

Hybrid Theorem Proving of Aerospace Systems: Applications and Challenges¹

Khalil Ghorbal,² Jean-Baptiste Jeannin,³ Erik Zawadzki,⁴

André Platzer⁵ and Geoffrey J. Gordon⁶
*Carnegie Mellon University, 5000 Forbes Avenue,
Gates Hillman Center, Pittsburgh, PA 15213.*

Peter Capell⁷
*Software Engineering Institute, Carnegie Mellon,
4500 Fifth Avenue Pittsburgh, PA 15213*

Complex software systems are becoming increasingly prevalent in aerospace applications, in particular to accomplish critical tasks. Ensuring the safety of these systems is crucial, while they can have subtly different behavior under slight variations in operating conditions. In this paper we advocate the use of formal verification techniques and in particular theorem proving for hybrid software-intensive systems as a well-founded complementary approach to the classical aerospace verification and validation techniques such as testing or simulation. As an illustration of these techniques, we study a novel lateral mid-air collision avoidance maneuver in an ideal setting, without accounting for the uncertainties of the physical reality. We then detail the challenges that naturally arise when applying such technology to industrial-scale applications and our proposals on how to address these issues.

I. Introduction

Aircraft software is becoming increasingly complex, as shown by rising development costs. These software systems are safety-critical: their failure can lead to major catastrophes. Past examples include a collision between a military airplane and a drone over Afghanistan in 2011 [1], or the 2002 tragic mid-flight collision in Überlingen [2]. The Überlingen collision was not caused by a system failure but rather by conflicting orders between air traffic control and onboard systems, which illustrates the significance of increasing trust in collision-avoidance systems. Such incidents and accidents demonstrate that the certification of these systems needs to respond to increased system complexity by becoming more exhaustive in the methods used to assure software. Relying on testing alone cannot achieve the desired requirements, as even the most careful testing campaign can miss some errors, corner cases and rare events.

Significant progress has recently been achieved to increase the reliability of critical software. Major aircraft manufacturers, such as Airbus, are now relying heavily on model-based architecture, model-based design, and certified code generation [3]. It can be argued that analysis using models is currently the only approach to ensuring a safe development of highly-reliable complex systems [4]. There are numerous immediate benefits to this approach such as model reusability, compositional

¹ This material is based upon work supported by the National Science Foundation under NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181, and under Grant Nos. CNS-1035800, by the Army Research Office under Award No. W911NF-09-1-0273 and the Federal Aviation Administration under grant number DTFAWA-11-C-00074.

² Postdoctoral Fellow, Department of Computer Science, kghorbal@cs.cmu.edu.

³ Postdoctoral Fellow, Department of Computer Science, jeannin@cs.cmu.edu.

⁴ Ph.D. Student, Department of Computer Science, epz@cs.cmu.edu.

⁵ Associate Professor, Department of Computer Science, aplatzer@cs.cmu.edu.

⁶ Associate Professor, Department of Machine Learning, ggordon@cs.cmu.edu.

⁷ Senior Member of Engineering Staff and Adjunct Professor, Department of Computer Science, psc@sei.cmu.edu.

design and certification cost reduction. However, the most important impact for safety is that model-based design describes the system by many layers of abstraction, with each layer corresponding to a set of refined models. These abstract models can then be used to reason formally about increasingly complex systems and to formally verify their properties.

Formal verification and validation of systems involves mathematically well-founded methodologies for specifying and verifying the system. In general, formal techniques have dual goals: either formally proving that the system meets its requirements or searching for a specific counterexample that falsifies a given property. Verification questions are not necessarily limited to functional requirements. Very often, requirements such as safety or timing are crucial to certifying the system.

Many large-scale formal verification attempts have been successful in the past decade: for example, hardware model checking of the Intel processor [5] is an outstanding achievement that increased the reliability of the basic computation units that any software relies on. More recently, formal verification techniques were applied to check for numerical runtime errors in the Airbus A380 [6, 7]. Such achievements demonstrate improvements in the underlying theories and related technologies of formal verification. Formal verification is becoming crucial for verification, validation and certification of all safety-critical systems, and aerospace systems are no exception.

Formal methods are complementary to—and not a replacement for—the existing classical verification, validation, and certification approaches, which are predominantly based on testing and simulation—including flight tests. Formal methods, including the hybrid systems theorem proving approach presented in this paper, provide an exhaustive approach, ensuring that a property is satisfied for all possible runs of the system. This contrasts with a testing approach that usually tests a limited number of runs of a system, and could easily miss some corner cases. Exhaustive testing is limited to relatively small systems—even a system with 7 variables, each with 1000 possible values requires an intractable 10^{21} test cases—and it cannot handle continuous variables—they generate an infinite number of tests. On the other hand, the hybrid systems theorem proving approach presented in this paper can handle infinitely many cases at once, as well as continuous variables.

In this paper, we focus on *hybrid systems theorem proving* [8, 9], which has been applied to many difficult problems where all other (formal and classical) methods have failed [10]. A *hybrid system* [11] is a system that involves both discrete and continuous behaviors—such as with the integration of discrete computer algorithms and continuous flight dynamics of an aircraft. Understanding such interactions is crucial for ensuring that the system remains within safe boundaries, *e.g.*, the aircraft is not close to stalling, and is never too close to any other aircraft. Without understanding the continuous motion of an aircraft in space, it is impossible to tell whether an aircraft might stall. But one also needs to consider the discrete control decisions that determine the control input for the flight dynamics. As an illustration of the use of hybrid systems theorem proving for aircraft, this paper focuses on the property of separation between two or more aircraft.

In Section II, we recall differential dynamic logic, a logic to reason about hybrid systems. This theoretical framework is used to model and verify the system. As an illustrative example showing the use of differential dynamic logic for aircraft, we study a novel *lateral* mid-air collision avoidance maneuver in Section III. The term lateral refers to the fact that unlike current collision avoidance systems where the pilot is advised to either climb or descend, we explore the possibility of avoiding a potential collision by advising the pilot to maneuver laterally (steering) without changing his altitude. To keep this particular study easy to follow, it does not account for any uncertainty (sensor noise, wind, etc.). Section IV details how we can push our reasoning further within a more realistic noisy environment. We also give an overview of the current and future challenges facing the formal verification community toward a successful application of these capabilities in full-scale industrial aerospace applications. The related work is discussed in Section V.

II. Theorem Proving of Hybrid Systems

Our group at Carnegie Mellon University has developed tools and techniques to model and prove properties of dynamic systems. Our main theoretical tool is *differential dynamic logic* [8, 9, 12], a logic for specifying and verifying properties of hybrid systems. The hybrid systems theorem prover KeYmaera [13] implements those and other logics in an automated and interactive theorem prover.

A. Differential Dynamic Logic

Differential dynamic logic [8, 9, 12] is the logic of hybrid systems, i.e., systems with interacting discrete and continuous dynamics. This logic can model the system, e.g., motion of aircraft, cars or trains, as well as expressing properties about such systems. Because differential dynamic logic comes with a proof calculus [8, 9, 12], it is both a specification and a verification language for hybrid systems.

We model systems being studied with *hybrid programs*, a programming language for hybrid systems. Hybrid programs combine differential equations and conventional computer program constructs. *Systems of ordinary differential equations* model continuous evolution of variables, and can be used to model relevant motion equations of aircraft. *Instantaneous assignments* can be used to represent the changes in the set-values that an autopilot performs. *Tests* are used to represent known constraints on the evolution of the system, either physical constraints on parameters or conditions on the system execution. *Sequential composition* is used to execute one hybrid program after another finishes.

A central feature of hybrid programs is *nondeterminism*: it enables modeling and proving properties about systems who have parts not controlled by the program. *Nondeterministic choice* is encountered when various control decisions can be made without any a priori knowledge about which one will be triggered first. This is useful to describe, for example, the effect that other aircraft could suddenly climb or descend. We handle this by accounting for all possible choices. *Nondeterministic repetition* is used to model control feedback loops. These operations are enough to model all classical computer programs (including `if` and `while` statements), as well as all hybrid systems.

Once hybrid programs have been defined, properties about the system can be expressed through differential dynamic logic itself. Differential dynamic logic is a combination of constructs used in arithmetic and logic. This logic can express comparisons (equality and inequalities) of terms; negation (not \neg), conjunction (and \wedge) and disjunction (or \vee) of formulas; universal (for all \forall) and existential quantifiers (exists \exists), expressing properties such as “for all real numbers, a certain property is true” or “there exists a real number such that a certain property is true”; and universal ($[\alpha]$) and existential modalities ($\langle\alpha\rangle$) indexed by hybrid programs α expressing properties such as “a property is always true after executing a certain hybrid program α ”, or “a certain hybrid program α has at least one execution making a certain property true”. Since those logical operators can be combined and nested in any way, this results in a flexible logic for specifying and verifying a rich set of complex properties of hybrid systems.

A precise semantics, i.e., mathematical meaning, for hybrid programs and differential dynamic logic is beyond the scope of this article but can be found in the literature [8, 9, 12]. These references also provide rules to reason about the logic and to verify systems without having recourse to their semantics. Those rules form a *proof calculus* for the logic. They are purely based on the syntax of terms, which means that they can be implemented and used in a computer, for example, in the theorem prover KeYmaera. The proof calculus is sound, i.e., it is faithful to the semantics. Furthermore, the proof calculus is complete relative to properties of differential equations, i.e., every semantically valid formula is provable in the proof calculus from elementary properties of differential equations.

B. The KeYmaera Theorem Prover for Hybrid Systems

KeYmaera [13] is a hybrid verification tool for hybrid systems implementing differential dynamic logic [8] and its extensions. It is an automatic and interactive theorem prover. It is automatic as it implements a set of proof strategies to automatically discharge the proof. It is also interactive, as a user can help the tool by providing extra hints when the automatic strategies fail.

KeYmaera can be used to prove that a hybrid system satisfies its specification (correctness), does not show unsafe behavior (safety), remains controllable (controllability), responds within a specified time (reactivity), and eventually achieves a specified goal (liveness).

KeYmaera has been used successfully for verifying properties of systems involving cars, trains, aircraft, and robots: local lane controllers for highway car traffic [14], left-turn assist controllers for cars at intersections [15], intelligent speed adaptation for variable speed limit control and incident management by traffic centers on highways [16], cooperation protocols of the European Train Control

System [17], airplane collision avoidance [18, 19], obstacle avoidance for ground robots [20], and force feedback to the surgeon from a surgical robot [21].

In Section III we discuss a variation of one of those systems, the curved flight collision avoidance maneuver, in order to provide a sense of the capabilities of the approach. This analysis yields new results for the analysis of collision avoidance protocols and generalizes previous related work [18].

III. Case Study: Curved Flight Collision Avoidance Maneuvers

As airspace becomes more crowded, the design of safe and automated traffic management of airplanes becomes even more critical. Such systems must be hybrid: the motion of each airplane is described by a set of continuous differential equations, while each pilot is allowed to make discrete decisions to control the aircraft. In this section we analyze an abstract model of an encounter between two aircraft. The findings generalize to prevent collisions between three or more aircraft. Our analysis focuses on proving under which conditions an aircraft, henceforth called the *ownship*, and an intruder could possibly collide. If a possible collision is detected we explicitly give a set of safe, flyable, choices that prevent the collision from happening.

Naturally, the analysis of the real system requires further refinement of the model to account for real-world settings: flight disturbances, sensor uncertainties, human behaviors, implementation issues, etc. Our current approaches to deal with such refinements are the topic of Section IV.

Section III A recalls the differential equations that describe Dubins trajectories in absolute and relative frames. In Section III B, we abstract the trajectories using automatically generated functional invariants (first integrals), i.e. non-trivial functions that are constant along all solutions of the differential equations. We then use these invariants to generate a safety monitor. The monitor will play a crucial role in our formal analysis, presented in Section III C, as it serves to synthesize safe commands for the system.

A. Flight Maneuvers Using Dubins Curves

Our conflict resolution model is based on curved maneuvers, where we allow for curved paths in addition to straight-line motion. For a fixed altitude, the planar path of an aircraft is modeled by a set of differential equations that captures the time evolution of the position and orientation of an aircraft. We use kinematic models based on the standard Dubins vehicle models for aircraft [22–24].

Let us consider two aircraft numbered 1 and 2. We suppose that aircraft 1 is the ownship and aircraft 2 is an intruder. Aircraft i is modeled by a single point in the plane with a safe surrounding disc. Its configuration is entirely determined by three variables: $(x_i, y_i) \in \mathbb{R}^2$ for its absolute position and $\theta_i \in [0, 2\pi)$ for its direction with respect to the x -axis (Figure 1). The motion of each aircraft with respect to an inertial frame is described by the system of ordinary differential equations given in equation (1). The time variable t is implicit: its time derivative is equal to 1 for all frames: time evolves linearly and is absolute. The angular and linear velocities of the aircraft (ω_i and $v_i > 0$) are considered as parameters. The angular velocity ω_i acts as a controlled input, stemming from the control decisions made by the pilot. This allows the airplane to fly following a sequence of circular paths ($\omega_i \neq 0$) and straight lines ($\omega_i = 0$): for instance, in Figure 1 the dotted lines show the projected trajectories of two aircraft with potentially conflicting path: aircraft 1 is flying in a straight line while aircraft 2 is performing a *hold*—a standard procedure with two circular and two straight-line legs that an aircraft maintains while waiting. For a real application, one needs to account for the engine capabilities and functional limits—for instance stalling—on the angular and linear velocities which lead to additional constraints on ω_i and v_i .

The ownship is equipped with the following hybrid program that captures the core principles of the collision avoidance system (`cas`):

$$\text{pilot} \equiv \omega_1 := *; \quad (2)$$

$$\text{cas} \equiv (\text{pilot}; ?\text{monitor}; \text{evolve})^* \quad (3)$$

The pilot—or autopilot—of the ownship first sets the angular velocity ω_1 of the airplane to an arbitrary real number (the wildcard `*` in equation (2)). The expression `evolve` encodes the set of differential equations describing the evolution of the intruder relative to the ownship. We use the

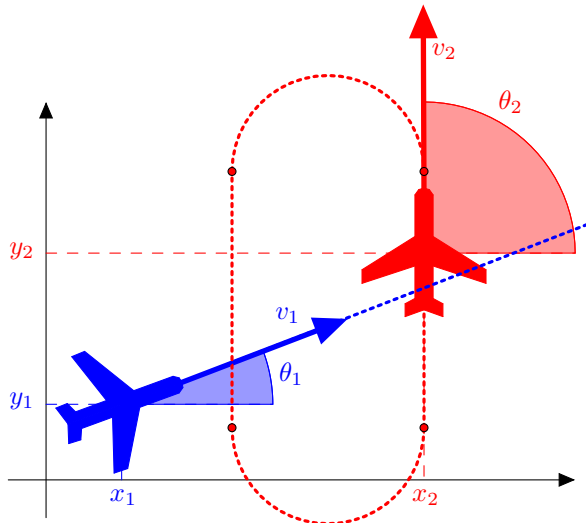


Fig. 1 Planar motion of two airplanes with respect to an inertial frame of reference. The dotted lines show an example of flyable trajectories that the model can describe.

$$\begin{aligned} \dot{x}_i &= v_i \cos \theta_i \\ \dot{y}_i &= v_i \sin \theta_i \\ \dot{\theta}_i &= \omega_i \end{aligned} \quad (1)$$

Fig. 2 Dubins motion for the airplane i , for $i = 1, 2$.

relative motion of the intruder with respect to the ownship, by representing the motion equation relative to a frame fixed to the ownship (Figure 3). With respect to this relative frame, the set of ordinary differential equations is given in equation (4), where x , y and θ denote respectively the relative position and direction of the intruder with respect to the ownship. The relative equations are obtained from the absolute ones (1) by computing the time derivative of the relative position of the intruder with respect to the ownship while accounting for the angular velocity (along the axis orthogonal to the plane) of the rotating frame fixed to the ownship.

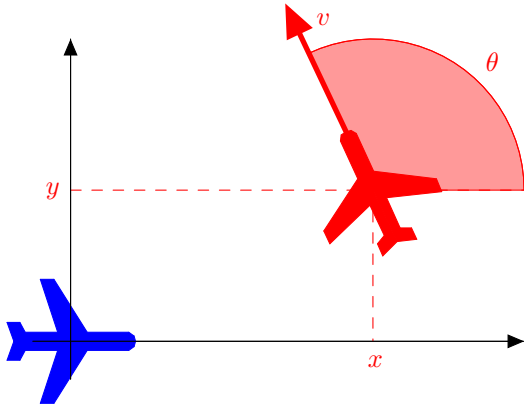


Fig. 3 Planar motion of the intruder with respect to a relative frame embedded in aircraft 1.

$$\begin{aligned} \dot{x} &= v_2 \cos \theta - v_1 + \omega_1 y \\ \dot{y} &= v_2 \sin \theta - \omega_1 x \\ \dot{\theta} &= \omega_2 - \omega_1 \end{aligned} \quad (4)$$

Fig. 4 Dubins motion of the intruder relative to the ownship.

The next section discusses our approach to generate an expression for `monitor` (equation (3)). We want the expression for `monitor` to ensure that the current and future behavior of the pilot always preserves safety.

B. Safety Monitoring Using Differential Invariants

In equation (3), the role of the monitor is to check whether the chosen action (ω_1) of the ownship is safe assuming that the evolution of the involved aircraft is governed by the differential equations given in equation (4). The symbol (?) tests whether the safety condition is satisfied. The test can be regarded as a filter that rules out unsafe decisions—a failed test should be understood as an unsafe, and immediately discarded, decision on ω_1 . Hence, **monitor** imposes conditions on the safe choices for ω_1 . If the test succeeds, the airplanes will evolve following their respective dynamics described by the set of ordinary differential equations given in (4). If, however, there are no safe choices for ω_1 , then the collision avoidance system reaches its operational limit. This case will be further discussed at the end of this section. Finally, the * operator, in the exponent of equation (3), means that this sequence of (i) discrete control, (ii) monitoring, and (iii) continuous evolution, can be repeated any number of times. It, thus, describes the operational nature of the collision avoidance system, which keeps on repeating any number of times as necessary. Indeed, any flight can be abstracted as a sequence of pilot decisions and evolution according to the laws of motion.

In order to generate a safe and non-restrictive monitor in equation (3), it is critical to effectively reason about the behavior of the solutions of the differential equations. Although a closed form solution of equation (4) is known, handling it symbolically would require automated procedures to reason about transcendental functions, a problem which is undecidable in general. Instead, we proceed with a qualitative analysis of the solutions by means of differential invariants [25]—functions that remain constant when evaluated along any solution of the differential equations. In [26], it is established that such invariants can be characterized exactly using higher-order Lie derivatives allowing for an automated and efficient generation procedure based on symbolic linear algebra.* This characterization is used to generate a set of constraints on the coefficients of a parametric polynomial (template) of a fixed degree. If a solution is found, we have an invariant. Otherwise, the degree is augmented and we look for a solution to the new set of constraints.

When $(\omega_1, \omega_2) = (0, 0)$, the direction $\theta(t)$ of the intruder relative to the ownship stays constant over time, that is $\theta(t) = \theta(0)$ for all t . In this case using the technique described in [26], we automatically generate the following non-trivial invariant function:

$$I_1(t) \stackrel{\text{def}}{=} (v_2 \sin \theta(0))x(t) - (v_2 \cos \theta(0) - v_1)y(t) = I_1(0) . \quad (5)$$

The quantity $I_1(t)$ is an invariant because its time derivative $\dot{I}_1(t)$ is zero for all t . The invariant function describes the relative motion of the intruder as a line in the phase space (x, y) . This line is entirely defined by the initial configuration $x(0), y(0), \theta(0)$.

On the other hand, when $(\omega_1, \omega_2) \neq (0, 0)$, we automatically generate the following, non-trivial invariant function:

$$I_2(t) \stackrel{\text{def}}{=} -\omega_1 \omega_2 (x(t)^2 + y(t)^2) + 2v_2 \omega_1 \sin \theta(t)x(t) + 2(v_1 \omega_2 - v_2 \omega_1 \cos \theta(t))y(t) + 2v_1 v_2 \cos \theta(t) = I_2(0) . \quad (6)$$

Again, $I_2(t)$ is an invariant because its time derivative $\dot{I}_2(t)$ is zero for all t . In this case, the relative motion draws a non-trivial helical trajectory (the radius of the circle changes over time) in the 3-dimensional phase space (x, y, θ) .

* The set of differential equations (4) can be rewritten into a set of algebraic differential equations—up to introducing two additional variables to encode the transcendental functions $\cos \theta$ and $\sin \theta$ à la differential axiomatization [25].

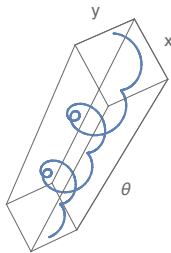


Fig. 5 Helical trajectory in the 3-dimensional phase space (x, y, θ) .

From the invariants $I_1(t)$ and $I_2(t)$, we now sketch how we derive a sufficient condition on the initial conditions, as well as the input commands, in order to avoid a collision. We illustrate the case where $(\omega_1, \omega_2) = (0, 0)$ in detail and then summarize the result for the case $(\omega_1, \omega_2) \neq (0, 0)$, for which an analogous derivation works.

We want the distance $r(t)$ between the two aircraft ($r(t)^2 = x(t)^2 + y(t)^2$) to always remain greater than a given safety bound $p > 0$ at all times:

$$\text{**safety**} \equiv \forall t \geq 0, r(t) > p . \quad (7)$$

Assuming $(\omega_1, \omega_2) = (0, 0)$, from equation (5), $(x(t), y(t))$ moves along a line encoded by $I_1(t) = I_1(0)$, where the constant $I_1(0) \in \mathbb{R}$ is entirely defined by the initial configuration $(x(0), y(0), \theta(0))$. We use this automatically generated invariant to manually derive a necessary condition on the initial configuration when a collision occurs anytime in the future. From there, we derive a sufficient condition—on the initial configuration—that preserves the separation property.

$$|I_1(0)| = |I_1(t)| \quad (8)$$

$$= |(v_2 \sin \theta(0))x(t) - (v_2 \cos \theta(0) - v_1)y(t)| \quad (9)$$

$$\leq v_1|y(t)| + v_2r(t) \left| \frac{x(t)}{r(t)} \sin \theta(0) - \frac{y(t)}{r(t)} \cos \theta(0) \right| \quad (10)$$

$$\leq v_1r(t) + v_2r(t) . \quad (11)$$

In equation (10), we used the fact that $\left| \frac{x(t)}{r(t)} \sin \theta(0) - \frac{y(t)}{r(t)} \cos \theta(0) \right|$ is equal to $|\sin(\theta(0) - \alpha(t))|$, where the angle $\alpha(t)$ is such that $\cos \alpha(t) = \frac{x(t)}{r(t)}$ and $\sin \alpha(t) = \frac{y(t)}{r(t)}$. If a collision occurs, then, there exists $t > 0$ such that $r(t) \leq p$. Therefore, from equation (11), we obtain:

$$\neg \text{**safety**} \longrightarrow |I_1(0)| \leq (v_1 + v_2)p, \quad (12)$$

where (\longrightarrow) denotes the logical implication. By contraposition, this leads to the following sufficient condition on the initial configurations to ensure the safety (when $(\omega_1, \omega_2) = (0, 0)$):

$$|I_1(0)| > (v_1 + v_2)p \longrightarrow \text{**safety**} \quad (13)$$

Figure 6 depicts this safe region for $v_1 = 1$, $v_2 = 2$, $\omega_1 = \omega_2 = 0$ and $p = 1$. For a given initial (relative) configuration $(x(0), y(0), \theta(0))$, if the line $I_1(t) = I_1(0)$ of the (x, y) plane is included in the gray region then the motion is safe as long as (ω_1, ω_2) remains equal to $(0, 0)$. The disk of radius p around the ownship depicts the safe separation distance that we want to keep between the ownship and any intruder at all times. When $(x(0), y(0))$ is in the white region, the motion cannot be proven safe using the invariant $I_1(t) = I_1(0)$. This does not necessarily mean that the motion is unsafe: we may observe *false alarms* when the motion is safe but cannot be proven. The use of invariants instead of the actual solutions of the differential equation as well as non-equivalent inequality bounds as in equation (11) are two common sources for false alarms. Hence, reducing false alarms requires more refined analysis where one targets the sources of large over-approximations.

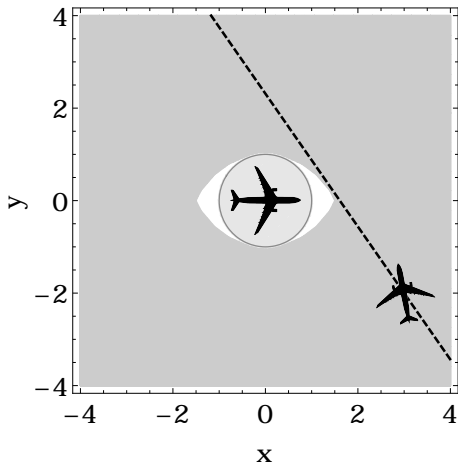


Fig. 6 Safe region (gray) relative to the ownship, centered in the disk of radius p , when $(\omega_1, \omega_2) = (0, 0)$. Given an initial relative configuration $(x(0), y(0), \theta(0))$, if the line described by $I_1(t) = I_1(0)$ lies in the safe region (for instance the black dashed line) then the relative motion is proven safe. If the line intersects the white region, our analysis fails to prove safety.

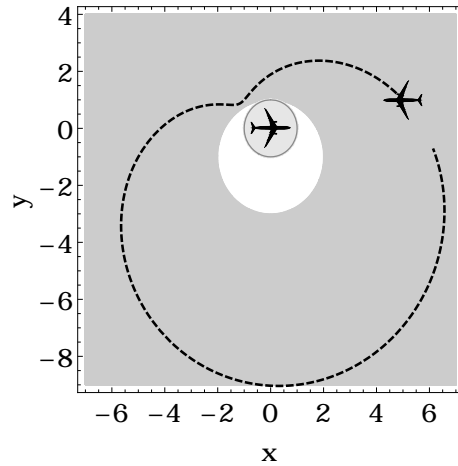


Fig. 7 Projection of the safe region (gray) relative to the ownship, centered in the disk of radius p , when $(\omega_1, \omega_2) \neq (0, 0)$. Given an initial relative configuration $(x(0), y(0), \theta(0))$, if the projection of the curve described by $I_2(t) = I_2(0)$ lies in the safe region then the relative motion is proven safe. If the projection intersects the white region, our analysis fails to prove safety. The black dashed curve depicts a concrete safe trajectory.

Following similar reasoning, when $(\omega_1, \omega_2) \neq (0, 0)$, we derive the following sufficient condition:

$$|I_2(0)| > 2v_1v_2 + 2p(v_2|\omega_1| + v_1|\omega_2|) + p^2|\omega_1\omega_2| \longrightarrow \text{safety} \quad (14)$$

In this case, the safe region is a 3-dimensional region in the configuration space (x, y, θ) described by (14). The projection of the safe region on the (x, y) plane is depicted in gray in Figure 7. Given an initial relative configuration $(x(0), y(0), \theta(0))$, if the projection of the curve described by $I_2(t) = I_2(0)$ lies in the safe region then the relative motion is proven safe. If, however, the projection of the curve intersects the white region, our analysis fails to prove safety. The black dashed curve depicts a safe trajectory in relative coordinates for $v_1 = v_2 = 2$, $\omega_1 = -1$, $\omega_2 = 1$ and $p = 1$.

The monitor of our collision avoidance system (equation (3)) combines the safety bounds given in equations (13) and (14) for the cases of zero and nonzero angular velocity, respectively:

$$\begin{aligned} \text{monitor} \equiv (\omega_1, \omega_2) = (0, 0) &\longrightarrow I_1(0) > p(v_1 + v_2) \wedge \\ (\omega_1, \omega_2) \neq (0, 0) &\longrightarrow I_2(0) > 2v_1v_2 + 2p(v_2|\omega_1| + v_1|\omega_2|) + p^2|\omega_1\omega_2|. \end{aligned} \quad (15)$$

C. Formal Safety Verification

In this section we use the monitor generated in the previous section to formally verify our collision avoidance system.

Theorem 1 (The collision avoidance system *cas* meets its safety requirement). *The following dL formula, expressing that the safety property always holds for the whole system, is valid, i.e., true in all states:*

$$\text{safety} \rightarrow [\text{cas}] \text{safety}, \quad (16)$$

where *cas* is given in (3) and *safety* is defined in (7).

Theorem 1 states that if two airplanes start at a safe state (where the separation is respected), and if the monitor defined in equation (15) is satisfied, then the evolution of the system is safe, hence

collision-free, for all possible executions of the system. The proof obtained with KeYmaera[†] together with Theorem 1 can therefore be used to certify that our model meets its safety requirement and ensures the separation of the equipped ownship from an intruder. The proof tree has 21 branches and 240 nodes. With a highly interactive proof strategy—200 interactive steps—the computation time to discharge the proof obligation takes less than 60s on a standard desktop (4GB of RAM and Intel Core 2 Duo 2.66GHz).

In addition to proving safety, Theorem 1 can be used to determine which control choices for the ownship will remain safe. For given linear velocities v_1, v_2 , intruder’s angular velocity ω_2 , and desired separation p , the set of safe control inputs ω_1 for the ownship can be computed as those that satisfy the monitor formula (15). For a concrete scenario— $v_1 = v_2 = 2, \omega_2 = 1, p = 1$ —in which the intruder is flying on a circular path, Figure 8 illustrates the beginnings of the unsafe trajectories resulting from the set of all ω_1 control choices that are deemed unsafe, because they do not satisfy the monitor (15). The dashed trajectory shows a trajectory for the ownship, $\omega_1 = -1$, that is provably safe, because $\omega_1 = -1$ satisfies the monitor (in particular it is entirely outside the potentially unsafe gray region).

When the ownship detects multiple intruders in its surroundings, the same constraints (equation (15)) can be used for each intruder to ensure separation from all the intruders by suitable coordination. Figure 9 shows a configuration with 3 aircraft where the gray zone depicts the projection of the trajectories resulting from an unsafe command ω_1 being selected. Observe how the gray zone evolved from Figure 8 where only one intruder was involved. It is now a union of all unsafe gray zones for all intruders. The choices that are proven safe for the ownship are considerably reduced. The selected dashed trajectory permits once again to evolve safely.

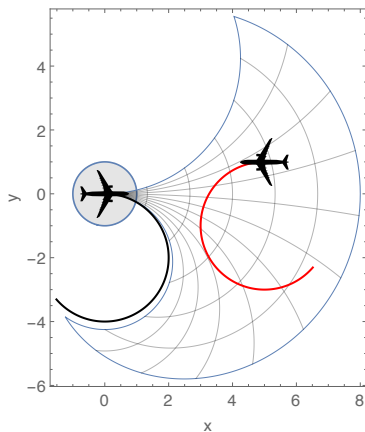


Fig. 8 The gray zone depicts the projection of the trajectories that results when an unsafe command ω_1 is selected. The dashed trajectory shows a proven safe choice for the ownship.

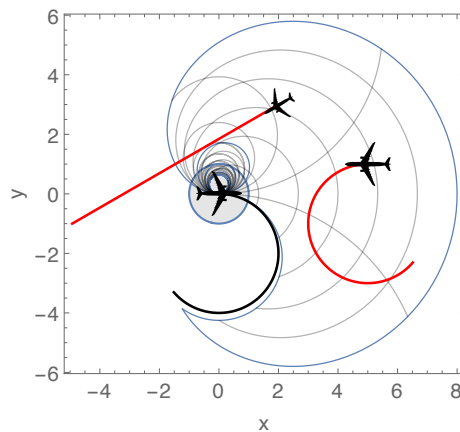


Fig. 9 The gray zone as in Fig 8 accounts now for the additional intruder. The dashed trajectory shows a proven safe choice for the ownship.

Figure 9 suggests that there exist scenarios where the ownship is “trapped”, that is no proven safe choice exists for the ownship. In such cases, and if all involved aircraft are equipped, the system can try to find an instantiation of commands $(\omega_1, \omega_2, \dots)$ (within their respective physical limits) for which the pairwise safety condition (equation (15)) of each two players is satisfied.[‡] When no such safe global collision-free choice exists, however, the system reaches its operational limit: it can no longer give a safe advisory to resolve the conflict.

[†] using an encoding of the transcendental functions (sine and cosine) with extra variables. The models and proof files are available <http://www.lix.polytechnique.fr/~ghorbal/aerospace>.

[‡] Notice that such an approach assumes that involved aircraft can communicate and that pilots are compliant with no delays.

IV. Aerospace Challenges

Aircraft and other physical systems do not always operate in ideal conditions. Physical reality is a noisy place, and aircraft interacting with reality must handle the resulting uncertainty. There are many examples of situations where understanding uncertainty is critical to faithfully rendering an aircraft, so it is important for representations and reasoning systems for aircraft to handle uncertainty concisely, simply, and accurately. In this Section, we first discuss why uncertainty poses a challenge for our approach to verification, how we are currently handling these issues, and how we expect to finally resolve them. We also classify the current and future challenges in the aerospace domain. We distinguish four different categories: uncertainty (Section IV A), proof automation (Section IV B), numerical issues (Section IV C) and finally scalability (Section IV D). Although not exhaustive, our classification is driven by the difficulties we have encountered analyzing real aerospace systems. The classification is also guided by the progress made in general by the formal verification community.

A. Uncertainty

1. Uncertain parameters and component faults

A common kind of uncertainty is with respect to conditions in which a plane is expected to operate. A plane is designed to be able to operate in a variety of atmospheric conditions (air speed, turbulence, and so forth). The same control signals in an aircraft may have different outcomes depending on the exact same atmospheric conditions, and this uncertainty must be accounted for. Our best choice to fully model such behavior of the plane may be to represent it as a distribution over outcomes fitted from data using suitable models.[§]

For example, wind is an uncertain force that affects the velocity of a plane. A collision avoidance advisory system should be robust to wind conditions that are common enough—any maneuver that requires more accurate position control than is possible under reasonable wind conditions should be rejected. It is difficult to represent probability distributions in hybrid systems,[¶] and thus our hybrid systems verification models would handle this type of parameter uncertainty by modeling windspeed as a variable that nondeterministically takes a value in some region. This region should be selected such that it encompasses a range of ‘reasonable wind conditions’.

Selecting this region is currently a matter of human judgment. If an uncertainty outcome is judged sufficiently common then we will conservatively allow this outcome to occur nondeterministically. This assumes that any uncertain parameter’s value is chosen by a hypothetical ‘adversary’ who is trying to derail our attempts to prove system safety—the adversary is some subspecies of Cartesian demon.** Ceding too much control to our adversary makes proving results either difficult or impossible. If, in our models, we hand total control over all uncertainty to the adversary, we will be unable to prove that planes could have left the ground in the first place, much less successfully avoid collisions midair.

One must therefore ignore (or abstract) sufficiently unlikely and unimportant events. We cannot model every possible event from deep in the tail of a probability distribution—we do not want to model glancing blows from meteors, which would make the aircraft system unsafe but are beyond our control anyhow. We must also omit events in a principled fashion or our model will be too optimistic in a way that will be difficult to quantify.

For the purposes of verification we want a global bound Δ on the total joint probability of the events we ignore. Since our proof is based on ignoring these events, this bound licenses us to certify that our proof applies with probability $1 - \Delta$. To simplify our proving task we currently decompose this global probability bound into a series of bounds on the range of parameters. This

[§] However, using a joint distribution that is far from the true uncertainty in, for example, Kullback-Leibler divergence may lead to subtle errors in the overall model.

[¶] Stochastic information can be incorporated into hybrid system models leading to stochastic hybrid systems, for which logic can be lifted [27]. Verification of stochastic hybrid systems is quite nontrivial, though, so that we pursue a separate approach here.

** This adversary is just a way of explaining a worst-case analysis; we are not describing a true game theoretic setting here.

decomposition ensures that we do not have to do complicated probability calculations every time we make a nondeterministic choice in our proof.

The exact form of a parameter bound is something that the prover could pick during the proving process—the bounds do not have to be set *a priori*. A particular parameter’s bounds could themselves be a parametric shape (corresponding to, e.g., a fixed distribution’s confidence intervals). The prover is free to pick these shapes as long as the joint probability of all of these shapes exceeds the global probability bound $1 - \Delta$. This gives our prover more flexibility than if we were to *a priori* set the bounds, and this additional freedom might be important for proving safety.

Estimating Δ —the joint probability bound—is essentially an integration question. At a high level: if Ω is a set of all possible joint parameter settings (e.g. $\vec{x} \in \mathbb{R}^n$), $\mathcal{E} \subset \Omega$ is the set of all joint settings that satisfy the local bounds (e.g. a box $\mathcal{E} = \{\vec{x} \in \mathbb{R}^n \mid l_i \leq x_i \leq u_i\}$), and p is a joint probability density function (e.g. a multivariate normal distribution), then we want to find a small Δ such that $\Delta \geq 1 - \int_{\mathcal{E}} p(\vec{x}) d\vec{x}$. Computing this integral exactly is—in general—not possible. The challenge is to find local bounds that allow us to easily prove safety while simultaneously permitting tight approximations of this integral. There may be a natural trade off, in many problems, between ease of proof and the strength of the probability bound.

2. Sensor uncertainty

Another source of uncertainty is an aircraft’s estimates about its state. In the collision avoidance example, the state of both planes (e.g. their positions and their velocities) were used in the model in Section III, but are really estimated from a number of different noisy sensors, including radar and GPS positioning information broadcast by ADS-B. Planes need to form an accurate understanding of the world with noisy sensors and do this by fusing the measurements from different sensors over time in order to filter out noise. This process of filtering noise—updating state estimates based on noisy information—poses several interesting challenges for verification.

First of all, uncertain models are larger and more difficult to reason about than certain models of the same system. This is one of the reasons why it is beneficial to first start a formal analysis under ideal-world conditions and then proceed to include uncertainties, because if the system is already unsafe under ideal-world sensing, it is not safe to be used in the real world. In order to take sensor uncertainty into account, we need to represent both the state of the plane and the plane’s estimates about its own state. This is necessary because the plane’s control systems (such as the pilot, or any automated system) can only react to its estimates, but their actions will take effect on the true state. The plane’s beliefs about position are a ‘first-class citizen’ of the proof that need to be modeled in the underlying logic.

Secondly, we need to reason about filtered estimates in a safe way. For example, we currently bound the beliefs of the entities in our verification of a robot collision avoidance problem [20] by some static confidence region centered at the true location of robot; the width of this region is based on sensor quality. The robot’s estimate of its position corresponds to the immediate sensor information. The robot never compares this measurement to any past measurements—it does not filter at all.

There are alternative ways to update estimates that consider past observations. A Kalman filter, for example, is the optimal way to estimate state for linear dynamical systems with Gaussian noise [28]. Updating estimates in this way tends to be more accurate than enthusiastically discarding past observations. This extra precision allows collision avoidance policies that are less conservative about uncertainty and are more efficient because they require smaller margins, for example the distance between planes passing each other at different altitudes. Smaller safety margins means fewer ‘false alarm’ advisories which waste fuel and may fray the pilot’s nerves, undermining the credibility of the system.

Unfortunately, if we hand over control of our sensors to our proof adversary, the adversary can maliciously pick noise that is highly correlated or anti-correlated. This could cause filtering algorithms to form very inaccurate state estimates—a filter was designed for a distribution of sensor noise, and our adversary can break the filter by picking a sequence of sensor measurements that is entirely unrepresentative of the assumed noise model. This is possible even if we restrict the adversary to measurements that are individually within some local $1 - \delta$ confidence region.

Therefore either we need to recommend that planes use inefficient policies that, like in [20], do

not filter, or further restrict the sequences that a proof’s adversary can employ. We want to restrict the adversary to sequences that never have subsequences that are too unlikely, for example asserting that all measurement subsequences of length k have some minimum probability. How this should be done efficiently in verification is an interesting open problem.

3. *Controllers that Deal with Uncertainty: ACAS X Case Study*

Controllers designed for uncertain settings are difficult to formally reason about. The controller synthesis problem is—by itself—difficult. Due to this difficulty, we might be forced to use synthesis methods that utilize potentially unsafe heuristics (although, reasonable heuristics should be safe with high probability). Since the resulting controller may not be safe, we need a separate, sound verification procedure.

Control policies may also be tough to represent compactly. We have been grappling with this representation issue on the Aircraft Collision Avoidance System (ACAS X) [29]. This system, a successor to TCAS, suggests pilot actions (from a small number of discrete advisories) whenever two or more planes are close enough to each other in midair to risk a near mid-air collision (NMAC), i.e., an unacceptable proximity value.

The control policy for this system (*i.e.* a function that maps from the system state to advisories) is found by discretizing the joint state space of a two plane encounter. The exact discretization scheme is subject to change, but one can imagine a continuous dimension of state like altitude being represented by an integer multiple of meters in some reasonable range of common altitudes. The system’s dynamics are encoded in this discrete state-space as probabilistic transitions between discrete states (this abstracts away underlying details of the physics, which the hybrid systems techniques handle). Undesirable behaviors and states are described by assigning a cost to certain state-action pairs. These costs are based on features like whether the state is an NMAC, and whether the action reverses the previous resolution advisory—repeatedly reversing suggestions is jarring to the pilot.

Modeling the continuous dynamics discretely in this way yields a *Markov decision process* (MDP); a common decision theory framework with several effective algorithms for finding optimal policies for an MDP (see, for example, [30]). MDPs are solved by dynamic programming algorithms that propagate the costs of states and actions to the states and actions that lead to these states. Discrete optimal policies found with these dynamic programming algorithms form the basis of the continuous state-space ACAS X controller. Every continuous point falls in the neighborhood of a number of discrete points, and the MDP solutions at these neighboring discrete points are linearly interpolated to find the cost of different actions at the continuous point.

The MDP controller is synthesized using a discretized model of our dynamics, but we want to prove its safety using more faithful continuous dynamics.^{††} In principle we could explicitly represent the controller in our logic and try to directly prove its safety. However, even in a compact representation the MDP policy is hundreds of megabytes large—monstrously large for all existing hybrid systems verification tools.

Instead, we suggest a two-stage method for proving safety with respect to the continuous dynamics. First, the theorem prover is employed to find sufficient conditions for safety in the hybrid systems model with its differential equations describing the continuous dynamics. These conditions take the form of *safe regions* in the continuous state space for each action. Boundaries are described with low-order polynomials. The regions have the semantics that if you start in S_a —the safe region for action a —then you will remain safe as long as you execute a . These safe regions are related to the **monitors** discussed in Section III. If one is in the intersection of two safe regions (*i.e.* in $S_a \cap S_b$) then there are at least two safe actions to choose from. The system has a problem if it is in no safe region S_a .

With these safety regions, we can check that the possible chains of actions recommended by a policy are all safe. In the MDP policy for ACAS X, this is a simple geometric problem that can be

^{††} In fact, the policy might not even be safe for the discrete dynamics. This is due to the fact that operational considerations like ‘not bothering the pilot’ are incorporated into the cost function.

done in parallel. We do this by partitioning the continuous state space and checking each cell. If action a is recommended anywhere in cell C , then that resolution advisory is safe if C is contained in a 's safety region: $C \subseteq S_a$. If a partition contains possibly unsafe cells, then we either refine the partition in these cells, or give up and report the problematic cells. If every cell is safe, then we conclude inductively that the policy is safe: we only jump from safe action to safe action.

This two-stage method offers two advantages over the more direct method of explicitly representing the policy. First, the safety regions are more compact symbolic regions than the MDP policy^{‡‡} and easier for the theorem prover to reason about. This is because safety regions abstract away a lot of the details the MDPs had to wrestle with, and use a symbolic polynomial representation that is richer than the tabular representation used by the MDP. Secondly, finding the safe regions is decoupled from checking that a specific policy actually is safe. So if the exact details of the policy are altered, we can simply reuse our safe regions and focus on checking that the new policy is still safe according to these regions without having to redo the hybrid systems safety proofs.

This approach clearly complements the use of probabilistic model checking to verify the ACAS X policy table [31].

B. Proof Automation

Automation for formal verification and validation is a key factor to integrating its use in industrial contexts. This is even more crucial when one wants to use such tools in a design loop in early development stages. Although nowadays engineers are more and more familiar with the theoretical background behind these approaches, full automation is an active area of research to expand the reach of formal verification & validation. In addition to the usual curse of dimensionality, expressive languages—although very useful in practice—also have their challenges in analysis as they allow the design of heterogeneous aspects of the systems. The typical example in hybrid systems modeling languages is the ability to mix continuous (dynamics) and discrete (control) evolution of time, which is crucial to understanding the interactions of advanced control with dynamics.

We are currently working on new ideas to reduce the needed number of interactions with the theorem prover. More precisely, in addition to the automated generation of differential invariants (such as I_1 and I_2 in Section III), we want to automate the generation of the safety bounds (such as the safety monitor in Section III). Such directions have been recently explored within our group [32].

C. Numerical Issues

A computer cannot effectively perform real numbers computations. Real numbers are always truncated to fit a specific finite representation, more suitable for machines. Floating and fixed point representations are the two commonly used finite representations. The challenge is that the models and flight control algorithms are usually designed assuming real numbers arithmetic. Real numbers are accurate for modeling the continuous behavior in the system, including the evolution of positions and orientations of the aircraft. However, the internal control implementation will ultimately work only with finite-precision arithmetic, which deviates from the true real state of the system and might lead to errors that accumulate over time. Therefore, the discrepancy introduced by approximating real numbers by finite representations needs to be accounted for. KeYmaera does not have a built-in way of addressing floating-point or fixed-point arithmetic.

There are formal verification tools, such as Fluctuat [33], that are designed to formally check the robustness and stability of numerical algorithms, that is, determine bounds on the error introduced by floating-point computations. But those tools cannot reason about the differential equations describing the aircraft dynamics nor about the true state of the aircraft in terms of its real position, velocity, and orientation. Tools like Fluctuat, thus, often lead to rather pessimistic

^{‡‡} The safety-region polynomials were much more compact than any representations we tried for either the optimal value-function or the policy regions for the ACAS X MDP. This included both lossless representations that tried to exploit regularity in the data, and representations that allowed a small amount of loss.

over-approximations, because they lose the functional relations between the actuators (output) and the sensors' data (input).

Consequently, theorem proving of hybrid systems should be leveraged to model and reason about the continuous part of the system and help tools like Fluctuat to tighten their approximations and increase the accuracy of their results. Conversely, KeYmaera could use the bounds and numerical relations discovered by Fluctuat as an overapproximation for the error introduced by the finite precision number arithmetic.

D. Scalability

As stated earlier in this paper, formal verification tools do not and should not claim to solve all verification problems of all aspects of all systems. Nevertheless, even after abstracting away unwanted details, the scalability of the formal verification tools still remains one of the biggest challenges for the community. Many tailored solutions have been successfully tested for abstract interpretation [6, 34], each tuned for one specific property. The state space explosion in model checking is yet an active area of research since its introduction in the early 80's (see [5] for a survey).^{§§} The $d\mathcal{L}$ framework has already pushed further the boundaries that previous approaches were limited to [9, 12]. It allows for sound compositional verification of (linear and nonlinear) hybrid systems. That is, the system is decomposed into subsystems, then each subsystem is analyzed independently from the other parts. This approach allows for the verification of real-world systems as it focuses on one smaller part at a time. Our current work focuses on pushing further the automation of the decomposition and abstraction procedures.

V. Related Work

The freshly published DO-178C [36] report about the certification guide lines for commercial software-based aerospace systems explicitly supports the use of formal methods as a new standard to ensure high-quality software for critical systems. Both software model checking [37] and abstract interpretation [6, 38] have been shown to help verifying the complex software of nowadays aircraft.

Nevertheless having an aircraft in which software runs as expected does not necessarily prevent collisions from happening. Indeed, such analysis needs to consider the behavior of the aircraft—in addition to its embedded software—within its operational environment. This property makes aircrafts essentially hybrid, because they involve both continuous flight dynamics and discrete control software, which is a combination that is out of scope for software model checking [37] and abstract interpretation [6, 38]. Formal verification and validation techniques for hybrid systems such as theorem proving [8, 9, 12] are suitable, as we have illustrated in the collision avoidance case study of Section III.

The benefit of formally verifying a collision avoidance system is obvious: missing corner cases is unacceptable. While exhaustive stress testing is infeasible, formal methods aim at proving the absence of mishaps, a highly desirable property. As a matter of fact, many collision avoidance systems have been proposed—see [39] for an overview—, but only few attempts have been made to formally verify such systems [24, 40–42]. Non-verified collision avoidance systems, such as those based on probabilistic modeling methods (e.g. [43, 44]) have the advantage of being designed to cope with the noisy physical environment. However, this leads to discrete time and space approximations of the motion of involved aircraft, leaving open questions about the error that these approximations induce on the overall behavior of the system. On the other hand, although easier to verify, worst-case modeling (e.g. [45, 46]) is too conservative. It assumes uniform distributions of events and resorts often to either large or non-flyable approximations of the dynamics. Likewise, the verification (not the design) of collision avoidance systems based on controllable sets [24, 47] uses either simplified, usually non-flyable approximations, such as a succession of straight-lines segments or instant turns as in [24] or falls back on non-exhaustive numerical testing when the computation is too complex

^{§§} Similar issues were noticed in other branches of computer science earlier, e.g. the *curse of dimensionality* observed by Bellman [35].

[47]. Numerical computation of the reachable sets of hybrid systems by means of partial differential equations [48] has been applied to design and verify the safety of collision avoidance systems for which an analytic solution is available. Such approach is interesting, especially in sufficiently low dimensions, but is not necessarily sound, because of numerical errors that can accumulate. Notice also that the computation of reachable sets is hard to automate except for hybrid systems with linear dynamics [49].

Previous approaches of using theorem provers, such as PVS, in [40, 41], for collision avoidance systems were helpful but also ignored the flight dynamics and used more abstract geometrical considerations such as straight lines. Once again, the main problem one faces in this case is the lack of any formal link between the aircraft model and its geometrical approximation. In addition, using a hybrid theorem prover such as KeYmaera, enriched with specific proof rules to reason about the continuous dynamics, is much more suitable for systems that are essentially hybrid.

The hybrid system theorem prover KeYmaera has been used to formally verify a circular round-about maneuver modeled by Dubins curves [42]. The full capabilities of such protocol proposed first in [24] were formally verified and even refined with flyable entries and exits.

The proposed approach in this work is different from [24, 42] in that it is not protocol-based. It hence allows for more flexible, yet flyable, collision resolutions. The verification does not consider an approximated continuous nor a discretized trajectory as in [24, 40, 41]. The analysis is performed with respect to the continuous motion of aircraft. Yet, it allows for the selection of a feasible control input as in [24, 47, 48], leading naturally to a flyable resolution of conflicts. It is however desirable to refine the considered model with relevant uncertainties in the sensing and actuation as in [43–46], which we leave as a future work.

Apart from collision avoidance, other aircraft maneuvers were also formally verified. For instance, the PVS theorem prover was used to prove the safety of an aircraft landing protocol [50, 51]. In [52] the safety analysis of aircraft autolander was performed by means of numerical computation of reachable sets. More recently, the distributed extension of the KeYmaera and its underlying logic was used to verify safe separation in a system with arbitrarily many aircraft or UAVs [19].

VI. Conclusion

The differential dynamic logic ($d\mathcal{L}$) framework gives an expressive and rich language for a formal description of hybrid systems. It comes with effective syntactic proof rules that were proven successful in different case studies, including applications in the domain of aviation. We sketched the avenues we are exploring to extend our approach to formally verify real aerospace systems. We are convinced that the complexity and nature of the aerospace systems requires a mixture of formal methods and classical empirical approaches to verify and validate the behavior of all parts of the system design and implementation. The drawbacks of one method are usually diminished—if not eliminated—by the complementary use of another approach. For instance abstract interpretation techniques can significantly improve the scalability of the analysis, whereas theorem proving turns out to be crucial to reason about high level abstraction that requires precise analysis and the handling of the flight dynamics. In the same vein, model checking can be used to reach a better coverage of the state space even for properties that are still out of reach for abstract interpreters. Similarly, theorem provers provide means of verifying systems that are out of reach for their present level of automation or for other automatic tools, because they provide ways of inspecting and where an automatic proof attempt failed and helping out.

With the astonishing advances in formal methods, and the surprisingly strong analysis that is possible today, we believe that formal methods will become—and remain—a crucial ingredient in the design of aerospace systems. We are convinced that formal methods, as supported by DO-178C, will play an increasingly important role in certification. Runs of a model checker or abstract interpreter give evidence that parts of the aerospace system function correctly, and this evidence should be incorporated in the certification process. In theorem proving approaches—such as the one shown in this paper—certification is direct and emphatic. The result of a theorem proving activity for an aerospace system is a proof, which can be checked independently as evidence that the aerospace system is correct with respect to the corresponding formal properties. This is a promising feature, because it makes the final proof artifact independent of the advanced automation techniques that

were used in the original proof system to come up with the proof in the first place.

- [1] Trimble, S., “AUVSI: RQ-7 likely not to blame for C-130 collision,” *Flight Global*, 2011.
- [2] Johnson, C., “Review of the BFU Überlingen Accident Report,” , 2004. www.dcs.gla.ac.uk/~johnson/Eurocontrol/Ueberlingen/Ueberlingen_Final_Report.PDF.
- [3] “Airbus - Esterel Technologies,” <http://www.esterel-technologies.com/success-stories/airbus/>. Accessed: July 2014.
- [4] Weiss, K. A., Dulac, N., Chiesi, S., Daouk, M., Zipkin, D., and Leveson, N., “Engineering Spacecraft Mission Software using a Model-Based and Safety-Driven Design Methodology,” *Journal of Aerospace Computing, Information, and Communication*, Vol. 3, No. 11, 2006, pp. 562–586, doi:10.2514/1.24677.
- [5] Fix, L., “Fifteen Years of Formal Property Verification in Intel,” in Grumberg, O. and Veith, H., eds., “25 Years of Model Checking,” Springer-Verlag, Berlin, Heidelberg, pp. 139–144, 2008.
- [6] Delmas, D. and Souyris, J., “Astrée: From Research to Industry,” in Nielson, H. and Filé, G., eds., “Static Analysis,” Springer Berlin Heidelberg, Vol. 4634 of *LNCS*, pp. 437–451, 2007.
- [7] Bertrane, J., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., and Rival, X., “Static Analysis and Verification of Aerospace Software by Abstract Interpretation,” in “AIAA Infotech@Aerospace 2010,” American Institute of Aeronautics and Astronautics, Atlanta, Georgia, 2010, pp. 1–38.
- [8] Platzer, A., “Differential Dynamic Logic for Hybrid Systems.” *Journal of Automated Reasoning*, Vol. 41, No. 2, 2008, pp. 143–189.
- [9] Platzer, A., “Logics of Dynamical Systems,” in “Logic in Computer Science (LICS’12),” IEEE, 2012, pp. 13–24.
- [10] Platzer, A. and Clarke, E. M., “Computing Differential Invariants of Hybrid Systems as Fixedpoints,” in Gupta, A. and Malik, S., eds., “Computer Assisted Verification (CAV’08),” Springer, Vol. 5123 of *LNCS*, 2008, pp. 176–189.
- [11] Henzinger, T. A., “The Theory of Hybrid Automata,” in “LICS,” IEEE Computer Society, 1996, pp. 278–292.
- [12] Platzer, A., *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*, Springer, Heidelberg, 2010.
- [13] Platzer, A. and Quesel, J.-D., “KeYmaera: A Hybrid Theorem Prover for Hybrid Systems.” in Armando, A., Baumgartner, P., and Dowek, G., eds., “International Joint Conference on Automated Reasoning (IJCAR’08),” Springer, Vol. 5195 of *LNCS*, 2008, pp. 171–178.
- [14] Loos, S. M., Platzer, A., and Nistor, L., “Adaptive Cruise Control: Hybrid, Distributed, and Now Formally Verified,” in “Formal Methods (FM’09),” Springer, Vol. 6664 of *LNCS*, 2011, pp. 42–56.
- [15] Loos, S. M. and Platzer, A., “Safe Intersections: At the Crossing of Hybrid Systems and Verification,” in Yi, K., ed., “Intelligent Transportation Systems (ITSC’11),” , 2011, pp. 1181–1186.
- [16] Mitsch, S., Loos, S. M., and Platzer, A., “Towards formal verification of freeway traffic control,” in Lu, C., ed., “Proceedings of the 2nd International Conference on Cyber-Physical Systems (ICCP’12),” IEEE, 2012, pp. 171–180.
- [17] Platzer, A. and Quesel, J.-D., “European Train Control System: A Case Study in Formal Verification,” in Breitman, K. and Cavalcanti, A., eds., “International Conference on Formal Engineering Methods (ICFEM’09),” Springer, Vol. 5885 of *LNCS*, 2009, pp. 246–265.
- [18] Platzer, A. and Clarke, E. M., “Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study,” in Cavalcanti, A. and Dams, D., eds., “Formal Methods (FM’09),” Springer, Vol. 5850 of *LNCS*, 2009, pp. 547–562.
- [19] Loos, S. M., Renshaw, D. W., and Platzer, A., “Formal Verification of Distributed Aircraft Controllers,” in Belta, C. and Ivancic, F., eds., “Hybrid Systems: Computation and Control (HSCC’13), Philadelphia, PA, USA, April 8-13, 2013 (part of CPS Week 2013),” ACM, 2013, pp. 125–130.
- [20] Mitsch, S., Ghorbal, K., and Platzer, A., “On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles,” in “Robotics: Science and Systems (RSS’13),” , 2013. <http://www.roboticsproceedings.org/rss09/p14.html>.
- [21] Kouskoulas, Y., Renshaw, D. W., Platzer, A., and Kazanzides, P., “Certifying the Safe Design of a Virtual Fixture Control Algorithm for a Surgical Robot,” in Belta, C. and Ivancic, F., eds., “Hybrid Systems: Computation and Control (HSCC’13), Philadelphia, PA, USA, April 8-13, 2013 (part of CPS Week 2013),” ACM, 2013, pp. 263–272.
- [22] Dubins, L. E., “On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents,” *American Journal of Mathematics*, pp. 497–516.
- [23] Bicchi, A., Marigo, A., Pappas, G., Pardini, M., Parlangei, G., Tomlin, C., and Sastry, S., “Decentralized air traffic management systems: performance and fault tolerance,” in “Proceedings of the IFAC International Workshop on Motion Control (MC’98),” , 1998, pp. 259–264.
- [24] Tomlin, C., Pappas, G. J., and Sastry, S., “Conflict resolution for air traffic management: a study in multi-agent hybrid systems,” *IEEE Transactions on Automatic Control*, Vol. 43, No. 4, 1998, pp. 509–521.
- [25] Platzer, A., “Differential-algebraic Dynamic Logic for Differential-algebraic Programs,” *J. Log. Comput.*, Vol. 20, No. 1, 2010, pp. 309–352. Advance Access published on November 18, 2008.

- [26] Ghorbal, K. and Platzer, A., “Characterizing Algebraic Invariants by Differential Radical Invariants,” in Ábrahám, E. and Havelund, K., eds., “TACAS,” Springer, Vol. 8413, 2014, pp. 279–294.
- [27] Platzer, A., “Stochastic Differential Dynamic Logic for Stochastic Hybrid Programs,” in Bjørner, N. and Sofronie-Stokkermans, V., eds., “Conference on Automated Deduction (CADE’11),” Springer, Vol. 6803 of *LNCS*, 2011, pp. 431–445.
- [28] Kalman, R. E. et al., “A new approach to linear filtering and prediction problems,” *Journal of Basic Engineering*, Vol. 82, No. 1, 1960, pp. 35–45.
- [29] Kochenderfer, M. J., Holland, J. E., and Chryssanthacopoulos, J. P., “Next-Generation Airborne Collision Avoidance System,” *Lincoln Laboratory Journal*, Vol. 19, No. 1.
- [30] Puterman, M. L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, John Wiley & Sons, 2009.
- [31] von Essen, C. and Giannakopoulou, D., “Analyzing the Next Generation Airborne Collision Avoidance System,” in Ábrahám, E. and Havelund, K., eds., “TACAS,” Springer, Vol. 8413 of *Lecture Notes in Computer Science*, 2014, pp. 620–635.
- [32] Mitsch, S. and Platzer, A., “ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models,” in Bonakdarpour, B. and Smolka, S. A., eds., “RV,” Springer, Vol. 8734 of *LNCS*, 2014, pp. 199–214.
- [33] Bouissou, O., Goubault, E., Putot, S., Tekkal, K., and Vedrine, F., “HybridFluctuat: A Static Analyzer of Numerical Programs within a Continuous Environment,” in “Computer Aided Verification (CAV’09),” Springer-Verlag, Berlin, Heidelberg, 2009, pp. 620–626.
- [34] Venet, A., “The Gauge Domain: Scalable Analysis of Linear Inequality Invariants,” in Madhusudan, P. and Seshia, S., eds., “Computer Aided Verification (CAV’12),” Springer Berlin Heidelberg, Vol. 7358 of *LNCS*, pp. 139–154, 2012.
- [35] Bellman, R. E., *Dynamic programming*, Princeton University Press, 1962.
- [36] RTCA, “DO-178C: Software Considerations in Airborne Systems and Equipment Certification,” , 2012.
- [37] Miller, S. P., Whalen, M. W., and Cofer, D. D., “Software model checking takes off,” *Commun. ACM*, Vol. 53, No. 2, 2010, pp. 58–64.
- [38] Souyris, J., Wiels, V., Delmas, D., and Delseny, H., “Formal Verification of Avionics Software Products,” in Cavalcanti, A. and Dams, D., eds., “FM,” Springer, Vol. 5850 of *Lecture Notes in Computer Science*, 2009, pp. 532–546.
- [39] Kuchar, J. K. and Yang, L. C., “A review of conflict detection and resolution modeling methods,” *Intelligent Transportation Systems, IEEE Transactions on*, Vol. 1, No. 4, 2000, pp. 179–189.
- [40] Dowek, G., Muñoz, C., and Carreño, V. A., “Provably Safe Coordinated Strategy for Distributed Conflict Resolution,” in “Proceedings of the AIAA Guidance, Navigation, and Control Conference and Exhibit 2005, AIAA-2005-6047,” , 2005.
- [41] Galdino, A. L., Muñoz, C., and Ayala-Rincón, M., “Formal Verification of an Optimal Air Traffic Conflict Resolution and Recovery Algorithm,” in Leivant, D. and de Queiroz, R., eds., “Workshop on Logic, Language, Information and Computation (WoLLIC’07),” Springer, Vol. 4576 of *LNCS*, 2007, pp. 177–188.
- [42] Platzer, A. and Clarke, E. M., “Computing Differential Invariants of Hybrid Systems as Fixedpoints,” *Formal Methods in System Design*, Vol. 35, No. 1, 2009, pp. 98–120.
- [43] Prandini, M., Hu, J., Lygeros, J., and Sastry, S., “A probabilistic approach to aircraft conflict detection,” *Intelligent Transportation Systems, IEEE Transactions on*, Vol. 1, No. 4, 2000, pp. 199–220.
- [44] Lygeros, J. and Prandini, M., “Aircraft and weather models for probabilistic collision avoidance in air traffic control,” in “Decision and Control, 2002, Proceedings of the 41st IEEE Conference on,” Vol. 3, 2002, pp. 2427–2432 vol.3.
- [45] Lachner, R., “Collision avoidance as a differential game: real-time approximation of optimal strategies using higher derivatives of the value function,” in “Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on,” Vol. 3, 1997, pp. 2308–2313 vol.3.
- [46] Bilimoria, K. D., Lee, H. Q., Mao, Z.-H., and Feron, E., “Comparison of Centralized and Decentralized Conflict Resolution Strategies for Multiple-Aircraft Problems,” in “AIAA Guidance, Navigation, and Control Conference,” , 2000.
- [47] Hwang, I. and Tomlin, C., “Protocol-based conflict resolution for air traffic control,” *Air Traffic Control Quarterly*, Vol. 15, No. 1, 2007, pp. 1–34.
- [48] Mitchell, I. M., Bayen, A. M., and Tomlin, C. J., “A time-dependent Hamilton-Jacobi formulation of reachable sets for continuous dynamic games,” *IEEE Trans. Automat. Contr.*, Vol. 50, No. 7, 2005, pp. 947–957.
- [49] Frehse, G., Guernic, C. L., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., and Maler, O., “SpaceEx: Scalable Verification of Hybrid Systems,” in Gopalakrishnan, G. and Qadeer, S., eds., “CAV,” Springer, Vol. 6806 of *Lecture Notes in Computer Science*, 2011, pp. 379–395.
- [50] Umeno, S. and Lynch, N. A., “Proving Safety Properties of an Aircraft Landing Protocol Using I/O Automata and the PVS Theorem Prover: A Case Study,” in Misra, J., Nipkow, T., and Sekerinski, E., eds., “Formal Methods (FM’06),” Springer, Vol. 4085 of *Lecture Notes in Computer Science*, 2006, pp. 64–80.

- [51] Umeno, S. and Lynch, N. A., “Safety Verification of an Aircraft Landing Protocol: A Refinement Approach,” in Bemporad, A., Bicchi, A., and Buttazzo, G., eds., “Hybrid Systems: Computation and Control (HSCC’07),” Springer, Vol. 4416 of *LNCS*, 2007, pp. 557–572.
- [52] Bayen, A. M., Mitchell, I. M., Osihi, M. K., and Tomlin, C. J., “Aircraft Autolander Safety Analysis Through Optimal Control-Based Reach Set Computation,” *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 1, 2007, pp. 68–77.