Therefore, the core proof needs to prove $J \rightarrow [ctrl; plant]J$. We express the proof thus far with the following tactic:

Listing 1.2. Loop Induction Tactic

```
1 implyR(1); loop(J, 1); <(QE,QE,nil)
```

The `implyR` tactic corresponds to the right implication rule ($\rightarrow$R) in sequent calculus; the first argument states that we should apply this proof rule at the first position in the succedent. The `loop` tactic uses the dL axioms about loops to derive three new subgoals: (1) the loop invariant holds initially ($init \rightarrow J$); (2) the loop invariant implies the post condition ($J \rightarrow post\ cond.$); and (3) the loop invariant is preserved throughout a single iteration of the loop ($J \rightarrow [ctrl; plant]J$). The `loop` rule in KeYmaera X is derived in Bellerophon from axioms and automatically retains assumptions about constants that do not change in the system. The `nil` tactic has no effect and is used in `<()` to keep subgoal (3) unchanged.

The branching combinator `<()` allows us to isolate each of these three subtasks from one another. Subgoals (1) and (2) are proven using a Real Arithmetic solver (`QE`, for `Q`uantifier `E`limination), since the arithmetic is easy enough here.

Decomposing Control Programs This model's control program is simple. It checks if it is safe to keep the parachute closed, or sets r to open the parachute (at any time, but at the latest when it is no longer safe to keep it closed). Therefore, we will immediately symbolically execute the control program and consider the two resulting subgoals, both of which are reachability conditions on purely continuous dynamical systems. This splitting could be done manually, as in Listing 1.1. But we decide to split it automatically using the `unfold` tactic.

Listing 1.3. Decomposing Control Programs

```
1 implyR(1); loop(J, 1); <(QE,QE,unfold)
```

ODE Tactics in the Standard Library The rest of the proof will make use of several tactics in the Bellerophon standard library:

`boxAnd` Splits $[\alpha](P \wedge Q)$ into separate postconditions $[\alpha]P$ and $[\alpha]Q$.

`dC(R)` Proves a new property R of an ODE and then restricts the differential equation to remain within the evolution domain R (differential cut).

`dW` Proves $[x' = f(x)\&Q]P$ by proving that domain Q implies postcondition P.

`dI` Proves $[\{x' = f(x)\}]P$ by proving P and its differential P' along $x' = f(x)$.

`dG(y'=ay+b,R)` Adds new differential equation $y' = ay+b$ to $[x' = f(x)\&Q]P$, and replaces the post condition by equivalent formula R (possibly mentioning the fresh *differential ghost variable* y).

These tactics perform significant automation on top of the $\mathsf{d}\mathcal{L}$ axioms. For example, `dI` performs automatic differentiation via exhaustive left-to-right rewriting of our axiomatization of differentials (e.g., $(s \cdot t)' = s't + st'$) and propagates the local effect of the differential equation. The `dI` tactic preserves initial value constraints for variables that are not changed by the differential equation. It often performs hundreds of axiom applications automatically. The difference between the sound Differential Induction axiom [27] and the automation provided by the `dI` tactic is an exemplary demonstration of the difference between a theoretically complete mathematical/logical foundation, and a pragmatically useful tactical library.

We are now ready to consider two purely continuous subgoals of the form $J \rightarrow [\mathit{plant}(r)]J$: one where $r = a$ (the parachute is closed) and one where $r = p$ (the parachute is open), which are both true for different reasons.

Closed Parachute: Chaining Inequalities We first consider the $r = a$ case, in which the parachute is closed. Symbolically executing the control program results in a remaining subgoal that requires us to prove:

$$J \wedge v - g \cdot T > -\sqrt{g/p} \rightarrow [\{x' = v,\ v' = a \cdot v^2 - g \& x \geq 0 \wedge v < 0 \wedge t \leq T\}]J$$

We use `boxAnd` to work on the conjuncts of the loop invariant J (1) separately, since each are preserved for different reasons. The proofs for the first two sets of loop invariants in J (labeled *ev. domain* and *diff. induction*) are identical to the $r = p$ case and will be discussed later. Here, we focus on the formula $J \wedge v - g \cdot T > -\sqrt{\frac{g}{p}} \rightarrow [\{x' = v, v' = a \cdot v^2 - g \& x \geq 0 \wedge v < 0 \wedge t \leq T\}] v > -\sqrt{\frac{g}{p}}$, which handles the third conjunct of J (see (1), labeled *hard*).

Compute that $v \geq v_0 - g \cdot t \geq v_0 - g \cdot T > -\sqrt{\frac{g}{p}}$, where v_0 is the value of v before the ODE. In Bellerophon proofs for differential equations, we use $old(v)$ to introduce initial values; you can read $old(v)$ and v_0 inter-changeably here.

Each of the subformulas in the postcondition above is a differentially inductive invariant, or else is valid after the domain constraint is automatically augmented with constants $g > 0 \wedge p > 0$. Therefore, we use a chain of `dC` justified either by `dI` or by `dW` for each inequality in this tactic:

Listing 1.4. A Chain of Inductive Inequalities

```
1 /* Key lemmas                              proofs of lemmas */
2 dC(v>=old(v)-g()*t,1);                   <(nil , dI(1));
3 dC(old(v)-g()*t>=old(v)-g()*T,1); <(nil , dW(1);QE);
4 dC(old(v)-g()*T>-c,1);                   <(nil , dI(1));
5 dW(1) ; QE
```

The argument is a sequence of differential cuts, each of which has a simple proof, and whose conjunction implies the post-condition. Each of the `nil` tactics in the `<()` passes along a single subgoal to the next tactic, so that at the end we have a long conjunction in our domain constraint containing each of the cuts. This style of proof is pervasive in hybrid systems verification, and easily expressed in Bellerophon. One key feature that makes this proof so concise is the use of `old(v)`, which introduces a variable v_0 that remembers the initial value of v. Tactic `dW`; QE on line 3 proves the cut from the evolution domain constraint.[4]

The inequalities in the evolution domain of the differential equation system are now sufficiently strong to guarantee the postcondition, so we use `dW` to obtain a final arithmetic subgoal: $\Gamma \vdash v \geq v_0 - g \cdot t \geq v_0 - g \cdot T > -\sqrt{\frac{g}{p}} \rightarrow v > -\sqrt{\frac{g}{p}}$, where Γ contains constants propagated by the rule `dW` (unlike the DW axiom).

Although this arithmetic fact is obvious to us, `QE` will take a substantial amount of time to prove this property (at least 15 minutes on a 32 core machine running version 10 Mathematica and version 4.3.7 of KeYmaera X). This is a fundamental limitation of Real Arithmetic decision procedures, which have extremely high theoretical and practical complexity [9].

The simplest way to help `QE` is to introduce a simpler formula that captures the essential arithmetic argument: e.g., cut in $\forall a, b, c, d\, (a \geq b \geq c > d \rightarrow a > d)$

[4] The attentive reader will notice we use `g()` instead of g. This is to indicate that the model has an arity 0 function symbol `g()`, rather than an assignable variable. This syntactic convention follows KeYmaera X and its predecessors.

and then instantiate this formula with our chain of inequalities. We take this approach for demonstration (see the implementation). As an alternative, transforming and abbreviating formulas in Bellerophon achieves a similar effect.

Open Parachute: Differential Ghosts We now consider case 2, where the parachute is already open ($r = p$). After executing the discrete program the remaining subgoal is: $J \rightarrow [\{x' = v,\ v' = p \cdot v^2 - g \ \&\ \underbrace{x \geq 0 \land v < 0 \land t \leq T}_{\text{evolution domain constraint}}\}]J$.

The proof proceeds by decomposing the post-condition J into three separate subgoals, one for each conjunct in (1). In Listing 1.5, the `boxAnd` tactic uses axiom $[\alpha](P \land Q) \leftrightarrow [\alpha]P \land [\alpha]Q$ from left to right, to rewrite the instance of $[\alpha](P \land Q)$ to separate corresponding conjuncts $[\alpha]P \land [\alpha]Q$. The first set of formulas in J (labeled *ev. domain*) are *not* differentially inductive, but are trivially invariant because the evolution domain constraint of the system already contains these properties. Differential weakening by `dW` is the appropriate proof technique for these formulas, see line 1 in Listing 1.5. The second set of formulas (labeled *diff. inductive*) are *not* implied by the domain constraint, but are inductive along the ODE because the left and right sides of each inequality have the same time-derivative (0). Differential induction by `dI` is the appropriate proof technique for establishing the invariance of these formulas, see line 2 in Listing 1.5.

Listing 1.5. Differential Weakening and Differential Induction

```
1 boxAnd(1); andR(1); <(dW(1);QE , nil);
2 boxAnd(1); andR(1); <(dI(1) , nil)
```

The third conjunct (labeled *hard*) requires serious effort: we have to show that $v > -\sqrt{\frac{g}{p}}$ is an invariant of the differential equation. This formula is *not* a differentially inductive invariant because it is getting less true over time. To become inductive, we require additional dynamics to describe energy conservation. The Bellerophon library provides a tactic to introduce additional dynamics as *differential ghosts* into a differential equation system. Often, differential ghosts can be constructed systematically. Here, we want to show $v > -c$ where $c = \sqrt{\frac{g}{p}}$, so we need a property with a fresh differential ghost y that entails $v + c > 0$, e.g., $y^2(v + c) = 1$. The formula $y^2(v + c) = 1$ becomes inductively invariant when $y' = -\frac{1}{2}p(v - c)$. In summary, tactic `dG` in Listing 1.6 introduces $y' = -\frac{1}{2}p(v - c)$ into the system and rewrites the post-condition to $y^2(v + c) = 1$ with the additional assumptions that y does not contain any singularities ($p > 0 \land g > 0$).

Listing 1.6. Finishing the parachute open case with a ghost

```
1 dG(y'=-1/2*p*(v-(g()/p)^(1/2)), p>0&g()>0&y^2*(v+c)=1, 1) ;
2 dI(1.0); QE
```

Tactic `dG` results in a goal of the form $\exists y[\cdots](p > 0 \land g > 0 \land y^2(v+c) = 1)$, so in line 2 of Listing 1.6 we apply `dI` at the first child position 1.0 of succedent 1 *in context* of the existential quantifier to show that the new property $y^2(v + c) = 1$ is differentially invariant with the differential ghost y.

If a system avoids possible singularities, the `ODE` tactic in the Bellerophon standard library automatically computes the differential ghost dynamics (here $y' = -\frac{1}{2}p(v-c)$) and postcondition (here $y^2(v+c) = 1$) with the resulting proof. Additionally, notice that `dG` conveniently constructs the axiom instance of DG [27], saving the proof developer from manually constructing such instances.

The proof in Listing 1.6 completes the invariant preservation proof for $r = p$. The full proof artifact for the skydiver demonstrates how Bellerophon addresses each of the major reasoning challenges in a typical hybrid systems verification effort.

5 Automatic Tactics in the Bellerophon Standard Library

This section presents two significant automated tactics in the Bellerophon standard library: a heuristic tactic for invariants of ODEs, and a general-purpose hybrid systems heuristic building upon ODE automation. These tactics use our embedding of Bellerophon as a DSL in Scala, the KeYmaera X host language.[5]

The combination of a tactical language and a general-purpose functional language allow us to more cleanly leverage complicated computations, such as integrators and invariant generators, without losing the high-level proof structuring and search strategy facilities provided by Bellerophon. Significant further Bellerophon programs that ship with KeYmaera X include an automated deduction approach to solving differential equations [27], the proof-guided runtime monitor synthesis algorithm ModelPlex [22] and real arithmetic simplification procedures. KeYmaera X provides an IDE [21] for programming Bellerophon tactics and inspecting their effect in a sequent calculus proof.

The purpose of this section is to explain, at a high level, how Bellerophon provides ways of automating hybrid systems proof search. We only present simplified versions of tactics and briefly discuss relevant implementation details.

5.1 Tactical Automation for Differential Equations

Automated reasoning for ODEs is critical to scalable analysis of hybrid systems. Even when human interaction is required, automation for simple reachability problems – such as reachability for solvable or univariate subsystems – streamlines analysis and reduces requisite human effort.

The skydiver example above illustrated the interplay between finding differential invariants and proving with differential induction and differential ghosts. The tactic `ODE` in the Bellerophon standard library automates this interplay for solvable systems and some unsolvable, nonlinear systems of differential equations, see Listing 1.7. The `ODEStep` tactic directly proves properties by differential induction, with differential ghosts, and from the evolution domain constraints. The `ODEInvSearch` tactic cuts additional differential invariants,

[5] Advanced automation generally uses the EDSL. Programs written in the EDSL are executed using the same interpreter as programs written in pure Bellerophon.

thereby strengthening the evolution domain constraints for `ODEStep` to ultimately succeed. Tactic `ODE` succeeds when `ODEStep` finds a proof; if `ODEStep` does not yet succeed, `ODEInvSearch` provides additional invariant candidates with differential cuts `dC` or by solving the ODE. This interaction between `ODEStep` and `ODEInvSearch` is implemented in Listing 1.7 by mixing recursion and repetition. Repetition is used in `ODE` so that `ODEStep` is prioritized over `ODEInvSearch` each time that a new invariant is added to the system. Recursion is used in `ODEInvSearch` so that a full proof search is started every time an invariant is successfully added to the domain constraint by `dC`. The `ODEInvSearch` tactic calls `ODEStep` on its second subgoal (the "show" branch of the `dC`) because differential cuts can be established in the right order without additional cuts.

Listing 1.7. Automated ODE Tactic for Non-Solvable Differential Equations

```
1 ODEStep(pos)       = dI(pos) | dgAuto(pos) | dW(pos) | ...
2 ODEInvSearch(pos) = dC(nextCandidate); <(ODE(pos), ODEStep(pos))
3                       | solve(pos)
4 ODE(pos)           = ( ODEStep(pos) | ODEInvSearch(pos) )*
```

The `ODEStep` tactic finds a proof with `dI` when the post-condition is differentially inductive, meaning that the vector field of the differential equation points into the set described by the differential equation. The `dgAuto` tactic will also attempt to make properties differentially inductive by constructing differential ghosts for the postcondition, such as the ghosts introduced in the skydiver example. In case the evolution domain of a differential equation system is sufficiently strong, tactic `dW` succeeds from just the evolution domain constraints. The `ODEStep` tactic implemented in KeYmaera X contains other proof search techniques (marked ... above) that are guaranteed to terminate but refrain from performing differential cuts.

The invariant search `ODEInvSearch` constructs candidates for differential invariants heuristically [28], see `dC(nextCandidate)` in Listing 1.7, or systematically for solvable differential equations with `solve`. Tactic `solve` follows an axiomatic ODE solver approach [27] that implements a solver in terms of the differential invariants, cuts, and ghosts reasoning principles to avoid a trusted built-in rule for solving differential equations (such trusted built-in rules are necessary in other hybrid systems tools, e.g., in KeYmaera [29]).

The `ODE` tactic described above is an idealized version of the ODE tactic implemented in KeYmaera X, which contains additional automated search procedures and specializes proof search based upon the shape of the post-condition.

5.2 Tactical Automation for Hybrid Systems

The `solve` and `ODE` tactics provide some automation for continuous systems proofs. The `master` tactic builds on these to provide a full heuristic for hybrid systems in the canonical form $init \rightarrow [\{ctrl;\ plant\}^*]safe$. Tactic `master` combines the three basic reasoning principles that together cover the reasoning

tasks arising in hybrid systems models of the above shape: propositional reasoning, symbolic execution of hybrid programs, and reasoning about loops and differential equations.

Listing 1.8. Proof Search Automation for Hybrid Systems

```
1 master = OnAll(prop | step | close | QE | loop | ODE)*
```

In such proofs, branching is prevalent, e.g., due to non-deterministic choices in programs, as well as loop and differential induction. In the proofs so far we specified explicitly how the proof proceeds on each branch using `<()`. This approach is useful to specifically tailor tactics and provide user insight to certain subgoals. In a general-purpose search tactic, however, we neither know *a priori* how many branches there will be, nor how the specific subgoals on each branch are tackled best. The Bellerophon library lets us specify such general-purpose proof search with tactic alternatives `|`, repetition `*`, and continuing proof search on all branches with `OnAll`. The `prop` tactic is executed first on each subgoal. Running `prop` moves *init* into the antecedent in the initial theorem, but also performs propositional reasoning on each new subgoal generated by the proof. This enables propositional simplifications both after symbolic execution and loop/ differential induction, as well as to uncover propositional truths handled by `close` and thereby avoid potentially expensive arithmetic reasoning in `QE`. The `step` tactic picks the canonical dynamical axioms for a formula (by indexing techniques) and applies it in the canonical direction. For example, when applied to $[\alpha \cup \beta]P$, the step tactic will produce a new subgoal $[\alpha]P \wedge [\beta]P$. The `step` tactic focuses on the portions of a program that do not need any decisions such as invariants for loops or differential equations. The `loop` tactic generates loop invariants [28] and performs loop induction for the outer control loops, whereas `ODE` handles differential equations. The KeYmaera X implementation of `master` contains several optimizations to the ordering of tactics based upon empirical experience.

The `ODE` and `master` tactics demonstrate how Bellerophons's combinators are used to construct proof search procedures out of components available in the Bellerophon standard library.

6 Related Work

The novel contributions of this paper are the design and implementation of a tactics language and library for hybrid systems which have shown themselves to make tactical proving fruitful for realistic hybrid systems verification tasks.

Tactics Programming Languages Tactics combinators appear in many general-purpose proof assistants, such as NuPRL [8], MetaPRL [15], Isabelle [3], Coq [20], and Lean [1]. However, our goals differ: all of the above aim to work for as many proving domains as possible, while we optimize for hybrid systems proving. In pursuing this aim, we have developed a unique, extensive suite of tactical automation for hybrid systems resting on a small trusted core. We integrate

key techniques for continuous systems (ODE solving, invariant generation, and conservation reasoning via differential ghosts) with heuristic simplifications for arithmetic that speed up the use of external real-closed field solvers.

Arithmetic Proving Proving theorems of first-order real arithmetic should not be confused with formalizing real analysis, though both are valuable. General-purpose proof assistants have been used to formalize much of real analysis [5, 14, 19, 31], and in fact some such formalizations [16, 17] have been used to prove the soundness of $\mathsf{d}\mathcal{L}$'s proof calculus on which KeYmaera X and Bellerophon rest [4]. However, the style of proof used is different: like other domains in which general-purpose provers excel, formalized analysis benefits from the forms of automation that these provers do well, such as automatically expanding definitions and applying syntactic simplification rules. Because hybrid systems verification is less definition-heavy and because simplification rules alone (e.g. ring axioms) do not make real arithmetic tractable, real arithmetic proofs face problems for which existing automation is insufficient. Since arithmetic proofs do arise in these provers as well, we believe our techniques to be of broader interest. While we provide new automation for important tasks, this does not preclude us from using existing tactical techniques for the subtasks where they are most appropriate, such as propositional reasoning and decomposing composite hybrid systems.

Tactical Proving Styles A set of patterns and anti-patterns have been proposed for Coq tactic programming in $\mathcal{L}_{\texttt{tac}}$ [6]. The suggestion is to use general-purpose automation as much as possible, conveying any problem-specific details through hints or lemmas. In keeping with this philosophy, the canonical usage of Bellerophon is to provide loop and sequences of differential invariants as hints to the automated `master` tactic. This reduces the proof to arithmetic. At this point the user can steer the proof further, e.g. by using Bellerophon's equational rewriting mechanisms to reduce complex arithmetic to simpler lemmas. This tactical proof-by-hint style can be mixed freely with other styles provided by the KeYmaera X user interface. For example, a user might use the UI to identify and apply an arithmetic simplification, at which point the corresponding tactic is generated automatically. They might then integrate this tactic into a larger proof-search algorithm which then solves similar proof cases automatically.

Analysis Tools for Hybrid Systems Compared with other hybrid system analysis tools such as PHAver [10], SpaceEx [11], and dReach [18], Bellerophon enjoys the ability to handle a broad class of systems from a small trusted core provided by the host prover KeYmaera X. The addition of Bellerophon to KeYmaera X increases the class of systems for which verification is practical by using proof scripting to solve problems that would be too tedious and time-consuming otherwise.

7 Conclusion and Future Work

Bellerophon and its standard library support both interactive and automated theorem proving for hybrid systems. The library provides users with a clean interface for expressing common insights that are essential in hybrid systems veri-

fication tasks. Bellerophon combinators allow users to combine these base tactics in order to implement proofs and proof search procedures. Through Bellerophon, KeYmaera X provides sound tactical theorem proving for hybrid systems.

Bellerophon provides a useful basis upon which further sound hybrid systems verification algorithms can be implemented succinctly. The small core of KeYmaera X is solely responsible for soundness, but provides enough flexibility to reason in many radically different ways about hybrid systems. Bellerophon makes this flexibility easily accessible for programming both high-level hybrid systems verification strategies and concrete case study proofs. Fruitful directions for future work include developing more expressive proof structuring languages and extending the tactic library with more proof techniques that leverage ODE analysis software to produce $\mathsf{d}\mathcal{L}$ proofs.

References

1. de Moura et. al., L.M.: The Lean theorem prover (system description). In: CADE-25. LNCS, vol. 9195, pp. 378–388. Springer (2015)
2. Alur, R., Courcoubetis, C., Henzinger, T.A., Ho, P.: Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In: Grossman et al. [13], pp. 209–229
3. Barras, B., González-Huesca, L.D.C., Herbelin, H., Régis-Gianas, Y., Tassi, E., Wenzel, M., Wolff, B.: Pervasive parallelism in highly-trustable interactive theorem proving systems. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (eds.) Intelligent Computer Mathematics - MKM, Calculemus, DML, and Systems and Projects 2013. LNCS, vol. 7961, pp. 359–363. Springer (2013)
4. Bohrer, R., Rahli, V., Vukotic, I., Völp, M., Platzer, A.: Formally verified differential dynamic logic. In: Certified Programs and Proofs - 6th ACM SIGPLAN Conference, CPP 2017. pp. 208–221. ACM (2017)
5. Boldo, S., Lelay, C., Melquiond, G.: Coquelicot: A user-friendly library of real analysis for Coq. Mathematics in Computer Science 9(1), 41–62 (2015)
6. Chlipala, A.: Certified Programming with Dependent Types - A Pragmatic Introduction to the Coq Proof Assistant. MIT Press (2013)
7. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. 12(3), 299–328 (1991)
8. Constable, R.L., Allen, S.F., Bromley, M., et. al.: Implementing mathematics with the Nuprl proof development system. Prentice Hall (1986)
9. Davenport, J.H., Heintz, J.: Real quantifier elimination is doubly exponential. J. Symb. Comput. 5(1/2), 29–35 (1988)
10. Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. STTT 10(3), 263–279 (2008)
11. Frehse, G., Guernic, C.L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) 23rd CAV, 2011. LNCS, vol. 6806, pp. 379–395. Springer (2011)
12. Fulton, N., Mitsch, S., Quesel, J.D., Völp, M., Platzer, A.: KeYmaera X: An axiomatic tactical theorem prover for hybrid systems. In: Felty, A.P., Middeldorp, A. (eds.) CADE. LNCS, vol. 9195, pp. 527–538. Springer (2015)

13. Grossman, R.L., Nerode, A., Ravn, A.P., Rischel, H. (eds.): Hybrid Systems, LNCS, vol. 736. Springer (1993)
14. Harrison, J.: A HOL Theory of Euclidean Space. In: Hurd, J., Melham, T.F. (eds.) TPHOLs. LNCS, vol. 3603, pp. 114–129. Springer (2005)
15. Hickey, J., Nogin, A., Constable, R.L., Aydemir, B.E., Barzilay, E., Bryukhov, Y., Eaton, R., Granicz, A., Kopylov, A., Kreitz, C., Krupski, V., Lorigo, L., Schmitt, S., Witty, C., Yu, X.: MetaPRL - A modular logical environment. In: Basin, D.A., Wolff, B. (eds.) TPHOLs. LNCS, vol. 2758, pp. 287–303. Springer (2003)
16. Hölzl, J., Immler, F., Huffman, B.: Type classes and filters for mathematical analysis in Isabelle/HOL. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) ITP 2013. LNCS, vol. 7998, pp. 279–294. Springer (2013)
17. Immler, F., Traut, C.: The flow of ODEs. In: Blanchette, J.C., Merz, S. (eds.) ITP 2016. Lecture Notes in Computer Science, vol. 9807, pp. 184–199. Springer (2016), https://doi.org/10.1007/978-3-319-43144-4
18. Kong, S., Gao, S., Chen, W., Clarke, E.M.: dReach: δ-reachability analysis for hybrid systems. In: Baier, C., Tinelli, C. (eds.) Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015. LNCS, vol. 9035, pp. 200–205. Springer (2015)
19. Krebbers, R., Spitters, B.: Type classes for efficient exact real arithmetic in Coq. Logical Methods in Computer Science 9(1) (2011)
20. The Coq development team: The Coq proof assistant reference manual. LogiCal Project (2004), http://coq.inria.fr, version 8.0
21. Mitsch, S., Platzer, A.: The KeYmaera X proof IDE: Concepts on usability in hybrid systems theorem proving. In: FIDE-3. EPTCS, vol. 240, pp. 67–81 (2016)
22. Mitsch, S., Platzer, A.: ModelPlex: Verified runtime validation of verified cyber-physical system models. Form. Methods Syst. Des. 49(1), 33–74 (2016), special issue of selected papers from RV'14
23. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: A Proof Assistant for Higher-order Logic. Springer-Verlag, Berlin, Heidelberg (2002)
24. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reas. 41(2), 143–189 (2008)
25. Platzer, A.: Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics. Springer, Heidelberg (2010)
26. Platzer, A.: Logics of dynamical systems. In: LICS. pp. 13–24. IEEE (2012)
27. Platzer, A.: A complete uniform substitution calculus for differential dynamic logic. J. Autom. Reas. 59(2), 219–266 (2017)
28. Platzer, A., Clarke, E.M.: Computing differential invariants of hybrid systems as fixedpoints. Form. Methods Syst. Des. 35(1), 98–120 (2009), special issue for selected papers from CAV'08
29. Platzer, A., Quesel, J.: KeYmaera: A hybrid theorem prover for hybrid systems (system description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 171–178. Springer (2008)
30. Platzer, A., Quesel, J., Rümmer, P.: Real world verification. In: Schmidt, R.A. (ed.) CADE'22. LNCS, vol. 5663, pp. 485–501. Springer (2009)
31. Solovyev, A., Hales, T.C.: Formal Verification of Nonlinear Inequalities with Taylor Interval Approximations. In: Brat, G., Rungta, N., Venet, A. (eds.) NASA Formal Methods. LNCS, vol. 7871, pp. 383–397. Springer (2013), https://doi.org/10.1007/978-3-642-38088-4