Form Methods Syst Des (2009) 35: 98–120
DOI 10.1007/s10703-009-0079-8

Computing differential invariants of hybrid systems as fixedpoints

André Platzer · Edmund M. Clarke

Published online: 10 July 2009

Abstract We introduce a fixedpoint algorithm for verifying safety properties of hybrid systems with differential equations whose right-hand sides are polynomials in the state variables. In order to verify nontrivial systems without solving their differential equations and without numerical errors, we use a continuous generalization of induction, for which our algorithm computes the required *differential invariants*. As a means for combining local differential invariants into global system invariants in a sound way, our fixedpoint algorithm works with a compositional verification logic for hybrid systems. With this compositional approach we exploit locality in system designs. To improve the verification power, we further introduce a *saturation procedure* that refines the system dynamics successively with differential invariants until safety becomes provable. By complementing our symbolic verification algorithm with a robust version of numerical falsification, we obtain a fast and sound verification procedure. We verify roundabout maneuvers in air traffic management and collision avoidance in train control and car control.

Keywords Verification of hybrid systems · Differential invariants · Verification logic · Fixedpoint engine

1 Introduction

Reachability questions for systems with complex continuous dynamics are among the most challenging problems in verifying embedded systems. Hybrid systems [1, 9, 12, 16] are models for these systems with interacting discrete and continuous transitions, with the latter being governed by differential equations. For simple systems whose differential equations have solutions that are polynomials in the state variables, quantifier elimination over real-closed fields [6] can be used for verification [2, 12, 23, 25]. Unfortunately, this symbolic approach does not scale to systems with complicated differential equations whose solutions

A. Platzer (✉) · E.M. Clarke
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA
e-mail: aplatzer@cs.cmu.edu

do not support quantifier elimination (e.g., when they are transcendental functions) or cannot be given in closed form.

Numerical or approximation approaches [3, 10, 26] can deal with more general dynamics. However, numerical or approximation errors need to be handled carefully as they easily cause unsoundness [26]. More specifically, we have shown previously that even single image computations of fairly restricted classes of hybrid systems are undecidable by numerical computation [26]. Thus, numerical approaches can be used for falsification but not (ultimately) for verification. Further, numerical approaches suffer from the curse of dimensionality: Numerical discretizations of the state space grow exponentially in the number of variables.

In this article, we present an approach that combines the soundness of symbolic approaches [2, 12, 25] with support for nontrivial dynamics that is classically more dominant in numerical approaches [3, 10, 26]. During continuous transitions, the system follows a solution of its differential equation. But for nontrivial dynamics, these solutions are much more complicated than the original equations. Solutions quickly become transcendental even if the differential equations are linear. For instance, the solutions of the system $x' = -y$, $y' = x$ are trigonometric functions that can simulate Turing machines. To overcome this, we handle continuous transitions based on their local vector fields, which are described by their differential equations. We use *differential induction* [24], a continuous generalization of induction that works with the differential equations themselves instead of their solutions. For the induction step, we use a condition that can be checked easily based on *differential invariants* [24], i.e., properties whose derivative holds true in the direction of the vector field of the differential equation. The derivative is a directional derivative in the direction of (the vector field generated by) the differential equation, and we generalize derivatives from functions to formulas appropriately. For this to work in practice, the most crucial steps are to find sufficiently strong local differential invariants for differential equations and compatible global invariants for the hybrid system.

To this end, we introduce a *sound* verification algorithm for hybrid systems that computes the differential invariants and system invariants in a fixedpoint loop. We follow the invariants as fixedpoints paradigm [5] using a verification logic that is generalized to hybrid systems accordingly [25]. For combining multiple local differential invariants into a global invariant in a sound way, we exploit the closure properties of the underlying verification logic [25] by forming appropriate logical combinations of multiple local safety statements. For instance, safety properties of a complex system correspond to a conjunction of local safety properties of subsystems in the logic. In addition, we introduce a *differential saturation process* that refines the hybrid system dynamics successively with auxiliary differential invariants until the safety statement becomes an invariant of the refined system. Finally, each fixedpoint iteration of our algorithm can be combined with numerical falsification to accelerate the overall symbolic verification in a sound way. Indeed, numerical falsification can be used to accelerate both local fixedpoint iterations for differential invariants and global fixedpoint iterations for global system invariants. We validate our algorithm by verifying *aircraft roundabout maneuvers* [26, 35], train control applications [30], and car speed controllers [7].

The major contribution in this work is the fixedpoint algorithm for computing differential invariants and its coupling with a differential saturation process. We show that it can verify realistic applications that were out of scope for related invariant approaches [32–34] or other approaches [12, 16, 23], both for theoretical reasons [24, 25] and for scalability issues. Our algorithm verifies collision avoidance for 5 aircraft in a system with nontrivial curved flight dynamics in 28 continuous dimensions.

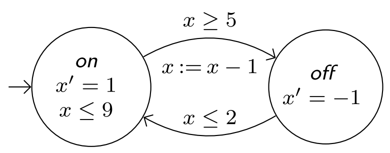
This is an extended version of previous work at CAV [27]. In this article, we extend our previous work [27] by providing actual proofs for our theorems, by including more extensive explanations, illustrations and by giving more examples. We present new detail on the fixedpoint verification algorithm, especially on the handling of existential parameter choices and we add a global fixedpoint algorithm. We further provide additional experimental verification results for a car case study.

2 Hybrid programs and differential dynamic logic

As operational models for hybrid systems, we use *hybrid programs* (HP), a program notation for hybrid automata (HA) [16]. HP can be decomposed syntactically into *fragments*: subprograms which correspond to partial executions of only a part of the full HP (programs are easier to split structurally into parts than graphs, because handling dangling edges between graph fragments is complicated). HP have a perfectly compositional semantics: The semantics of a compound HP is a simple function of the semantics of its fragments. This is important as our verification algorithm recursively decomposes an HP into fragments $\alpha_1, \ldots, \alpha_n$ (e.g., to find local invariants for each α_i) and recombines corresponding correctness statements about these fragments α_i later. We exploit the compositional relationship between the semantics of α and its fragments for our compositional verification approach.

Hybrid programs In order to represent HA [16] textually as an HP, we represent each discrete and continuous transition as a sequence of statements, with a nondeterministic choice ($\cup$) between these transitions. Consider a simple water tank system (Fig. 1) where x denotes the current water level. For instance, the second line in the HP of Fig. 1 represents a continuous transition. It tests (denoted by $?q = on$) if the current location q is on, and then fills the tank by following the differential equation $x' = 1$ restricted to invariant region $x \le 9$ (i.e., the conjunction $x' = 1 \wedge x \le 9$). The third line tests the guard $x \ge 5$ when in state on, then resets x by a discrete assignment ($x := x - 1$), and then changes location q to off. The additional test $?x \le 9$ in the fourth line is needed, because the HA is only allowed to enter mode on when its invariant region $x \le 9$ is satisfied. The * at the end of the HP indicates that the transitions of a HA repeat indefinitely. Alternatively, the resulting HP in Fig. 1 can be considered as the essential part of a program exported from Stateflow/Simulink enriched with differential equations for the continuous plant dynamics. Every safety property that this HP satisfies is fulfilled for *all* deterministic implementation refinements.

Formally, let V be a set of state variables of the system and auxiliary variables. As *terms* we allow polynomials over the rationals with variables in V. To make a structural decomposition of HP into fragments possible, each operation of a HP only has a single effect. There are separate classes of program statements with purely discrete effect, purely continuous effect, and statements for regulating their interaction. *Hybrid programs (HP)* are built with the statements in Table 1 (where H is a formula of first-order real arithmetic, which we



$q := on$; /* initial location is on */
$($ $(?q = on;\ \ x' = 1 \wedge x \le 9)$
$\cup$ $(?q = on \wedge x \ge 5;\ \ x := x - 1;\ \ q := off)$
$\cup$ $(?q = off;\ \ x' = -1)$
$\cup$ $(?q = off \wedge x \le 2;\ \ q := on;\ \ ?x \le 9)\big)^*$

Fig. 1 Natural hybrid program rendition of hybrid automaton (simple water tank)

Table 1 Statements and (informal) effects of hybrid programs (HP)

Notation	Operation	Effect
$x := \theta$	discrete assignment	assigns term θ to variable $x \in V$
$x := random$	nondet. assignment	assigns any real value to $x \in V$
$x_1' = \theta_1 \wedge x_2' = \theta_2 \wedge \dots$ $\cdots \wedge x_n' = \theta_n \wedge H$	continuous evolution	differential equations for $x_i \in V$, terms θ_i with arithmetic constraint H (domain)
$?H$	state check	test formula H at current state
$\alpha; \beta$	seq. composition	HP β starts after HP α finishes
$\alpha \cup \beta$	nondet. choice	choice between alternatives HP α or HP β
α^*	nondet. repetition	repeats HP α n-times for any $n \in \mathbb{N}$

can assume to be quantifier-free using quantifier elimination in real-closed fields [6]). The effect of $x := \theta$ is an instantaneous discrete jump assigning θ to x. Instead, $x := random$ randomly assigns *any* real value to x by a nondeterministic choice. During a continuous evolution $x_1' = \theta_1 \wedge \cdots \wedge x_n' = \theta_n \wedge H$, *all* conjuncts need to hold. Its effect is a continuous transition controlled by the differential equation system $x_1' = \theta_1, \dots, x_n' = \theta_n$ that always satisfies the arithmetic constraint H (thus remains in the region described by H). This directly corresponds to a continuous evolution mode of a HA. The effect of state check $?H$ is a *skip* (i.e., no change) if formula H is true in the current state and that of *abort*, otherwise. The non-deterministic choice $\alpha \cup \beta$ expresses alternatives in the behavior of the hybrid system. Sequential composition $\alpha; \beta$ expresses a behavior in which β starts after α finishes (as usual, β never starts if α continues indefinitely). Non-deterministic repetition α^*, repeats α an arbitrary number of times, possibly zero. All HA can be represented as HP. Further, all classical discrete control structures can be defined in terms of the operations of Table 1, for instance:

$$\text{if } H \text{ then } \alpha \text{ else } \beta \equiv (?H; \alpha) \cup (?\neg H; \beta)$$

$$\text{while } H \text{ do } \alpha \equiv (?H; \alpha)^*; ?\neg H$$

Formulas of d$\mathcal{L}$ Our verification algorithm repeatedly decomposes and recombines HP. As a logical framework where these operations are sound and have a well-defined semantics, we use a logic in which simultaneous correctness properties about multiple subsystems are expressible. The *differential dynamic logic* d$\mathcal{L}$ [25] is an extension of first-order logic over the reals with modal formulas like $[\alpha]\phi$, which is true iff all states reachable by following the transitions of HP α satisfy property ϕ (*safety*).

Definition 1 (d$\mathcal{L}$ formulas) The *formulas of* d$\mathcal{L}$ are defined by the following grammar (where θ_1, θ_2 are terms, $\sim \in \{=, \leq, <, \geq, >\}$, ϕ, ψ are formulas, $x \in V$, and α is an HP built from the statements in Table 1):

$$\text{Formula} ::= \theta_1 \sim \theta_2 \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \phi \rightarrow \psi \mid \forall x \phi \mid \exists x \phi \mid [\alpha]\phi.$$

A Hoare-triple $\{\psi\}\alpha\{\phi\}$ can be expressed as the formula $\psi \rightarrow [\alpha]\phi$, which is true iff all states reachable by HP α satisfy ϕ when starting from an initial state that satisfies ψ. For initial states that satisfy ψ, the modal formula $[\alpha]\phi$ in $\psi \rightarrow [\alpha]\phi$ expresses that all states reachable by following α satisfy ϕ. For initial states that do not satisfy ψ, the formula $\psi \rightarrow [\alpha]\phi$ states nothing because the condition ψ is false.

For instance, let *wctrl* abbreviate the body of the loop of the HP for the water controller from Fig. 1 such that $q := on; (wctrl)^*$ corresponds to the full HP from Fig. 1. Then the following d$\mathcal{L}$ formula states that the water level is always below 10 when it starts at level $x \le 3$:

$$x \le 3 \to [q := on; (wctrl)^*]\, x < 10. \tag{1}$$

Unlike Hoare-logics, dynamic logics are closed under logical connectives [15]. Hence, we can express simultaneous correctness statements about multiple fragments α_i using conjuncts $[\alpha_1]\phi_1 \wedge [\alpha_2]\phi_2$. With this, a proof for a property $[\alpha]\phi$ can be decomposed soundly into $[\alpha_1]\phi_1 \wedge [\alpha_2]\phi_2$, when the formulas $[\alpha]\phi$ and $[\alpha_1]\phi_1 \wedge [\alpha_2]\phi_2$ are equivalent for appropriate fragments α_i of α and subproperties ϕ_i of ϕ. In turn, if a verification algorithm with input $[\alpha_i]\phi_i$ yields formulas $\tilde{\phi}_i$ as output, recursively, these $\tilde{\phi}_i$ can be recombined soundly to the overall verification result $\tilde{\phi}_1 \wedge \tilde{\phi}_2$ for $[\alpha]\phi$. By the semantics of d$\mathcal{L}$, this process gives a *sound* way of combining local invariants required in the respective subgoals $[\alpha_i]\phi_i$ to a global system invariant.

Finally, d$\mathcal{L}$ and its proof techniques are closed under quantification, which we use to quantify over parameter choices of local invariants. For example, $\exists p([\alpha_1]\phi_1 \wedge [\alpha_2]\phi_2)$ can be used to determine if there is a common choice for parameter p that makes both subgoals $[\alpha_i]\phi_i$ about subsystems α_i true at once. Note that this formula states the existence of a parameter choice making two reachability statements $[\alpha_1]\phi_1$ and $[\alpha_2]\phi_2$ true at the same time.

Semantics The semantics of d$\mathcal{L}$ and HP is a Kripke semantics in which states of the Kripke model are states of the hybrid system. A state is a map $\nu : V \to \mathbb{R}$ assigning real numbers to all variables; the set of all states is denoted by States. We write $\nu \models \phi$ if formula ϕ is true at state ν (Definition 3 below). Likewise, $[\![\theta]\!]_\nu$ denotes the real value of term θ at state ν. The semantics of HP α is captured by the state transitions that are possible by running α (reachable state semantics). For continuous evolutions, the transition relation holds for pairs of states that can be interconnected by a continuous flow respecting the differential equation and invariant region. That is, there is a continuous transition along $x' = \theta \wedge H$ from state ν to state ω, if there is a solution of the differential equation $x' = \theta$ that starts in state ν and ends in ω and that always remains within the region H during its evolution. As in [9, 16], we assume non-zero behavior, for simplicity.

Definition 2 (Transition system of hybrid programs) The *transition relation*, $\rho(\alpha)$, of HP α, specifies which state ω is reachable from a state ν by operations of α and is defined as follows

1. $(\nu, \omega) \in \rho(x := \theta)$ iff the state ω is identical to state ν except that $\omega(x) = [\![\theta]\!]_\nu$.
2. $(\nu, \omega) \in \rho(x := random)$ iff the state ω agrees with state ν except for the value of x, which can assume an arbitrary real value in ω.
3. $(\nu, \omega) \in \rho(x_1' = \theta_1 \wedge \cdots \wedge x_n' = \theta_n \wedge H)$ iff for some $r \ge 0$, there is a (*flow*) function $\varphi : [0, r] \to$ States with $\varphi(0) = \nu$, $\varphi(r) = \omega$, such that:
   - The differential equation holds, i.e., for each variable x_i and each time $\zeta \in [0, r]$,
   $$\frac{\mathrm{d}[\![x_i]\!]_{\varphi(t)}}{\mathrm{d}t}(\zeta) = [\![\theta_i]\!]_{\varphi(\zeta)}.$$
   - The value of other variables $y \notin \{x_1, \ldots, x_n\}$ remains constant: $[\![y]\!]_{\varphi(\zeta)} = [\![y]\!]_{\varphi(0)}$ for each $\zeta \in [0, r]$.

– The invariant is always respected, i.e., $\varphi(\zeta) \models H$ for each $\zeta \in [0, r]$.

4. $\rho(\alpha \cup \beta) = \rho(\alpha) \cup \rho(\beta)$
5. $\rho(\alpha; \beta) = \{(\nu, \omega) : (\nu, z) \in \rho(\alpha), (z, \omega) \in \rho(\beta)$ for a state $z\}$
6. $(\nu, \omega) \in \rho(\alpha^*)$ iff there are an $n \in \mathbb{N}$ and states $\nu = \nu_0, \ldots, \nu_n = \omega$ such that $(\nu_i, \nu_{i+1}) \in \rho(\alpha)$ for all $0 \leq i < n$.

Definition 3 (Interpretation of d$\mathcal{L}$ formulas) The *interpretation* $\models$ of a d$\mathcal{L}$ formula with respect to state ν uses the standard meaning of first-order logic:

1. $\nu \models \theta_1 \sim \theta_2$ iff $[\![\theta_1]\!]_\nu \sim [\![\theta_2]\!]_\nu$ for $\sim \in \{=, \leq, <, \geq, >\}$
2. $\nu \models \phi \wedge \psi$ iff $\nu \models \phi$ and $\nu \models \psi$, accordingly for $\neg, \vee, \rightarrow, \leftrightarrow$
3. $\nu \models \forall x \phi$ iff $\omega \models \phi$ for all states ω that agree with state ν except for the value of x
4. $\nu \models \exists x \phi$ iff $\omega \models \phi$ for some state ω that agrees with state ν except for the value of x

It extends to correctness statements about a HP α using the transition relation $\rho(\alpha)$:

5. $\nu \models [\alpha]\phi$ iff $\omega \models \phi$ for all states ω with $(\nu, \omega) \in \rho(\alpha)$.

3 Inductive verification by combining local fixedpoints

For verifying safety properties of hybrid systems without having to solve their differential equations, we use a continuous form of induction. In the induction step, we use a condition on directional derivatives in the direction of the vector field generated by the differential equation. The resulting properties are invariants of the differential equation (whence called *differential invariants* [24]). The crucial step for verifying discrete systems by induction is to find sufficiently strong invariants (e.g., for loops α^*). Similarly, the crucial step for verifying dynamical systems (which correspond to a single continuous mode of a hybrid system) by induction is to find sufficiently strong invariant properties of the differential equation. Consequently, for verifying hybrid systems inductively, local invariants need to be found for each differential equation and a global system invariant needs to be found that is compatible with all local invariants.

To compute the required invariants and differential invariants, we combine the invariants as fixedpoints approach from [5] with the lifting of verification logics to hybrid systems from [25]. We introduce a verification algorithm that computes invariants of a system as fixedpoints of safety constraints on subsystems. In order to obtain a local algorithm that works by decomposing global properties of HP into local properties of subsystems, we exploit compositionality of HP and closure properties of d$\mathcal{L}$: HP can be decomposed into subsystems easily and d$\mathcal{L}$ can combine safety statements about multiple subsystems simultaneously.

3.1 Verification by symbolic decomposition

A *safety statement* corresponds to a d$\mathcal{L}$ formula $\psi \rightarrow [\alpha]\phi$ with an HP α, a safety property ϕ about its reachable states, and an arithmetic formula ψ that characterizes the set of initial states symbolically. *Validity* of formula $\psi \rightarrow [\alpha]\phi$ (i.e., truth in all states) corresponds to ϕ being true in all states reachable by HP α from initial states that satisfy ψ [25]. Our verification algorithm defines the function *prove*($\psi \rightarrow [\alpha]\phi$) for verifying this safety statement recursively.

```
1   function prove($\psi \rightarrow [x := \theta]\phi$):
2     return prove($\psi \wedge \hat{x} = \theta \rightarrow \phi_x^{\hat{x}}$) where $\hat{x}$ is a new auxiliary variable
3   function prove($\psi \rightarrow [?H]\phi$):
4     return prove($\psi \wedge H \rightarrow \phi$)
5   function prove($\psi \rightarrow [\alpha \cup \beta]\phi$):
6     return prove($\psi \rightarrow [\alpha]\phi$) and prove($\psi \rightarrow [\beta]\phi$) /* thus $\psi \rightarrow [\alpha]\phi \wedge [\beta]\phi$ */
7   function prove($\psi \rightarrow [\alpha;\beta]\phi$):
8     return prove($\psi \rightarrow [\alpha][\beta]\phi$)
9   function prove($\psi \rightarrow [x := random]\phi$):
10    return prove($\psi \rightarrow \forall x\,\phi$)
11  function prove($\psi \rightarrow \phi$) where isFirstOrder($\phi$):
12    return QuantifierElimination($\psi \rightarrow \phi$)
13  function prove($\psi \rightarrow Qx\,\phi$) where $Qx$ is $\forall x$ or $\exists x$:
14    return QuantifierElimination($\psi \rightarrow Qx\,$prove$(\phi)$)
```

Fig. 2 d$\mathcal{L}$-based verification by symbolic decomposition

The cases of function *prove* where d$\mathcal{L}$ [25] enables us to verify a property of an HP directly by decomposing it into a property of its parts are shown in Fig. 2. For a concise presentation, the case in line 1 introduces an auxiliary variable $\hat{x}$ to handle discrete assignments by substituting the auxiliary variable $\hat{x}$ for the new value of x in $\phi_x^{\hat{x}}$. By $\phi_x^{\hat{x}}$ we denote the result of substituting $\hat{x}$ for x in ϕ. For instance, $x \leq 3 \rightarrow [x := x - 1]x < 10$ is shown by proving $x \leq 3 \wedge \hat{x} = x - 1 \rightarrow \hat{x} < 10$. Our implementation in the verification tool KeYmaera [29] uses optimizations based on nested modalities in dynamic logic to avoid auxiliary variables [25]. State checks $?H$ are shown by assuming the test succeeds, i.e., H holds true (line 3), because there is nothing to show for failed tests, as $?H$ will not allow any transitions when H is false. Nondeterministic choices split into their alternatives (line 5): To show that $\psi \rightarrow [\alpha \cup \beta]\phi$ is valid (i.e., that all transitions choosing between α and β from initial states satisfying ψ lead to states satisfying ϕ), our algorithm shows, instead, that all transitions along α satisfy this property and—independently—all transitions along β do. Sequential compositions are proven using nested modalities (line 7): The states that we reach by following all transitions of the sequential composition $\alpha; \beta$ (in $[\alpha; \beta]\phi$) are the same as the states reachable by following all transitions of β (in $[\beta]\phi$) from any state reachable by following any transition of α (in $[\alpha][\beta]\phi$). Random assignments are proven by universal quantification (line 9), because all real numbers could be assigned to x when following *all* possible transitions of assigning a random value to x, nondeterministically.

The base case in line 11, where ϕ is a formula of first-order real arithmetic, can be proven by quantifier elimination over real-closed fields [6]. Despite its complexity, this can remain feasible, because the formulas resulting from our algorithm do not depend on the complicated solutions of differential equations but only their right-hand sides. Further, the decompositions in our verification algorithm generally lead to local properties with lower complexities. Using a temporary form of Skolemization together with Deskolemization, quantifier elimination can even be lifted to eliminate quantifiers from d$\mathcal{L}$ formulas [25]. We present here only a simplified treatment of quantifiers in d$\mathcal{L}$ formulas like $\forall x \phi$ that contain modal subformulas: The *prove* function is applied recursively to the unquantified kernel ϕ first and the resulting formula is handled by quantifier elimination (line 13). More generally, quantifier elimination can be lifted to quantifiers in d$\mathcal{L}$ formulas like $\forall x \phi$ as follows. After introducing a fresh Skolem term s for variable x in ϕ, the analysis continues with the unquantified kernel ϕ_x^s. Later on when the formulas resulting from recursive application of *prove* contain no more modalities, the quantifier for s can be reintroduced (Deskolemization) and eliminated equivalently using quantifier elimination in real-closed fields. Handling

$\exists x\phi$ for modal formulas ϕ is similar. The crucial part is that Skolem term dependencies can be exploited to prevent unsound quantifier rearrangements; see previous work [25].

Overall, the algorithm in Fig. 2 recursively reduces safety of HP to separate properties of continuous evolutions or of repetitions, which we verify in the next sections.

3.2 Discrete and differential induction, differential invariants

In the sequel, we present algorithms for verifying loops in $\psi \to [\alpha^*]\phi$ by discrete induction and continuous evolutions in $\psi \to [x' = \theta \land H]\phi$ by differential induction, which is a continuous form of induction. In either case, we prove that an invariant F holds initially (in the states characterized symbolically by ψ, thus $\psi \to F$ is valid) and finally entails the postcondition ϕ (i.e., $F \to \phi$). The cases differ in their induction step.

Definition 4 (Discrete induction) Formula F is a *(discrete inductive) invariant* of $\psi \to [\alpha^*]\phi$ iff the following formulas are valid:

1. $\psi \to F$ (induction start), and
2. $F \to [\alpha]F$ (induction step).

An invariant is *sufficiently strong* if, further, $F \to \phi$ is valid.

Definition 5 (Continuous invariants) Let $\mathcal{D}$ be a differential equation system and H a first-order formula. Formula F is a *continuous invariant* of $\psi \to [\mathcal{D} \land H]\phi$ iff the following formulas are valid:

1. $\psi \land H \to F$ (induction start), and
2. $F \to [\mathcal{D} \land H]F$ (induction step).

Again, a continuous invariant is *sufficiently strong* if $F \to \phi$ is valid.

Note that the presence of the evolution domain H in the continuous dynamics enables us to weaken case 1of Definition 5 to assume H for the induction start as there are no transitions along $x' = \theta \land H$ (hence nothing to show) unless H is true in the beginning.

To prove that F is a continuous invariant, it is sufficient to check a condition on the directional derivatives of all terms of the formula, which expresses that no atomic subformula of F changes its truth-value along the dynamics of the differential equation [24]. This condition is much easier to check than a reachability property ($F \to [\mathcal{D} \land H]F$) of a differential equation. Yet, applications like aircraft maneuvers need invariants with multiple mixed equations and inequalities. Thus, we generalize directional derivatives from functions to logical formulas.

Definition 6 (Differential induction) Let $\mathcal{D}$ be the differential equation system $x_1' = \theta_1 \land \cdots \land x_n' = \theta_n$ and H a first-order formula. A quantifier-free first-order formula F is a *differential invariant* of $\psi \to [\mathcal{D} \land H]\phi$ iff the following formulas are valid:

1. $\psi \land H \to F$ (induction start), and
2. $H \to \nabla_{\mathcal{D}} F$ (differential induction step)

where $\nabla_{\mathcal{D}} F$ is defined as the conjunction of all directional derivatives of atomic formulas in F in the direction of the vector field of $\mathcal{D}$:

$$\nabla_{\mathcal{D}} F \equiv \bigwedge_{(b \sim c) \in F} (\nabla_{\mathcal{D}} b \sim \nabla_{\mathcal{D}} c) \quad \text{for} \sim \in \{=, \geq, >, \leq, <\}$$

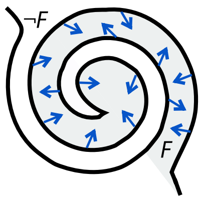


Fig. 3 Differential invariant F

For a term c, the *directional derivative* $\nabla_{\mathcal{D}} c$ is defined in terms of the partial derivatives $\frac{\partial c}{\partial x_i}$ of c by x_i as

$$\nabla_{\mathcal{D}} c := \sum_{i=1}^{n} \frac{\partial c}{\partial x_i} \theta_i .$$

These partial derivatives of terms are well-defined in the Euclidean space spanned by the variables and can be computed symbolically.

The region corresponding to a differential invariant F is illustrated in Fig. 3. Formula $\nabla_{\mathcal{D}} F$ is a directional derivative of F in the direction of the dynamics of $\mathcal{D}$. Intuitively, formula $\nabla_{\mathcal{D}} F$ is true if the gradient arrows of the dynamics are pointing inside/transversal to the (possibly unbounded) region consisting of the points where F is true but never outside. We refer to [24] for the general theory of differential invariants.

The central property of differential invariants for verification purposes is that they can be used to replace infeasible or even impossible reachability analysis with feasible symbolic computation.

Proposition 1 (Principle of differential induction [24]) *All differential invariants are continuous invariants.*

Before we show a proof of Proposition 1, we illustrate its use in a simple example. Consider the dynamics $x' = x^2 \wedge y' = -3$. Differential invariants can be used to show that $3x \geq 4y$ is an invariant for this dynamics without using any state based reachability verification. We just compute symbolically

$$\begin{aligned}\nabla_{x'=x^2 \wedge y'=-3}(3x \geq 4y) &\equiv \frac{\partial 3x}{\partial x} x^2 + \frac{\partial 3x}{\partial y}(-3) \geq \frac{\partial 4y}{\partial x} x^2 + \frac{\partial 4y}{\partial y}(-3) \\ &\equiv 3x^2 \geq -12.\end{aligned}$$

Since the latter formula is easily found to be valid, $3x \geq 4y$ is proven to be a differential invariant and thus remains true whenever it holds true initially (case 1 of Definition 6).

Note that—contrary to common suggestions—condition 2 in Definition 6 can *not* be relaxed to the border of F! For the region F defined as $x^2 \leq 0$, the relaxed condition $\nabla_{x'=5}(x^2 \leq 0)$, i.e., $(2x)5 \leq 0$, would hold on the border $x = 0$ of $x^2 \leq 0$. But $x^2 \leq 0$ clearly is no invariant property of the dynamics $x' = 5$. Thus, we have defined differential invariance conditions to avoid such unsound reasoning carefully. Unfortunately, the counterexample carries over to other approaches [14, 31], revealing soundness issues there.

For the proof of Proposition 1, we first prove a result showing that the formal directional derivative $\nabla_{\mathcal{D}} F$ of formula F as defined in Definition 6 is a generalization of standard function derivatives. We show that the directional derivatives $\nabla_{\mathcal{D}} c$ of terms c in the direction of the vector field of $\mathcal{D}$, which are used for forming $\nabla_{\mathcal{D}} F$, agree with the standard differentiation. That is, they agree with the differentiation in the Euclidean real space of the value of these terms along a flow solving the corresponding differential equation $\mathcal{D}$.

Lemma 1 *Let $\mathcal{D} \wedge H$ be a continuous evolution and let $\varphi : [0,r] \to \mathrm{States}$ be a corresponding flow of duration $r > 0$ (case 2 of Definition 2). Then for all terms c and all $\zeta \in [0,r]$ we have the identity*

$$\frac{\mathrm{d}[\![c]\!]_{\varphi(t)}}{\mathrm{d}t}(\zeta) = [\![\nabla_{\mathcal{D}} c]\!]_{\varphi(\zeta)}.$$

In particular, $[\![c]\!]_{\varphi(t)}$ is continuously differentiable.

Proof The proof is by induction on term c. Let $\mathcal{D}$ be $x_1' = \theta_1 \wedge \cdots \wedge x_n' = \theta_n$.

- If c is one of the variables x_j for some j (for other variables, the proof is simple because c is constant during φ) then:

$$\frac{\mathrm{d}[\![x_j]\!]_{\varphi(t)}}{\mathrm{d}t}(\zeta) = [\![\theta_j]\!]_{\varphi(\zeta)} = \left[\!\!\left[\sum_{i=1}^{n} \frac{\partial x_j}{\partial x_i} \theta_i \right]\!\!\right]_{\varphi(\zeta)}.$$

  The first equation holds by Definition 2. The last equation holds as $\frac{\partial x_j}{\partial x_j} = 1$ and $\frac{\partial x_j}{\partial x_i} = 0$ for $i \neq j$. The derivatives exist because φ is (continuously) differentiable.
- If c is of the form $a + b$, the desired result can be obtained by using the properties of derivatives and interpretations:

$$\begin{aligned}
&\frac{\mathrm{d}[\![a+b]\!]_{\varphi(t)}}{\mathrm{d}t}(\zeta) \\
&= \frac{\mathrm{d}([\![a]\!]_{\varphi(t)} + [\![b]\!]_{\varphi(t)})}{\mathrm{d}t}(\zeta) && [\![\cdot]\!]_\nu \text{ is a linear operator} \\
&= \frac{\mathrm{d}[\![a]\!]_{\varphi(t)}}{\mathrm{d}t}(\zeta) + \frac{\mathrm{d}[\![b]\!]_{\varphi(t)}}{\mathrm{d}t}(\zeta) && \frac{\mathrm{d}}{\mathrm{d}t} \text{ is a linear operator} \\
&= [\![\nabla_{\mathcal{D}} a]\!]_{\varphi(\zeta)} + [\![\nabla_{\mathcal{D}} b]\!]_{\varphi(\zeta)} && \text{by induction hypothesis} \\
&= [\![\nabla_{\mathcal{D}} a + \nabla_{\mathcal{D}} b]\!]_{\varphi(\zeta)} && [\![\cdot]\!]_\nu \text{ is a linear operator} \\
&= [\![\nabla_{\mathcal{D}} (a+b)]\!]_{\varphi(\zeta)} && \nabla \text{ is linear, because } \tfrac{\partial}{\partial x_i} \text{ is linear}
\end{aligned}$$

- The case if c is of the form $a \cdot b$ is accordingly, using Leibniz's product rule for $\frac{\partial}{\partial x_i}$.

$\square$

Proof of Proposition 1 We have to show that $\nu \models F \to [\mathcal{D} \wedge H]F$ for all states ν. Let ν satisfy $\nu \models F$ as, otherwise, there is nothing to show. We can assume F to be in disjunctive normal form and consider any disjunct G of F that is true at ν. In order to show that F remains true during the continuous evolution, it is sufficient to show that each conjunct of G is. We can assume these conjuncts to be of the form $c \geq 0$ (or $c > 0$ where the proof is accordingly). Finally, using vectorial notation, we write $x' = \theta$ for the differential equation system. Now let $\varphi : [0,r] \to \mathrm{States}$ be any flow of $x' = \theta \wedge H$ beginning in $\varphi(0) = \nu$ according to Definition 2. If the duration of φ is $r = 0$, we have $\varphi(0) \models c \geq 0$ immediately, because $\nu \models c \geq 0$. For duration $r > 0$, we show that $c \geq 0$ holds all along the flow φ, i.e., $\varphi(\zeta) \models c \geq 0$ for all $\zeta \in [0, r]$.

Suppose there was a $\zeta \in [0, r]$ with $\varphi(\zeta) \models c < 0$, which will lead to a contradiction. The function $h : [0, r] \to \mathbb{R}$ defined as $h(t) = [\![c]\!]_{\varphi(t)}$ satisfies the relation $h(0) \geq 0 > h(\zeta)$, because $h(0) = [\![c]\!]_{\varphi(0)} = [\![c]\!]_\nu$ and $\nu \models c \geq 0$ by assumption (induction start of Definition 6). By Lemma 1, h is continuous on $[0, r]$ and differentiable at every $\xi \in (0, r)$. By mean value theorem, there is a $\xi \in (0, \zeta)$ such that $\frac{\mathrm{d}h(t)}{\mathrm{d}t}(\xi) \cdot (\zeta - 0) = h(\zeta) - h(0) < 0$. In particular, since $\zeta \geq 0$, we can conclude that $\frac{\mathrm{d}h(t)}{\mathrm{d}t}(\xi) < 0$. Now Lemma 1 implies that $\frac{\mathrm{d}h(t)}{\mathrm{d}t}(\xi) = [\![\nabla_{\mathcal{D}} c]\!]_{\varphi(\xi)} < 0$. This, however, is a contradiction, because the induction step of Definition 6 implies that the formula $H \to \nabla_{\mathcal{D}} c \geq 0$ is true in all states along φ, including $\varphi(\xi) \models H \to \nabla_{\mathcal{D}} c \geq 0$. In particular, as φ is a flow for $\mathcal{D} \wedge H$, we know that $\varphi(\xi) \models H$ holds, and we have $\varphi(\xi) \models \nabla_{\mathcal{D}} c \geq 0$, which contradicts $[\![\nabla_{\mathcal{D}} c]\!]_{\varphi(\xi)} < 0$. □

In Sects. 3.4–3.6, we present algorithms for finding differential invariants for differential equations, and for finding global invariants for repetitions.

3.3 Example: flight dynamics in air traffic collision avoidance

Aircraft collision avoidance maneuvers resolve conflicting flight paths, e.g., by roundabout maneuvers [35], see Fig. 4a–b. Their nontrivial dynamics makes safe separation of aircraft difficult to analyze, in particular as good timing and coordination of movement in space are crucial [8, 11, 18, 21, 26, 35]. Correct functioning of these maneuvers under all circumstances is difficult to guarantee without formal verification, especially in light of the counterexample (Fig. 4c) that our model checker discovered for the classical roundabout maneuver. Thus, verification is important for aircraft maneuvers, and—at the same time—aircraft dynamics is a challenge for hybrid systems verification.

For simplicity, we consider planar movement of aircraft (if the aircraft are collision free in a planar projection, they are collision free in space). The parameters of two aircraft at (planar) position $x = (x_1, x_2) \in \mathbb{R}^2$ and $y = (y_1, y_2)$ with angular orientation ϑ and ς are illustrated in Fig. 4d (with $\vartheta = 0$). Following [35], aircraft dynamics is determined by their linear speeds $v, u \in \mathbb{R}$ and angular speeds $\omega, \varrho \in \mathbb{R}$, respectively:

$$\begin{aligned} x_1' &= v \cos \vartheta, & x_2' &= v \sin \vartheta, & \vartheta' &= \omega, \\ y_1' &= u \cos \varsigma, & y_2' &= u \sin \varsigma, & \varsigma' &= \varrho. \end{aligned} \tag{2}$$

That is, position x moves with speed v into the direction with angular orientation ϑ, which rotates with angular velocity ω (likewise for y, u, ς, ϱ). In safe flight configurations, aircraft are separated by at least distance p:

$$(x_1 - y_1)^2 + (x_2 - y_2)^2 \geq p^2. \tag{3}$$

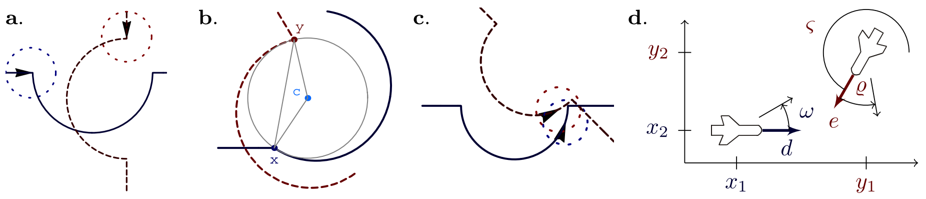


Fig. 4 Roundabout maneuvers for air traffic collision avoidance

To handle the transcendental functions in equation (2), we axiomatize sin and cos by differential equations and reparametrize the system using linear velocity vectors

$$d = (d_1, d_2) := (v \cos\vartheta, v \sin\vartheta) \in \mathbb{R}^2 \text{ and } e = (e_1, e_2) := (u \cos\varsigma, u \sin\varsigma) \in \mathbb{R}^2$$

which describe both the linear speed $\|d\| := \sqrt{d_1^2 + d_2^2} = v$ and the orientation of the aircraft in space, see vectors d and e in Fig. 4d:

$$\begin{bmatrix} x_1' = d_1 & x_2' = d_2 & d_1' = -\omega d_2 & d_2' = \omega d_1 & t' = 1 \\ y_1' = e_1 & y_2' = e_2 & e_1' = -\varrho e_2 & e_2' = \varrho e_1 & s' = 1 \end{bmatrix}. \qquad (\mathcal{F})$$

Using symbolic derivations, it is easy to see that equations ($\mathcal{F}$) and (2) are equivalent up to reparameterization. For illustration purposes, we add clock variables t, s that are usually needed for synchronizing collision avoidance maneuvers. By a simple computation, we can show that $d_1^2 + d_2^2 \ge a^2$ is a differential invariant of the differential equations ($\mathcal{F}$), thereby showing that the linear speed $\|d\|$ of aircraft does not drop below some stalling speed a during maneuvers (assuming $\|d\| \ge a$ holds initially):

$$\begin{aligned} \nabla_{\mathcal{F}} \left(d_1^2 + d_2^2 \ge a^2\right) &\equiv \nabla_{(d_1' = -\omega d_2 \wedge d_2' = \omega d_1)} \left(d_1^2 + d_2^2 \ge a^2\right) \\ &\equiv \frac{\partial(d_1^2 + d_2^2)}{\partial d_1}(-\omega d_2) + \frac{\partial(d_1^2 + d_2^2)}{\partial d_2}\omega d_1 \ge \frac{\partial a^2}{\partial d_1}(-\omega d_2) + \frac{\partial a^2}{\partial d_2}\omega d_1 \\ &\equiv 2d_1(-\omega d_2) + 2d_2 \omega d_1 \ge 0. \end{aligned}$$

As a stronger statement, the constant linear speed equation $d_1^2 + d_2^2 = a^2$ can be proven to be a differential invariant. Similarly, the conjunction $d_1^2 + d_2^2 = a^2 \wedge e_1^2 + e_2^2 \ge a^2$ can be shown to be a differential invariant of $\mathcal{F}$. The theory of differential invariants shows that this is a general phenomenon: Conjunctions of differential invariants are differential invariants but not conversely so [24]. There are examples where only the propositional combination itself is a differential invariant but none of its parts is [24]. Thus, allowing conjunctions in differential invariants is crucial and the verification power of differential invariants is higher than that of other approaches [31–34], which do not support propositional operators.

3.4 Local fixedpoint computation for differential invariants

Like for verification with invariants, the central practical question for verification with differential invariants is how to find them. Figure 5 depicts our fixedpoint algorithm for constructing differential invariants for each continuous evolution $\mathcal{D} \wedge H$ (where $\mathcal{D}$ is a differential equation system and H a first-order formula). The algorithm in Fig. 5 (called *Differential Saturation*) successively refines the domain H by differential invariants until saturation, i.e., until H accumulates enough information to become a sufficiently strong invariant that implies postcondition ϕ (line 2). If domain H already entails ϕ, then $\psi \to [\mathcal{D} \wedge H]\phi$ is proven trivially (line 2). Otherwise, the algorithm considers candidates F for augmenting H (line 3). If F is a differential invariant (line 4), then H can soundly be refined to $H \wedge F$ (line 5) without affecting the states reachable by $\mathcal{D} \wedge H$ (Proposition 2 below). Then, the fixedpoint loop repeats (line 6). At each iteration of this fixedpoint loop, the previous invariant H can be used to prove the next level of refinement $H \wedge F$ (line 4). Hence, each differential invariant provides more information to simplify subsequent iterations. The refinement of the dynamics at line 5 is sound by the following proposition, using that the

```
function prove($\psi \to [\mathcal{D} \land H]\phi$):
  if prove($\forall_{cl}(H \to \phi)$) then return true        /* property proven */
  for each $F$ ∈Candidates($\psi \to [\mathcal{D} \land H]\phi$, $H$) do
    if prove($\psi \land H \to F$) and prove($\forall_{cl}(H \to \nabla_{\mathcal{D}} F)$) then
      $H$ := $H \land F$              /* refine by differential invariant */
      goto 2;              /* repeat fixedpoint loop */
  end for
  return "not provable using candidates"
```

Fig. 5 Fixedpoint algorithm for differential invariants (*Differential Saturation*)

conditions in line 4 imply that F is a differential invariant and, thus, a continuous invariant by Proposition 1.

Proposition 2 (Differential saturation) *Assume F is a continuous invariant of $\psi \to [\mathcal{D} \land H]\phi$, then $\psi \to [\mathcal{D} \land H]\phi$ and $\psi \to [\mathcal{D} \land H \land F]\phi$ are equivalent.*

Proof The proof is a stronger version of a result in previous work [24]. Let F be a continuous invariant, which implies that $\psi \to [\mathcal{D} \land H]F$ is valid. Let state ν be a state satisfying ψ (otherwise there is nothing to show). Then, $\nu \models [\mathcal{D} \land H]F$. Since this means that F is true all along all flows φ of $\mathcal{D} \land H$ that start in state ν (Definition 2), we know that $\mathcal{D} \land H$ and $\mathcal{D} \land H \land F$ have the same dynamics and the same reachable states from state ν. That is, $(\nu, \omega) \in \rho(\mathcal{D} \land H)$ holds if and only if $(\nu, \omega) \in \rho(\mathcal{D} \land H \land F)$ (Definition 2). Thus, we can conclude that $\psi \to [\mathcal{D} \land H]\phi$ and $\psi \to [\mathcal{D} \land H \land F]\phi$ are equivalent, because their semantics (Definition 3) uses the same transition relation. □

This progressive differential saturation turns out to be crucial in practice. For instance, the aircraft separation property (3) cannot be proven until ($\mathcal{F}$) has been refined by invariants for d and e, because these variables determine x' and y' in ($\mathcal{F}$). This makes sense intuitively: Unless we have discovered some invariant about the directions d and e where the aircraft are flying to, we cannot conclude good invariants about their positions x and y, because the evolution of the positions over time depends on the directions.

In Fig. 5, function *Candidates* determines a set of candidates for induction (line 3) depending on transitive differential dependencies, as will be explained in Sect. 3.5. When these are insufficient for proving $\psi \to [\mathcal{D} \land H]\phi$, the algorithm fails (line 8, with improvements in subsequent sections). Finally, $\forall_{cl}(\phi)$ denotes the *universal closure* of ϕ. It is required in lines 2 and 4, because the respective formulas need to hold in *all* states reachable along $\mathcal{D}$ (and that satisfy H). Clearly, this set of states is overapproximated conservatively by the universal closure with respect to all variables for which there are differential equations in $\mathcal{D}$.

3.5 Dependency-directed induction candidates

In this section, we construct likely candidates for differential induction (function *Candidates*). Later, we use the same procedure for finding global loop invariants. We construct two kinds of candidates in an order induced by differential dependencies. By following the effect of hybrid systems symbolically along their decompositions, our verification algorithm enriches preconditions ψ of goals $\psi \to [\alpha_i]\phi$ successively with more precise information about the symbolic prestate as obtained by the symbolic decompositions and proof steps in Figs. 2 and 5. To exploit this, we first look for invariant symbolic state information that

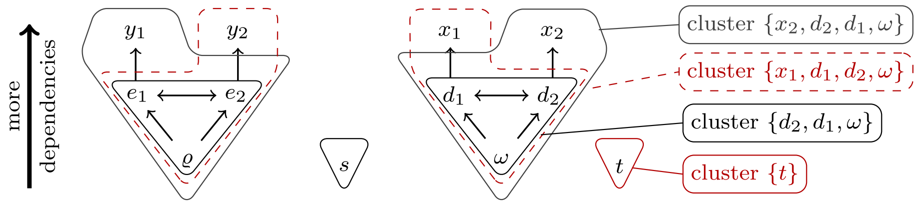

more dependencies
y_1
y_2
e_1
e_2
ϱ
s
x_1
x_2
d_1
d_2
ω
t
cluster $\{x_2, d_2, d_1, \omega\}$
cluster $\{x_1, d_1, d_2, \omega\}$
cluster $\{d_2, d_1, \omega\}$
cluster $\{t\}$

Fig. 6 Differential dependencies (*arrows*) and (triangular) variable clusters of ($\mathcal{F}$)

accumulated in ψ and ϕ during the iterative symbolic decomposition by selecting subformulas of ψ and ϕ that are not yet contained in H. In practice, this gives particularly good candidates for highly parametric hybrid systems.

Secondly, we generate parametric invariants. Let $V = \{x_1, \dots, x_n\}$ be a set of relevant variables. We choose fresh names $a^{(l)}_{i_1,\dots,i_n}$ for *formal parameters* of the invariant candidates and build polynomials $p_1, \dots, p_k$ of degree d with variables V using formal parameters as symbolic coefficients: $p_l := \sum_{i_1+\dots+i_n \le d} a^{(l)}_{i_1,\dots,i_n} x_1^{i_1} \dots x_n^{i_n}$ for $1 \le l \le k$. We define the set of *parametric candidates* (operator $\vee$ is similarly) as:

$$\mathit{ParaForm}(k, d, V) := \left\{ \bigwedge_{l=1}^{i} p_l \ge 0 \wedge \bigwedge_{l=i+1}^{k} p_l = 0 \ :\ 0 \le i \le k \right\}.$$

For instance, the parametric candidate $a_{0,0} + a_{1,0}d_1 + a_{0,1}x_2 = 0$ yields a differential invariant of ($\mathcal{F}$) for the choice $a_{0,0} = 0$, $a_{1,0} = 1$, $a_{0,1} = \omega$. By simple combinatorics, *ParaForm*(k, d, V) contains $k + 1$ candidates with $k\binom{n+d}{d}$ formal parameters $a^{(l)}_{i_1,\dots,i_n}$, which are existentially quantified. Existence of a common satisfying instantiation for these parameters can be expressed by prefixing the resulting d$\mathcal{L}$ formula that uses this parametric candidate with $\exists a^{(l)}_{i_1,\dots,i_n}$ for each of the formal parameters $a^{(l)}_{i_1,\dots,i_n}$. For this search for common solutions to be feasible, the number of parameters is crucial, which we minimize by respecting (differential) dependencies. We will illustrate the placement of existential quantifiers in Example 1 of Sect. 3.6 after we have presented the global fixedpoint algorithm.

To accelerate the differential saturation process in Sect. 3.4, it is crucial to explore candidates in a promising order from simple to complex, because the algorithm in Fig. 5 uses successful differential invariants to refine the dynamics, thereby simplifying subsequent proofs. Again, property (3) is only provable after the dynamics has been refined with invariants for d and e, because x' and y' depend on the direction d and e. In fact, safety of roundabouts crucially depends on compatible directions of the aircraft, because there are counterexamples otherwise, see Fig. 4c. We construct candidates in a natural order based on variable occurrence that is consistent with the *differential dependencies* of the differential equations. For a differential equation $\mathcal{D}$, variable x depends on variable y according to the differential equation system $\mathcal{D}$ if y occurs on the right-hand side of the equation for x' (or transitively so). The resulting set *depend*$(\mathcal{D})$ of dependencies is the transitive closure of $\{(x, y) : (x' = \theta) \in \mathcal{D}$ and y occurs in $\theta\}$. From the differential equation system ($\mathcal{F}$), we determine the differential dependencies indicated as arrows (pointing to the dependent variables x) in Fig. 6.

From these dependencies we determine an order on candidates. The idea is that, as the value of x_1 depends on that of d_1, it makes sense to look for invariant expressions of d_1 first, because refinements with these help differential saturation in proving invariant expressions involving also x_1. Thus, we order variables by differential dependencies, which resembles the back substitution order in Gaussian elimination (if, in triangular form, x_1 depends on d_1 then equations for d_1 must be solved first). Now we call a set V of variables a *cluster* of the differential equation $\mathcal{D}$ iff V is closed with respect to $\mathit{depend}(\mathcal{D})$, i.e., variables of V only depend on variables in V:

$$x \in V \text{ and } (x, y) \in \mathit{depend}(\mathcal{D}) \text{ imply } y \in V.$$

The resulting variable clusters for system ($\mathcal{F}$) are marked as triangular shapes in Fig. 6. Finally, we choose candidates from ψ and $\mathit{ParaForm}(k, d, V)$ starting with candidates whose variables lie in small clusters V and cover larger fractions of that cluster. Thus, the differential invariant $d_1^2 + d_2^2 \geq a^2$ of Sect. 3.3 within cluster $\{d_2, d_1, \omega\}$ can be discovered before invariants like $d_1 = -\omega x_2$ that involve x_2, because x_2 depends on d_2.

3.6 Global fixedpoint computation for loop invariants

With the uniform setup of d$\mathcal{L}$, we can adapt the algorithm in Fig. 5 easily to obtain a fixedpoint algorithm for loops (in formulas of the form $\psi \to [\alpha^*]\phi$) in place of continuous evolutions ($\psi \to [\mathcal{D} \wedge H]\phi$): In line 4 of Fig. 5, we replace the induction step from Definition 6 by the induction step for loops (Definition 4). As an optimization, we can transfer the reuse of partial differential invariants according to Proposition 2 to discrete invariants for loops. Invariants H of previous iterations can be exploited as refinements of the hybrid system dynamics, similar to previous differential invariants that can be used in future iterations by refining the dynamics using differential saturation:

Proposition 3 (Loop saturation) *If H is a discrete invariant of $\psi \to [\alpha^*]\phi$, then $H \wedge F$ is a discrete invariant iff $\psi \to F$ and $H \wedge F \to [\alpha](H \to F)$ are valid.*

Proof Let H be a discrete invariant of $\psi \to [\alpha^*]\phi$. Let, further, $H \wedge F$ be a discrete invariant of $\psi \to [\alpha^*]\phi$. Then $\psi \to H \wedge F$ and $H \wedge F \to [\alpha](H \wedge F)$ are valid by Definition 4. Hence, trivially, $H \wedge F \to [\alpha](H \to F)$ is valid, because all states that satisfy $H \wedge F$ also satisfy the weaker property $H \to F$. Finally, the validity of $\psi \to H \wedge F$ clearly entails $\psi \to F$.

Conversely, let, H be a discrete invariant. Let, further, $H \wedge F \to [\alpha](H \to F)$ and $\psi \to F$ be valid. For $H \wedge F$ to be a discrete invariant, we have to show that F satisfies the induction step of Definition 4 (the induction start $\psi \to H \wedge F$ is an immediate combination of the validity of $\psi \to H$ and $\psi \to F$). Since H is a discrete invariant, $H \to [\alpha]H$ is valid, which entails $H \wedge F \to [\alpha]H$ as a special case. Since $H \wedge F \to [\alpha](H \to F)$ is valid and $H \wedge F \to [\alpha]H$ is valid, we conclude that $H \wedge F \to [\alpha](H \wedge F)$ is valid for the following reason. Let state ν be a state satisfying the initial constraints $H \wedge F$. Then the above validities yield $\nu \models [\alpha]H$ and $\nu \models [\alpha](H \to F)$. Hence, all states ω reachable from ν by α satisfy $\omega \models H$ and $\omega \models H \to F$. Thus, they satisfy $\omega \models H \wedge F$, essentially by modus ponens. Consequently, we have shown that $H \wedge F \to [\alpha](H \wedge F)$ is valid, and, hence, $H \wedge F$ is a discrete invariant of $\psi \to [\alpha^*]\phi$. □

The induction step from Proposition 3 can generally be proven faster, because it is a weaker property than that of Definition 4.

To adapt our approach from Sect. 3.5 to loops, we use discrete data-flow and control-flow dependencies of α. There is a direct *data-flow dependency* with the value of x depending on y, if $x := \theta$ or $x' = \theta$ occurs in α with a term θ that contains y. Accordingly, there

```
1 function prove($\psi \to [\alpha^*]\phi$):
2   $H$ := true            /* currently known invariant of $\psi \to [\alpha^*]\phi$ */
3   if prove($\forall_{cl}(H \to \phi)$) then return true      /* property proven */
4   for each $F \in$Candidates($\psi \to [\mathcal{D} \wedge H]\phi$, $H$) do
5     if prove($\psi \wedge H \to F$) and prove($\forall_{cl}(H \wedge F \to [\alpha](H \to F))$) then
6       $H$ := $H \wedge F$              /* refine by discrete invariant */
7       goto 3;              /* repeat fixedpoint loop */
8   end for
9   return "not provable using candidates"
```

Fig. 7 Fixedpoint algorithm for discrete loop invariants (loop saturation)

is a direct *control-flow dependency*, if, for any term θ, $x := \theta$ or $x' = \theta$ occurs in α after a test $?H$ containing y. The respective data-flow and control-flow dependencies are the transitive closures of these relations.

The algorithm in Fig. 7 verifies loops. It adapts that in Fig. 5, using Proposition 3 as an induction step for loops. The algorithm in Fig. 7 performs a fixedpoint computation for loops and recursively combines the local differential invariants obtained by differential saturation to form a global invariant. It recursively uses *prove* for verifying its subtasks, which handle the discrete switching behavior according to Fig. 2 and infer local differential invariants according to differential saturation by the fixedpoint algorithm in Fig. 5.

Example 1 (Existential parameter quantification) Note that the ability of formulas in differential dynamic logic to have quantifiers in front of reachability modalities is crucial here. To illustrate, consider the simple water tank system from Fig. 1. During the verification run for property (1), we need to show a property of the following form with a loop *wctrl**:

$$x \leq 3 \wedge q = on \to [(wctrl)]x < 10.$$

For a parametric candidate F of the form $a_1 x + a_0 \geq 0$, line 5 of the algorithm in Fig. 7 produces subtasks for discrete induction (Definition 4), which will be handled recursively by the *prove* function:

$$\text{prove}(x \leq 3 \wedge q = on \wedge H \to a_1 x + a_0 \geq 0)$$
$$\textbf{and}\ \text{prove}(\forall_{cl}(a_1 x + a_0 \geq 0 \wedge H \to [wctrl](H \to a_1 x + a_0 \geq 0))).$$

In the first iteration (where H is still *true*), the combination of these subtasks by conjunction corresponds to proving the following overall d$\mathcal{L}$ formula:

$$\exists a_0 \exists a_1 \big((x \leq 3 \wedge q = on \to a_1 x + a_0 \geq 0) \wedge \forall x (a_1 x + a_0 \geq 0 \to [wctrl]\, a_1 x + a_0 \geq 0)\big). \tag{4}$$

The universal quantifier $\forall x$ in (4) results from the universal closure $\forall_{cl}$ with respect to all variables changed in *wctrl*, i.e., x. The existential quantifiers for a_0 and a_1 are for formal parameters in the parametric candidate (see Sect. 3.5). Observe that the outer placement of existential quantifiers is required for the resulting instance of $a_1 x + a_0 \geq 0$ to be a *common solution* implied by the precondition (left conjunct) and inductive for *wctrl* (right conjunct). In particular, the right conjunct requires finding parameter choices for which $a_1 x + a_0 \geq 0$ holds true after executing one iteration *wctrl* of the loop if it was true before. Note that this inherently requires quantifiers around reachability properties, which are readily expressible in the first-order verification logic d$\mathcal{L}$. The left conjunct expresses that the parameter choices need to make $a_1x + a_0 \geq 0$ true in the beginning. The conjunction and outer placement of $\exists a_0 \exists a_1$ ensures that the required parameter choices fit together. Equation (4) also illustrates the need for the universal closure, because we need the same choice for the formal parameters a_0, a_1 to be able to conclude the induction step (case 2 of Definition 4) for all states x. Different incompatible choices for a_0, a_1 at each state would not yield an inductive argument. To prove the example in Fig. 1, our algorithm will discover the parameter combination $a_1 = -1$ and $a_0 = 9.5$, for instance.

For verification with parametric candidates to be feasible, we exploit that we can keep existential quantifiers as local as possible in d$\mathcal{L}$, which we ensure by the symbolic decompositions in our logic. For instance, quantifiers for the formal parameters of a differential invariant will remain local to the formulas in the algorithm in Fig. 5 rather than using global quantifiers for the whole verification problem at once. Minimizing the number of parameters according to Sect. 3.5 further improves the computational tractability.

3.7 Interplay of local and global fixedpoint loops

The local and global fixedpoint algorithms jointly verify correctness properties of HP. Their interplay needs to be coordinated with fairness. If the local fixedpoint algorithm in Fig. 5 does not converge, stronger invariants may need to be found by the global fixedpoint algorithm which iteratively result in stronger preconditions ψ_i for the local fixedpoint algorithm at the respective subtasks, see Fig. 8.

Thus, for fairness reasons, the local fixedpoint algorithm should stop when it cannot prove its postcondition, either because of an actual counterexample for its local subtask or because it runs out of candidates for differential invariants. As in the work of Prajna [32], the degrees of parametric invariants, therefore, need to be bounded and increased iteratively. As in [32], there is no natural measure for how these degrees should be increased. Instead, here, we exploit the fact that the candidates computed by the function *Candidates* are independent and we explore them in parallel with fair time interleaving.

During fixedpoint computations, wrong choices of candidates are time consuming. Thus, in practice, it is important to discover futile attempts quickly. For this, non-exhaustive sampling with numerical simulations can be used to look for counterexamples. Note that this numerical counterexample search can be performed separately for each local subtask in the decomposition. For instance, a counterexample for a candidate F for node $\psi_4 \to [\alpha_4]\phi_4$ in

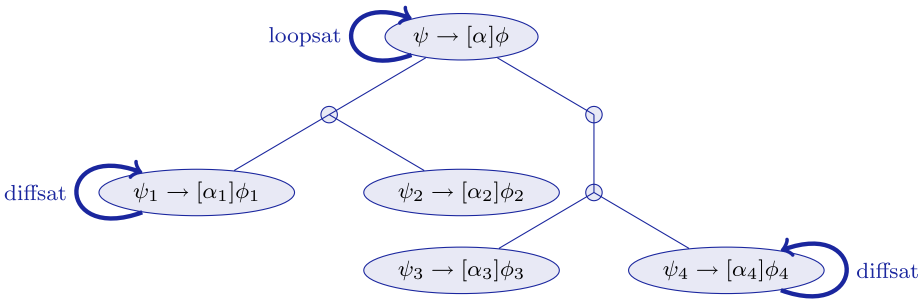

loopsat
$\psi \to [\alpha]\phi$
diffsat
$\psi_1 \to [\alpha_1]\phi_1$
$\psi_2 \to [\alpha_2]\phi_2$
$\psi_3 \to [\alpha_3]\phi_3$
$\psi_4 \to [\alpha_4]\phi_4$
diffsat

Fig. 8 Recursive symbolic decompositions (top-down) and interplay of local (diffsat) and global (loopsat) fixedpoints verification loops during symbolic decomposition

Fig. 8 will only abort the attempt to prove this candidate F. A counterexample found by local numerical simulation for the reachability property $\psi_4 \to [\alpha_4]\phi_4$ itself, instead, will terminate all proof search for that subtask by propagating the need for a stronger precondition ψ_4 up the decomposition tree in Fig. 8. To prevent rejecting good candidates due to numerical errors, we discard fragile counterexamples with tolerances below the numerical accuracy. Unlike in other approaches [3, 19, 23, 26, 32], numerical errors are *not* critical for soundness here, because safety is exclusively established by sound symbolic verification.

3.8 Soundness

We show that our verification algorithm produces correct verification results. That is, whenever it returns "`true`" for a property, that property actually holds true for the respective system.

Theorem 1 (Soundness) *The verification algorithm in Sect. 3 is sound, i.e., whenever* `prove(`$\psi \to [\alpha]\phi$`)` *returns "*`true`*", the* d$\mathcal{L}$ *formula* $\psi \to [\alpha]\phi$ *is true in all states, i.e., all states reachable by* α *from states satisfying* ψ *satisfy* ϕ.

Proof Soundness is a consequence of the fact that every statement of the algorithm that returns "true" is justified by a valid formula in the logic [24]. We prove by induction on the structure of the algorithm that, for any d$\mathcal{L}$ formula ϕ, ϕ is true in every state ν where (the formula returned by) $prove(\phi)$ is true. That is: $\nu \models prove(\phi)$ implies $\nu \models \phi$. In particular, if the algorithm returns "true", the input formula is true in all states.

- In the base case (line 11 of Fig. 2), *prove* returns the result of quantifier elimination, which is a sound decision procedure [6]. The result of quantifier elimination is equivalent to its input. Hence one is true in a state ν if and only if the other is.
- If α is of the form $x := \theta$, the algorithm in line 1 of Fig. 2 is responsible. Consider any state ν where the formula returned by $prove(\psi \to [x := \theta]\phi)$ is true. That is $prove(\psi \wedge \hat{x} = \theta \to \phi_x^{\hat{x}})$ is true at state ν. Hence, by induction hypothesis, $\psi \wedge \hat{x} = \theta \to \phi_x^{\hat{x}}$ is true at state ν. Now, because $\hat{x}$ was a fresh variable, the substitution lemma can be used to show that $\psi \to \phi_x^{\theta}$ and $\psi \to [x := \theta]\phi$ are true at state ν.
- If α is of the form $x := random$, the algorithm in line 9 of Fig. 2 is responsible. The proof is a direct consequence of the fact that ϕ being true after *all* random assignments to x is equivalent to ϕ being true for all real values of x. Hence, $\psi \to [x := random]\phi$ is true in a state ν if and only if $\psi \to \forall x \phi$ is.
- For formulas of the form $\psi \to Qx\, \phi$ where Q is a quantifier, line 13 of Fig. 2 is responsible. If the formula returned by $prove(\psi \to Qx\, \phi)$ is true in state ν, then $QuantifierElimination(\psi \to Qx\, prove(\phi))$ is true in state ν. As quantifier elimination yields an equivalent formula, this implies that $\psi \to Qx\, prove(\phi)$ itself is true at state ν. Assume that ψ holds at state ν as there is nothing to show otherwise. Then $Qx\, prove(\phi)$ is true at state ν. Thus for all (or some if Q is $\exists$) states ω that agree with state ν except for the value of x, we know that the formula returned by $prove(\phi)$ is true at ω. The formula ϕ is structurally simpler, hence, by induction hypothesis, ϕ is true at ω. Consequently, both $Qx\, \phi$ and $\psi \to Qx\, \phi$ are true at state ν, as ω was arbitrary.
- The other cases of Fig. 2 are accordingly.
- If α is of the form $\mathcal{D} \wedge H$ for a differential equation system $\mathcal{D}$, the algorithm in Fig. 5 is responsible. If it returns "true" in line 2 in the first place, then the call to *prove* in line 2 must have resulted in "true", hence, by induction hypothesis, H entails ϕ. Thus, postcondition ϕ is true in a subregion of the evolution domain H. Therefore $\psi \to [\mathcal{D} \wedge H]\phi$

is true in all states, trivially, because all evolutions along $\mathcal{D} \wedge H$ always satisfy H and, hence, ϕ. If, however, H was changed in line 5 during the fixedpoint computation, then the calls to *prove* for the properties in line 4 must have returned "true". Thus, by induction hypothesis, the d$\mathcal{L}$ formulas $\psi \wedge H \rightarrow F$ and $\forall_{cl}(H \rightarrow \nabla_{\mathcal{D}} F)$ are valid. Hence, formula F is a differential invariant of $\psi \rightarrow [\mathcal{D} \wedge H]\phi$ by Definition 6. Consequently, by Proposition 1, F also is a continuous invariant (Definition 5). Thus, by Proposition 2, the d$\mathcal{L}$ formulas $\psi \rightarrow [\mathcal{D} \wedge H]\phi$ and $\psi \rightarrow [\mathcal{D} \wedge H \wedge F]\phi$ are equivalent, and we can (soundly) verify the former by proving the latter. Consequently, the modification of the evolution domain H to $H \wedge F$ in line 5 is sound, because the algorithm will continue proving a refined but equivalent formula for a refined but equivalent system.

- If α is a loop of the form β^*, the proof is similar to the case for differential equations, except that it uses Proposition 3 instead of Proposition 1. □

Since reachability of hybrid systems is undecidable, our algorithm must be incomplete. It can fail to converge when the required invariants are not expressible in first-order logic (yet, the required invariants are always expressible in d$\mathcal{L}$ [25]). Consequently, a fixedpoint in d$\mathcal{L}$ always exists but our algorithm may fail to find appropriate (differential) invariants in first-order logic.

4 Experimental results

As an example with nontrivial dynamics, we analyze aircraft roundabout maneuvers [35]. Curved flight as in roundabouts is challenging for verification, because of its transcendental solutions. The maneuver in Fig. 4a from [35] and the maneuver in Fig. 4b from [24, 26] are not flyable, because they still involve a few instant turns. A flyable roundabout maneuver without instant turns is depicted in Fig. 9. We verify safety properties for most (but not yet all) phases of Fig. 9 fully automatically and provide verification results in Table 2. Finally, note that the required invariants for the roundabout maneuver cannot even be found from Differential Gröbner Bases [20].

Verification results for roundabout aircraft maneuvers [8, 24, 26, 28, 35], the European Train Control System (ETCS) [30], and a simplified version of an automatic speed controller in car platooning [7] are in Table 2. The case studies behind Table 2 are of independent interest. An overview can be found in [7, 28, 30].[1] Full details and a more general context about extended case studies will be presented in separate work, since the full presentation of each of the case studies is beyond the scope of this article.

Results are from a 2.6 GHz AMD Opteron™ 1218 with 4 GB memory. Memory consumption of quantifier elimination is shown in Table 2, excluding the front-end. The results

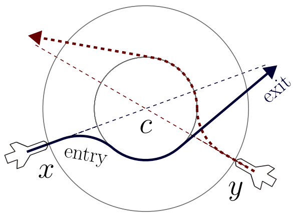


Fig. 9 Flyable aircraft roundabout

[1] Verification tool KeYmaera [29] available at http://symbolaris.com/info/KeYmaera.html.

Table 2 Experimental results

Case study	Time (s)	Memory (MB)	Proof steps	Dimension
tangential roundabout (2 aircraft)	14	8	117	13
tangential roundabout (3 aircraft)	387	42	182	18
tangential roundabout (4 aircraft)	730	39	234	23
tangential roundabout (5 aircraft)	1964	88	317	28
bounded speed roundabout entry	20	34	28	12
flyable roundabout entry (simplified)	6	10	98	8
ETCS-kernel safety	41	28	53	9
ETCS safety	183	87	169	15
ETCS train controllability	1	6	17	5
ETCS RBC controllability	1	7	45	16
car speed control (simplified)	3632	480	98	9

are only slightly worse on a 1.7 GHz Pentium M laptop with 1 GB. The dimension of the overall continuous state space is indicated. Notice that we handle *all* these variables symbolically leading to state spaces up to $\mathbb{R}^{28}$. The experimental results are encouraging, in particular as the memory consumption is fairly moderate. High memory consumption would limit scalability much more than time consumption. For instance, numerical techniques on a rather coarse numerical mesh with only 10 samples per variable would already need to consider reachability analysis from 10^{28} initial states for this case. The results in Table 2 indicate that scalability of our approach is substantially less limited by the number of variables than in other approaches. Instead, the complexity of the constraints and dependencies among the variables have more impact. These dependencies, for instance, are more complex in the car controller than in 5 aircraft roundabouts. In parallel systems, there is usually a smaller fraction of dependencies among continuous variables. Multiple aircraft, e.g., are not coupled physically (see Fig. 6) but interact only by discrete dynamics originating from communication.

5 Related work

Tools like HyTech [17], PHAVer [13], CheckMate [4], or other approaches [12, 16, 23] cannot handle our applications with nonlinear switching, nonlinear discrete and continuous dynamics, and high-dimensional state spaces.

Other authors [31–34] already argued that invariant techniques scale to more general dynamics than explicit reach-set computations or techniques that require solutions for differential equations [12, 23]. However, their techniques are focused on purely equational systems and cannot handle hybrid systems with inequalities in initial sets or switching surfaces [33, 34]. Due to tolerances, inequalities occur in most real applications like aircraft maneuvers. *Barrier certificates* [31, 32] only work for single inequalities, but invariants of roundabout maneuvers require mixed equations and inequalities [24]. Prajna et al. [32] search for barrier certificates of a fixed degree by global optimization over the set of all proof attempts for the whole system at once, which is infeasible: Even with degree bound 2, it would already require solving a 5848-dimensional optimization problem for ETCS [30] and a 10005-dimensional problem for roundabouts with 5 aircraft.

Finally, important distinctions of our work compared to others [31–34] are: (i) we allow arbitrary formulas as differential invariants, which provably improves verification power; (ii) we increase the verification power further by nesting differential invariants using differential saturation to refine the system dynamics; and (iii) our compositional verification logic allows local generation of differential invariants and natural local existential quantification of formal parameters for local verification subtasks.

Tomlin et al. [35] derive saddle solutions for aircraft maneuver games using Hamilton-Jacobi-Isaacs partial differential equations and propose roundabout maneuvers. Their exponential state space discretizations for PDEs, however, do not scale to larger dimensions (they consider dimension 3) and can be numerically unsound [26]. Differential invariants, instead, work for 28-dimensional systems and are sound.

Straight-line aircraft maneuvers have been analyzed by geometrical meta-level reasoning [11, 18]. We directly verify the actual hybrid flight dynamics, including curved roundabout maneuvers instead of straight-line maneuvers with non-flyable instant turns. A few approaches [8, 21] have been undertaken to Model Check if there are *orthogonal* collisions in discretizations of roundabout maneuvers. However, the counterexamples (see Fig. 4c) found by our model checker show that *non-orthogonal* collisions can happen in these maneuvers.

6 Conclusions and future work

We have presented a *sound* algorithm for verifying hybrid systems with nontrivial dynamics. It handles differential equations using differential invariants instead of requiring solutions of the differential equations, because the latter quickly yield undecidable arithmetic. We compute differential invariants as fixedpoints using a verification logic for hybrid systems. In the logic we can decompose the system for computing local invariants and we obtain sound recombinations into global invariants. Moreover, we introduce a differential saturation procedure that verifies more complicated properties by refining the system dynamics successively in a sound way in our logic. We validate our algorithm on challenging *roundabout collision avoidance maneuvers* for aircraft, on collision avoidance protocols for trains, and on automatic speed controllers for cars. We give verification results with a sound algorithm for challenging dynamics of curved flight in up to 28 dimensional continuous state spaces.

Our algorithm works particularly good for highly parametric hybrid systems, because their parameter constraints can be combined faster to find invariants than for systems with a single initial state, where simulation is more appropriate. Our decompositional approach exploits locality in system designs. In well-designed systems, subsystems do not generally depend on all other parts of the system but are built according to modularity and locality principles in engineering. In these cases, our verification logic achieves good decompositions and the required invariants for one part of the system only need very little information about other parts of the systems, so that the fixedpoint algorithm terminates faster. Our algorithm performs worse for systems that violate locality principles.

Scalability, e.g., to applications from other domains like chemical process control or biomedical devices remains an interesting topic for future research. An independent question for future research, underlying virtually all hybrid systems verification approaches, is how to scale real arithmetic handling. Semidefinite programming relaxations [22] may be an interesting direction for future research. Differential induction and the logic $\mathsf{d}\mathcal{L}$ generalize to liveness properties and to systems with disturbances. In future work, we want to generalize the synthesis of corresponding differential (in)variants. Other invariant constructions for differential equations, e.g., [33] can be added and lifted to hybrid systems using our uniform algorithm.

Acknowledgements We thank the reviewers for their careful and most helpful comments. The first author was partially supported by the German Research Council (DFG) under grant SFB/TR 14 AVACS. The second author was partially supported by the National Science Foundation (NSF) under grants No. CCR-0411152, CCF-0429120, CCF-0541245, by General Motors and the Carnegie Mellon University-General Motors Collaborative Research Laboratory under grant No. GM9100096UMA, by the Semiconductor Research Corporation (SRC) under contracts No. 2008TJ1860, by the Gigascale Systems Research Center (GSRC) under grant No. SA423679952, by the International Collaboration for Advanced Security Technology of the National Science Council, Taiwan, under grant No. 1010717, and by the Air Force (University of Vanderbilt) under contract No. 18727S3. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution or government.

References

1. Alur R, Pappas GJ (eds) (2004) Hybrid systems: computation and control, Proceedings of the 7th International Workshop, HSCC 2004, Philadelphia, PA, USA, March 25–27, 2004. LNCS, vol 2993. Springer, Berlin
2. Anai H, Weispfenning V (2001) Reach set computations using real quantifier elimination. In: Benedetto MDD, Sangiovanni-Vincentelli AL (eds) HSCC. LNCS, vol 2034. Springer, Berlin, pp 63–76. doi:10.1007/3-540-45351-2_9
3. Asarin E, Dang T, Girard A (2003) Reachability analysis of nonlinear systems using conservative approximation. In: Maler O, Pnueli A (eds) HSCC. LNCS, vol 2623. Springer, Berlin, pp 20–35. doi:10.1007/3-540-36580-X_5
4. Chutinan A, Krogh BH (2003) Computational techniques for hybrid system verification. IEEE Trans Autom Control 48(1):64–75. doi:10.1109/TAC.2002.806655
5. Clarke EM (1979) Program invariants as fixedpoints. Computing 21(4):273–294. doi:10.1007/BF02248730
6. Collins GE, Hong H (1991) Partial cylindrical algebraic decomposition for quantifier elimination. J Symb Comput 12(3):299–328. doi:10.1016/S0747-7171(08)80152-6
7. Damm W, Hungar H, Olderog ER (2006) Verification of cooperating traffic agents. Int J Control 79(5):395–421
8. Damm W, Pinto G, Ratschan S (2005) Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. In: Peled D, Tsay YK (eds) ATVA. LNCS, vol 3707. Springer, Berlin, pp 99–113. doi:10.1007/11562948_10
9. Davoren JM, Nerode A (2000) Logics for hybrid systems. IEEE 88(7):985–1010. doi:10.1109/5.871305
10. Donzé A, Maler O (2004) Systematic simulation using sensitivity analysis. In: Bemporad A et al (eds) Hybrid systems: computation and control, Proceedings of the 10th international conference, HSCC 2007, Pisa, Italy. LNCS, vol 4416. Springer, Berlin, pp 174–189. doi:10.1007/978-3-540-71493-4_16
11. Dowek G, Muñoz C, Carreño VA (2005) Provably safe coordinated strategy for distributed conflict resolution. In: Proceedings of the AIAA guidance navigation, and control conference and exhibit 2005, AIAA-2005-6047
12. Fränzle M (1999) Analysis of hybrid systems: an ounce of realism can save an infinity of states. In: Flum J, Rodríguez-Artalejo M (eds) CSL. LNCS, vol 1683. Springer, Berlin, pp 126–140
13. Frehse G (2005) PHAVer: Algorithmic verification of hybrid systems past HyTech. In: Morari M, Thiele L (eds) Hybrid systems: computation and control, Proceedings of the 8th international workshop, HSCC 2005, Zurich, Switzerland, March 9–11, 2005. LNCS, vol 3414. Springer, Berlin, pp 258–273. doi:10.1007/b106766
14. Gulwani S, Tiwari A (2008) Constraint-based approach for analysis of hybrid systems. In: Gupta A, Malik S (eds) Proceedings computer aided verification, CAV 2008, Princeton, NJ, USA. LNCS, vol 5123. Springer, Berlin, pp 190–203. doi:10.1007/978-3-540-70545-1
15. Harel D, Kozen D, Tiuryn J (2000) Dynamic logic. MIT Press, Cambridge
16. Henzinger TA (1996) The theory of hybrid automata. In: LICS. IEEE Computer Society, Los Alamitos, pp 278–292
17. Henzinger TA, Ho PH, Wong-Toi H (1997) HyTech: a model checker for hybrid systems. In: Grumberg O (ed) CAV. LNCS, vol 1254. Springer, Berlin, pp 460–463

18. Hwang I, Kim J, Tomlin C (2007) Protocol-based conflict resolution for air traffic control. Air Traffic Control Q 15(1):1–34
19. Lanotte R, Tini S (2005) Taylor approximation for hybrid systems. In: Morari M, Thiele L (eds) Hybrid systems: computation and control, Proceedings of the 8th international workshop, HSCC 2005, Zurich, Switzerland, March 9–11, 2005. LNCS, vol 3414. Springer, Berlin, pp 402–416. doi:10.1007/b106766
20. Mansfield EL (1991) Differential Gröbner bases. PhD thesis, University of Sydney
21. Massink M, Francesco ND (2001) Modelling free flight with collision avoidance. In: Andler SF, Offutt J (eds) ICECCS. IEEE Computer Society, Los Alamitos, pp 270–280. doi:10.1109/ICECCS.2001.930186
22. Parrilo PA (2003) Semidefinite programming relaxations for semialgebraic problems. Math Program 96(2):293–320. doi:10.1007/s10107-003-0387-5
23. Piazza C, Antoniotti M, Mysore V, Policriti A, Winkler F, Mishra B (2005) Algorithmic algebraic model checking, I: challenges from systems biology. In: Etessami K, Rajamani SK (eds) CAV. LNCS, vol 3576. Springer, Berlin, pp 5–19. doi:10.1007/11513988_3
24. Platzer A (2008) Differential-algebraic dynamic logic for differential-algebraic programs. J Log Comput. doi:10.1093/logcom/exn070
25. Platzer A (2008) Differential dynamic logic for hybrid systems. J Autom Reas 41(2):143–189. doi:10.1007/s10817-008-9103-8
26. Platzer A, Clarke EM (2007) The image computation problem in hybrid systems model checking. In: Bemporad A et al (eds) Hybrid systems: computation and control, Proceedings of the 10th international conference, HSCC 2007, Pisa, Italy. LNCS, vol 4416. Springer, Berlin, pp 473–486. doi:10.1007/978-3-540-71493-4_37
27. Platzer A, Clarke EM (2008) Computing differential invariants of hybrid systems as fixedpoints. In: Gupta A, Malik S (eds) Proceedings computer aided verification, CAV 2008, Princeton, NJ, USA. LNCS, vol 5123. Springer, Berlin, pp 176–189. doi:10.1007/978-3-540-70545-1_17
28. Platzer A, Clarke EM (2008) Computing differential invariants of hybrid systems as fixedpoints. Tech Rep CMU-CS-08-103, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA
29. Platzer A, Quesel JD (2008) KeYmaera: A hybrid theorem prover for hybrid systems. In: Armando A, Baumgartner P, Dowek G (eds) IJCAR. LNCS, vol 5195. Springer, Berlin, pp 171–178. doi:10.1007/978-3-540-71070-7_15
30. Platzer A, Quesel JD (2008) Logical verification and systematic parametric analysis in train control. In: Egerstedt M, Mishra B (eds) HSCC. LNCS, vol 4981. Springer, Berlin, pp 646–649. doi:10.1007/978-3-540-78929-1_55
31. Prajna S, Jadbabaie A (2004) Safety verification of hybrid systems using barrier certificates. In: Alur R, Pappas GJ (eds) Hybrid systems: computation and control, Proceedings of the 7th international workshop, HSCC 2004, Philadelphia, PA, USA, March 25–27, 2004. LNCS, vol 2993. Springer, Berlin, pp 477–492. doi:10.1007/b96398
32. Prajna S, Jadbabaie A, Pappas GJ (2007) A framework for worst-case and stochastic safety verification using barrier certificates. IEEE Trans Autom Control 52(8):1415–1429. doi:10.1109/TAC.2007.902736
33. Rodríguez-Carbonell E, Tiwari A (2005) Generating polynomial invariants for hybrid systems. In: Morari M, Thiele L (eds) Hybrid systems: computation and control, Proceedings of the 8th international workshop, HSCC 2005, Zurich, Switzerland, March 9–11, 2005. LNCS, vol 3414. Springer, Berlin, pp 590–605. doi:10.1007/b106766
34. Sankaranarayanan S, Sipma H, Manna Z (2004) Constructing invariants for hybrid systems. In: Alur R, Pappas GJ (eds) Hybrid systems: computation and control, Proceedings of the 7th international workshop, HSCC 2004, Philadelphia, PA, USA, March 25–27, 2004. LNCS, vol 2993. Springer, Berlin, pp 539–554. doi:10.1007/b96398
35. Tomlin C, Pappas GJ, Sastry S (1998) Conflict resolution for air traffic management: a study in multiagent hybrid systems. IEEE Trans Autom Control 43(4):509–521. doi:10.1109/9.664154