Computing Differential Invariants of Hybrid Systems as Fixedpoints

André Platzer[1] and Edmund M. Clarke[2]

[1] University of Oldenburg, Department of Computing Science, Germany
[2] Carnegie Mellon University, Computer Science Department, Pittsburgh, PA
{aplatzer|emc}@cs.cmu.edu

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. 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.

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 [1–4] 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 [5] can be used for verification [3, 6–9]. Unfortunately, this symbolic approach does not scale to systems with complicated differential equations whose solutions do not support quantifier elimination (e.g., when they are transcendental functions) or cannot be given in closed form.

Numerical or approximation approaches [10–12] can deal with more general dynamics. However, numerical or approximation errors need to be handled carefully as they easily cause unsoundness [11]. More specifically, we have shown previously that even single image computations of fairly restricted classes of hybrid systems are undecidable by numerical computation [11]. Thus, numerical approaches can be used for falsification but not (ultimately) for verification.

A. Gupta and S. Malik (Eds.): CAV 2008, LNCS 5123, pp. 176–189, 2008.

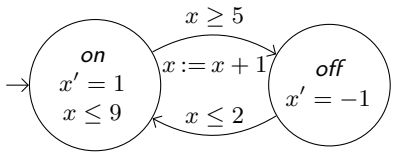
In this paper, we present an approach that combines the soundness of symbolic approaches [3, 7–9] with support for nontrivial dynamics that is classically more dominant in numerical approaches [10–12]. During continuous transitions, the system follows a solution of its differential equation. But for nontrivial dynamics, these solutions are much more complicated than the original equations. Solutions quickly become transcendental even if the differential equations are linear. To overcome this, we handle continuous transitions based on their vector fields, which are described by their differential equations. We use *differential induction* [13], 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* [13], 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 [14] using a verification logic that is generalized to hybrid systems accordingly [8, 9]. For combining multiple local differential invariants into a global invariant in a sound way, we exploit the closure properties of the underlying verification logic [8, 9] by forming appropriate logical combinations of multiple safety statements. In addition, we introduce a *differential saturation process* that refines the hybrid 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 [15]. We validate our algorithm by verifying *aircraft roundabout maneuvers* [16, 11] and train control applications [17].

The major contribution in this work is the fixedpoint algorithm for computing differential invariants coupled with a differential saturation process. We show that it can verify realistic applications that were out of scope for related invariant approaches [18–20] or [1, 3, 6], both for theoretical reasons [9, 13] and scalability.

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) [1]. 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). 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.



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

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

Hybrid Programs. In order to represent HA [1] textually as an HP, we represent each discrete and continuous transition as a sequence of statements, with a nondeterministic choice ($\cup$) between these transitions. For instance, the second line in Fig. 1 represents a continuous transition. It tests (denoted by $?q = on$) if the current location q is *on*, and then follows a differential equation restricted to invariant region $x \leq 9$ (i.e., the conjunction $x' = 1 \wedge x \leq 9$). The third line tests the guard $x \geq 5$ when in state *on*, resets x by a discrete assignment, and then changes location q to *off*. The * at the end 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 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 $\mathbb{Q}$ 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 Tab. 1. 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 $x_1' = \theta_1, \ldots, x_n' = \theta_n$ 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.,

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

notation	statement	effect
$x := \theta$	discrete assignment	assigns term θ to variable $x \in V$
$x := random$	nondet. assignment	assigns any real value to $x \in V$
$x_1' = \theta_1 \wedge \ldots$ $\cdots \wedge x_n' = \theta_n \wedge H$	continuous evolution	diff. equations for $x_i \in V$ and 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 β
α^*	nondet. repetition	repeats HP α n-times for any $n \in \mathbb{N}$

no change) if H is true in the current state and that of *abort*, otherwise. 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.

Formulas of $\mathsf{d}\mathcal{L}$. Our verification algorithm repeatedly decomposes and recombines HP. As a logical framework where these operations are sound, we use a logic in which simultaneous correctness properties about multiple subsystems are expressible. The *differential dynamic logic* $\mathsf{d}\mathcal{L}$ [8, 9] 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 ($\mathsf{d}\mathcal{L}$ formulas). *The* formulas of $\mathsf{d}\mathcal{L}$ *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 Tab. 1):*

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

A Hoare-triple $\{\psi\}\alpha\{\phi\}$ can be expressed as $\psi \rightarrow [\alpha]\phi$, which is true iff all states reachable by HP α satisfy ϕ when starting from an initial state that satisfies ψ. Unlike Hoare-logics, dynamic logics are closed under logical connectives [21]. 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 $[\alpha]\phi$ can be decomposed soundly into $[\alpha_1]\phi_1 \wedge [\alpha_2]\phi_2$, when $[\alpha]\phi$ 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 the verification algorithm with input $[\alpha_i]\phi_i$ yields $\tilde{\phi}_i$, these can be recombined soundly to the verification result $\tilde{\phi}_1 \wedge \tilde{\phi}_2$ for $[\alpha]\phi$. By the semantics of $\mathsf{d}\mathcal{L}$, 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, $\mathsf{d}\mathcal{L}$ 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 $[\alpha_i]\phi_i$ true. The *semantics* of $\mathsf{d}\mathcal{L}$ and HP is a Kripke semantics [8, 9].

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* [13]). 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 [14] with the lifting of verification logics to hybrid systems from [8, 9]. We introduce a verification algorithm that computes invariants of a system as fixedpoints of safety constraints on subsystems. We exploit the fact that HP can be decomposed into subsystems and that $\mathsf{d}\mathcal{L}$ can combine safety statements about multiple subsystems simultaneously.

A *safety statement* corresponds to a $\mathsf{d}\mathcal{L}$ 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 [9]. Our verification algorithm defines the function $prove(\psi \rightarrow [\alpha]\phi)$ for verifying this safety statement recursively.

3.1 Verification by Symbolic Decomposition

The cases of *prove* where $\mathsf{d}\mathcal{L}$ enables us to verify a property of an HP directly by decomposing it into a property of its parts [9] 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 $\hat{x}$ for x in $\phi_x^{\hat{x}}$: E.g., $x \geq 2 \rightarrow [x := x - 1]x \geq 0$ is shown by proving $x \geq 2 \wedge \hat{x} = x - 1 \rightarrow \hat{x} \geq 0$. Our implementation uses optimizations to avoid auxiliary variables [9]. State checks $?H$ are shown by assuming the test succeeds, i.e., H holds true (line 3), nondeterministic choices split into their alternatives (line 4), sequential compositions are proven using nested modalities (line 6), and random assignments by universal quantification (line 7).

The base case in line 8, where ϕ is a formula of first-order real arithmetic, can be proven by real quantifier elimination [5]. Despite its complexity, this can remain feasible, because the formulas resulting from our algorithm do not depend on the solutions of differential equations but only their right-hand sides. Using a temporary form of Skolemization together with Deskolemization, quantifier elimination can be lifted to eliminate quantifiers from $\mathsf{d}\mathcal{L}$ formulas [9].

The algorithm in Fig. 2 recursively reduces safety of HP to 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 by discrete induction and continuous evolutions 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 $\psi \rightarrow F$ is valid) and finally entails the postcondition ϕ (i.e., $F \rightarrow \phi$). The cases differ in their induction step.

Definition 2 (Discrete induction). *Formula F is a* (discrete) invariant *of $\psi \rightarrow [\alpha^*]\phi$ iff the following formulas are valid: $\psi \rightarrow F$ (induction start), and $F \rightarrow [\alpha]F$ (induction step). An invariant is* sufficiently strong *if $F \rightarrow \phi$ is valid.*

```
1  function prove(ψ → [x := θ]φ):
2    return prove(ψ ∧ x̂ = θ → φ_x^x̂) where x̂ is a new auxiliary variable
3  function prove(ψ → [?H]φ): return prove(ψ ∧ H → φ)
4  function prove(ψ → [α ∪ β]φ):
5    return prove(ψ → [α]φ) and prove(ψ → [β]φ) /* thus ψ → [α]φ ∧ [β]φ */
6  function prove(ψ → [α; β]φ): return prove(ψ → [α][β]φ)
7  function prove(ψ → [x := random]φ): return prove(ψ → ∀x φ)
8  function prove(ψ → φ) where isFirstOrder(φ):
9    return QuantifierElimination(ψ → φ)
```

Fig. 2. dℒ-based verification by symbolic decomposition

Definition 3 (Continuous invariants). *Let $\mathcal{D}$ be a differential equation. Formula F is a* continuous invariant *of $\psi \to [\mathcal{D} \wedge H]\phi$ iff the following formulas are valid: $\psi \wedge H \to F$ (induction start), and $F \to [\mathcal{D} \wedge H]F$ (induction step). Again, a continuous invariant is* sufficiently strong *if $F \to \phi$ is valid.*

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 [13]. This condition is much easier to check than a reachability property ($F \to [\mathcal{D} \wedge H]F$) of a differential equation. Applications like aircraft maneuvers need invariants with mixed equations and inequalities. Thus, we generalize directional derivatives from functions to logical formulas.

Definition 4 (Differential induction). *Let the differential equation system $\mathcal{D}$ be $x_1' = \theta_1 \wedge \cdots \wedge x_n' = \theta_n$. Formula F is a* differential invariant *of $\psi \to [\mathcal{D} \wedge H]\phi$ iff the following formulas are valid: $\psi \wedge H \to F$ and $H \to \nabla_{\mathcal{D}} F$, 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 $\mathcal{D}$ (the partial derivative of b by x_i is $\frac{\partial b}{\partial x_i}$):*

$$\nabla_{\mathcal{D}} F \;\equiv\; \bigwedge_{(b \sim c) \in F} \left(\left(\sum_{i=1}^{n} \frac{\partial b}{\partial x_i} \theta_i \right) \sim \left(\sum_{i=1}^{n} \frac{\partial c}{\partial x_i} \theta_i \right) \right) \quad \text{for } \sim \in \{=, \geq, >, \leq, <\}.$$

Proposition 1 (Principle of differential induction [13]). *All differential invariants are continuous invariants.*

See [13] for the theory of differential invariants and [15] for specific proofs. 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 are pointing inside the (possibly unbounded) region consisting of the points where F is true. In Sections 3.4–3.6, we present algorithms for finding differential invariants for differential equations, and for finding global invariants for repetitions.

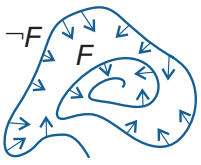


Fig. 3. Differential invariant F

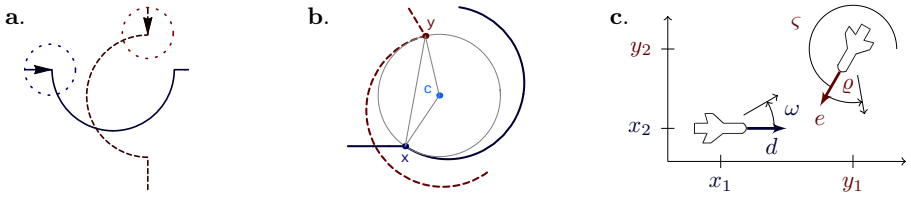


Fig. 4. Roundabout maneuvers for air traffic collision avoidance

3.3 Example: Flight Dynamics in Air Traffic Collision Avoidance

Aircraft collision avoidance maneuvers resolve conflicting flight paths, e.g., by roundabout maneuvers [16], see Fig. 4a–b. Their nontrivial dynamics makes safe separation of aircraft difficult to verify [16, 22–24, 11, 25]. 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. 4c (with $\vartheta = 0$). Their dynamics is determined by their linear speeds $v, u \in \mathbb{R}$ and angular speeds $\omega, \varrho \in \mathbb{R}$, see [16]:

$$x_1' = v\cos\vartheta \quad x_2' = v\sin\vartheta \quad \vartheta' = \omega \quad y_1' = u\cos\varsigma \quad y_2' = u\sin\varsigma \quad \varsigma' = \varrho \tag{1}$$

In safe flight configurations, aircraft are separated by at least distance p:

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

To handle the transcendental functions in (1), we axiomatize sin and cos by differential equations and reparametrize the system using a linear velocity vector $d = (d_1, d_2) := (v\cos\vartheta, v\sin\vartheta) \in \mathbb{R}^2$, which describes both the linear velocity $\|d\| := \sqrt{d_1^2 + d_2^2} = v$ and orientation of the aircraft in space, see Fig. 4c:

$$\left[\begin{array}{lllll} x_1' = d_1 & x_2' = d_2 & d_1' = -\omega d_2 & d_2' = \omega d_1 & t' = 1 \\ y_1' = e_1 & y_2' = e_2 & e_1' = -\varrho e_2 & e_2' = \varrho e_1 & s' = 1 \end{array}\right] \tag{$\mathcal{F}$}$$

Equations ($\mathcal{F}$) and (1) are equivalent up to reparameterization [13]. We add clock variables t, s that we need for synchronizing collision avoidance maneuvers [15]. By a simple computation, $d_1^2 + d_2^2 \geq a^2$ is a differential invariant of ($\mathcal{F}$):

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

3.4 Local Fixedpoint Computation for Differential Invariants

Fig. 5 depicts the fixedpoint algorithm for constructing differential invariants for each continuous evolution $\mathcal{D} \wedge H$ with a differential equation system $\mathcal{D}$. The

```
1  function prove(ψ → [D ∧ H]φ):
2    if prove(∀cl(H → φ)) then return true /* property proven */
3    for each F ∈Candidates(ψ → [D ∧ H]φ, H) do
4      if prove(ψ ∧ H → F) and prove(∀cl(H → ∇D F)) then
5        H := H ∧ F        /* refine by differential invariant */
6        goto 2;           /* repeat fixedpoint loop */
7    end for
8    return "not provable using candidates"
```

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

algorithm in Fig. 5 (called *Differential Saturation*) successively refines the domain H by differential invariants until saturation, i.e., H accumulates enough information to become a strong invariant that implies postcondition ϕ (line 2). If domain H already entails ϕ, then $\psi \rightarrow [\mathcal{D} \wedge H]\phi$ is proven (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). The refinement of the dynamics at line 5 is correct by the following proposition, using that the conditions in line 4 imply that F is a differential invariant and, thus, a continuous invariant by Proposition 1, see proofs [15, 13].

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

This progressive differential saturation turns out to be crucial in practice. For instance, the aircraft separation property (2) cannot be proven until ($\mathcal{F}$) has been refined by invariants for d and e, because these determine x' and y'.

Function *Candidates* determines candidates for induction (line 3) depending on transitive differential dependencies, as will be explained in Section 3.5. When these are insufficient for proving $\psi \rightarrow [\mathcal{D} \wedge H]\phi$, 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 (that satisfy H), see [15] for improvements.

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. Our algorithm enriches ψ successively with more precise information about the symbolic prestate as obtained by the symbolic decompositions and proof steps in Fig. 2 and 5. We first look for invariant symbolic state information in ψ and ϕ by selecting subformulas that are not yet contained in H. In practice, this gives good candidates for highly parametric hybrid systems.

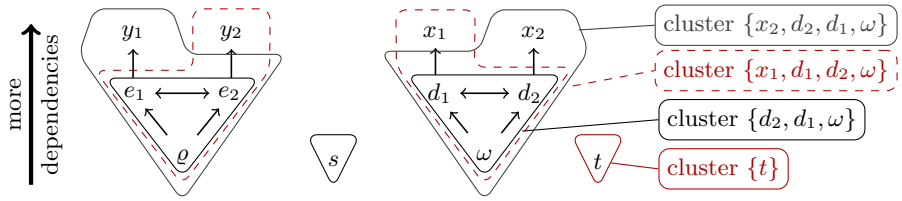


Fig. 6. Differential dependencies (arrows) and (triangular) variable clusters of $(\mathcal{F})$

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

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

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* contains $k+1$ candidates with $k\binom{n+d}{d}$ formal parameters $a^{(l)}_{i_1,\ldots,i_n}$, which are existentially quantified. Existence of a common satisfying instantiation for these parameters can be expressed by adding $\exists a^{(l)}_{i_1,\ldots,i_n}$ to the resulting $\mathsf{d}\mathcal{L}$ formulas. For this to be feasible, the number of parameters is crucial, which we minimize by respecting (differential) dependencies.

To accelerate the differential saturation process in Section 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: E.g., (2) is only provable after the dynamics has been refined with invariants for d and e. 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 for x' (or transitively so). The resulting set *depend*$(\mathcal{D})$ of dependencies is the transitive closure of $\{(x, y) \mid (x' = \theta) \in \mathcal{D}$ 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 *depend*($\mathcal{D}$), i.e., variables of V only depend on variables 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 *ParaForm*(k, d, V) starting with candidates whose variables lie in small clusters V. Thus, the differential invariant $d_1^2 + d_2^2 \geq a^2$ of Section 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 $\mathtt{d}\mathcal{L}$, we can adapt the algorithm in Fig. 5 easily to obtain a fixedpoint algorithm for loops ($\psi \rightarrow [\alpha^*]\phi$) in place of continuous evolutions ($\psi \rightarrow [\mathcal{D} \wedge H]\phi$): In line 4 of Fig. 5, we replace the induction step from Def. 4 by the step for loops (Def. 2). As an optimization, invariants H of previous iterations can be exploited as refinements of the hybrid system dynamics:

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

See [15] for a proof. The induction step from Proposition 3 can generally be proven faster, because it is a weaker property than that of Def. 2.

To adapt our approach from Section 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 is a direct *control-flow dependency*, if, for any term θ, $x := \theta$ or $x' = \theta$ occurs in α after a $?H$ containing y.

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 result in stronger preconditions ψ for the local algorithm. Thus, the local fixedpoint algorithm should stop when it cannot prove its postcondition, either because of a counterexample or because it runs out of candidates for differential invariants. As in the work of Prajna [20], the degrees of parametric invariants, therefore, need to be bounded and increased iteratively. As in [20], there is no natural measure for how these degrees should be increased. Instead, here, we exploit the fact that the candidates of *Candidates* are independent and we explore them in parallel with fair time interleaving.

3.8 Soundness

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

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

See [15] for a proof. 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, they are always expressible in $\mathsf{d}\mathcal{L}$ [9]).

4 Experimental Results: Aircraft Roundabout Maneuver

As an example with nontrivial dynamics, we analyze aircraft roundabout maneuvers [16]. Curved flight as in roundabouts is challenging for verification, because of its transcendental solutions. The maneuver in Fig. 4a from [16] and the maneuver in Fig. 4b from [11, 13] are not flyable, because they still involve a few instant turns. A flyable roundabout maneuver without instant turns is depicted in Fig. 7. We verify safety properties for most (but not yet all) phases of Fig. 7 and provide verification results in Tab. 2, see [15]. Finally, note that the required invariants for the roundabout maneuver cannot even be found from Differential Gröbner Bases [26].

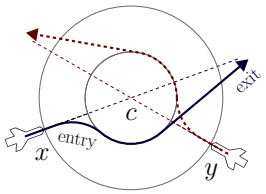


Fig. 7. Flyable aircraft roundabout

Verification results for roundabout aircraft maneuvers [16, 24, 11, 13, 15] and the European Train Control System (ETCS) [17] are in Tab. 2. Results are from a 2.6GHz AMD Opteron with 4GB memory. Memory consumption of quantifier elimination is shown in Tab. 2, excluding the front-end. The results are only slightly worse on a 1.7GHz Pentium M laptop with 1GB. We handle *all* variables symbolically. The dimension of the continuous state space is indicated.

5 Related Work

Other authors [18–20] already argued that invariant techniques scale to more general dynamics than explicit reach-set computations or techniques that require solutions for differential equations [3, 6, 8]. However, they cannot handle hybrid systems with inequalities in initial sets or switching surfaces [18, 19], which occur

in most real applications like aircraft maneuvers. *Barrier certificates* [20] only work for inequalities, but invariants of roundabout maneuvers require mixed equations and inequalities [13]. Prajna et al. [20] 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 already requires solving a 5848-dimensional optimization problem for ETCS [17] and a 10005-dimensional problem for roundabouts with 5 aircraft.

Tomlin et al. [16] 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 unsound [11]. Differential invariants, instead, work for 28-dimensional systems.

Straight-line aircraft maneuvers have been analyzed by geometrical meta-level reasoning [23, 25]. We directly verify the hybrid flight dynamics, including curved roundabout maneuvers instead of straight-line maneuvers with non-flyable instant turns. A few approaches [22, 24] have been undertaken to Model Check if there are *orthogonal* collisions in discretizations of roundabout maneuvers. However, the counterexamples found by our model checker in previous work [11] show that *non-orthogonal* collisions can happen in these maneuvers.

Tools like HyTech, PHAVer, CheckMate, or other approaches [1, 3, 6] cannot handle our applications with nonlinear switching, nonlinear discrete and continuous dynamics, and high-dimensional state spaces.

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. We validate our algorithm on challenging *roundabout collision avoidance maneuvers* for aircraft and on collision avoidance protocols for trains.

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. Thus, it probably performs worse for systems that violate locality principles. We want to validate this in further experiments and analyze scalability.

References

1. Henzinger, T.A.: The theory of hybrid automata. In: LICS, IEEE (1996) 278–292

2. Davoren, J.M., Nerode, A.: Logics for hybrid systems. Proc. IEEE **88**(7) (2000)
3. Fränzle, M.: Analysis of hybrid systems. In Flum, J., Rodríguez-Artalejo, M., eds.: CSL. Volume 1683 of LNCS., Springer (1999) 126–140
4. Alur, R., Pappas, G.J., eds.: HSCC. Volume 2993 of LNCS., Springer (2004)
5. Collins, G.E., Hong, H.: Partial cylindrical algebraic decomposition for quantifier elimination. J. Symb. Comput. **12**(3) (1991) 299–328
6. Piazza, C., Antoniotti, M., Mysore, V., Policriti, A., Winkler, F., Mishra, B.: Algorithmic algebraic model checking I: Challenges from systems biology. In Etessami, K., Rajamani, S.K., eds.: CAV. Volume 3576 of LNCS., Springer (2005) 5–19
7. Anai, H., Weispfenning, V.: Reach set computations using real quantifier elimination. In Benedetto, M.D.D., Sangiovanni-Vincentelli, A.L., eds.: HSCC. Volume 2034 of LNCS., Springer (2001) 63–76
8. Platzer, A.: Differential dynamic logic for verifying parametric hybrid systems. In Olivetti, N., ed.: TABLEAUX. Volume 4548 of LNCS., Springer (2007) 216–232
9. Platzer, A.: Differential dynamic logic for hybrid systems. J. Autom. Reasoning (2008)
10. Asarin, E., Dang, T., Girard, A.: Reachability analysis of nonlinear systems using conservative approximation. In Maler, O., Pnueli, A., eds.: HSCC. Volume 2623 of LNCS., Springer (2003) 20–35
11. Platzer, A., Clarke, E.M.: The image computation problem in hybrid systems model checking. [27] 473–486
12. Donzé, A., Maler, O.: Systematic simulation using sensitivity analysis. [27] 174–189
13. Platzer, A.: Differential algebraic dynamic logic for differential algebraic programs. Submitted (2007)
14. Clarke, E.M.: Program invariants as fixedpoints. Computing **21**(4) (1979) 273–294
15. Platzer, A., Clarke, E.M.: Computing differential invariants of hybrid systems as fixedpoints. Technical Report CMU-CS-08-103, Carnegie Mellon University (2008)
16. Tomlin, C., Pappas, G.J., Sastry, S.: Conflict resolution for air traffic management: a study in multi-agent hybrid systems. IEEE T. Automat. Contr. **43**(4) (1998)
17. Platzer, A., Quesel, J.D.: Logical verification and systematic parametric analysis in train control. In Egerstedt, M., Mishra, B., eds.: HSCC. LNCS, Springer (2008)
18. Sankaranarayanan, S., Sipma, H., Manna, Z.: Constructing invariants for hybrid systems. [4] 539–554
19. Rodríguez-Carbonell, E., Tiwari, A.: Generating polynomial invariants for hybrid systems. In Morari, M., Thiele, L., eds.: HSCC. Volume 3414 of LNCS., Springer (2005) 590–605
20. Prajna, S., Jadbabaie, A., Pappas, G.J.: A framework for worst-case and stochastic safety verification using barrier certificates. IEEE T. Automat. Contr. **52**(8) (2007)
21. Harel, D., Kozen, D., Tiuryn, J.: Dynamic logic. MIT Press (2000)
22. Massink, M., Francesco, N.D.: Modelling free flight with collision avoidance. In: ICECCS, IEEE Computer Society (2001) 270–280
23. Dowek, G., Muñoz, C., Carreño, V.A.: Provably safe coordinated strategy for distributed conflict resolution. In: AIAA Conference Proc. AIAA-2005-6047. (2005)
24. Damm, W., Pinto, G., Ratschan, S.: Guaranteed termination in the verification of LTL properties of non-linear robust discrete time hybrid systems. In Peled, D., Tsay, Y.K., eds.: ATVA. Volume 3707 of LNCS., Springer (2005) 99–113
25. Hwang, I., Kim, J., Tomlin, C.: Protocol-based conflict resolution for air traffic control. Air Traffic Control Quarterly **15**(1) (2007)
26. Mansfield, E.L.: Differential Gröbner Bases. PhD thesis, University Sydney (1991)
27. Bemporad, A., Bicchi, A., Buttazzo, G., eds.: HSCC. Volume 4416 of LNCS., Springer (2007)