JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 11, No. 10, October 2014

Hybrid Theorem Proving of Aerospace Systems: Applications and Challenges

Khalil Ghorbal,* Jean-Baptiste Jeannin,† Erik Zawadzki,‡ André Platzer,§
Geoffrey J. Gordon,¶ and Peter Capell**
Carnegie Mellon University, Pittsburgh, Pennsylvania 15213

DOI: 10.2514/1.I010178

Complex software systems are becoming increasingly prevalent in aerospace applications: in particular, to accomplish critical tasks. Ensuring the safety of these systems is crucial, as they can have subtly different behaviors under slight variations in operating conditions. This paper advocates 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, a novel lateral midair collision-avoidance maneuver is studied in an ideal setting, without accounting for the uncertainties of the physical reality. The challenges that naturally arise when applying such technology to industrial-scale applications is then detailed, and proposals are given 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] and the 2002 tragic midflight 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 Industries, are now relying heavily on model-based architecture, model-based design, and certified code generation.†† It can be argued that an analysis using models is currently the only approach to ensuring a safe development of highly reliable, complex systems [3]. There are numerous immediate benefits to this approach, such as model reusability, compositional 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 [4] 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 [5,6]. 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 it could easily miss some corner cases. Exhaustive testing is limited to relatively small systems (even a system with seven 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* [7,8], which has been applied to many difficult problems where all other (formal and classical) methods have failed [9]. A *hybrid system* [10] 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

Received 17 September 2013; revision received 14 June 2014; accepted for publication 28 July 2014; published online 1 October 2014.

*Postdoctoral Fellow, Department of Computer Science, 5000 Forbes Avenue, Gates Hillman Center; kghorbal@cs.cmu.edu.
†Postdoctoral Fellow, Department of Computer Science, 5000 Forbes Avenue, Gates Hillman Center; jeannin@cs.cmu.edu.
‡Ph.D. Student, Department of Computer Science, 5000 Forbes Avenue, Gates Hillman Center; epz@cs.cmu.edu.
§Associate Professor, Department of Computer Science, 5000 Forbes Avenue, Gates Hillman Center; aplatzer@cs.cmu.edu.
¶Associate Professor, Department of Machine Learning, 5000 Forbes Avenue, Gates Hillman Center; ggordon@cs.cmu.edu.
**Senior Member of Engineering Staff and Adjunct Professor, Software Engineering Institute, Department of Computer Science, 4500 Fifth Avenue; psc@sei.cmu.edu.

††"Airbus — Esterel Technologies," http://www.esterel-technologies.com/success-stories/airbus/ [retrieved July 2014].

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 Sec. II, we recall differential dynamic logic, which is 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* midair collision-avoidance maneuver in Sec. 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 Sec. 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* [7,8,11], which is a logic for specifying and verifying properties of hybrid systems. The hybrid systems theorem prover KeYmaera [12] implements those and other logics in an automated and interactive theorem prover.

A. Differential Dynamic Logic

Differential dynamic logic [7,8,11] is the logic of hybrid systems, i.e., systems with interacting discrete and continuous dynamics. This logic can model the system (e.g., the motion of aircraft, cars, or trains), as well as express properties about such systems. Because differential dynamic logic comes with a proof calculus [7,8,11], it is both a specification and a verification language for hybrid systems.

We model systems being studied with *hybrid programs*, which is 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 they 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 that have parts not controlled by the program. A *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 of other aircraft suddenly climbing or descending. 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 the hybrid programs have been defined, properties about the system can be expressed through the 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), 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 paper but can be found in the literature [7,8,11]. 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. KeYmaera Theorem Prover for Hybrid Systems

KeYmaera [12] is a hybrid verification tool for hybrid systems implementing differential dynamic logic [7] 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 [13], left-turn assist controllers for cars at intersections [14], intelligent speed adaptation for variable speed limit control and incident management by traffic centers on highways [15], cooperation protocols of the European Train Control System [16], airplane collision avoidance [17,18], obstacle avoidance for ground robots [19], and force feedback to the surgeon from a surgical robot [20].

In Sec. 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 [17].

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, whereas 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 Sec. IV.

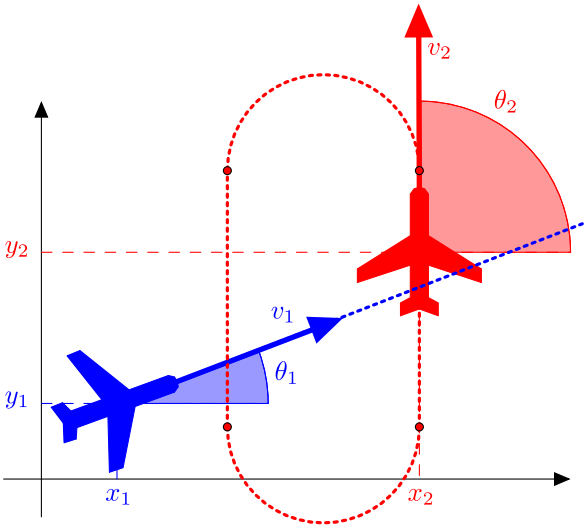

Fig. 1 Planar motion of two airplanes with respect to an inertial frame of reference. Dotted lines show an example of flyable trajectories that the model can describe.

Section III.A recalls the differential equations that describe Dubins trajectories in absolute and relative frames. In Sec. III.B, we abstract the trajectories using automatically generated functional invariants (first integrals), i.e., nontrivial 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 Sec. 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 [21–23].

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 disk. 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 (Fig. 1). The motion of an aircraft i, $i = 1, 2$, with respect to an inertial frame is described by the system of ordinary differential equations given in Eq. (1):

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

The time variable t is implicit: its time derivative is equal to one 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 Fig. 1, the dotted lines show the projected trajectories of two aircraft with potentially conflicting paths: aircraft 1 is flying in a straight line, whereas 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 that lead to additional constraints on ω_i and v_i.

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

$$\texttt{pilot} \equiv \omega_1 := * \tag{2}$$

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

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 Eq. (2)]. The expression "`evolve`" in the hybrid program `cas` encodes the set of differential equations describing the evolution of the intruder relative to the ownship. We use the relative motion of the intruder with respect to the ownship by representing the motion equation relative to a frame fixed to the ownship (Fig. 2). With respect to this relative frame, the set of ordinary differential equations is given in Eq. (4):

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

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 [Eq. (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.

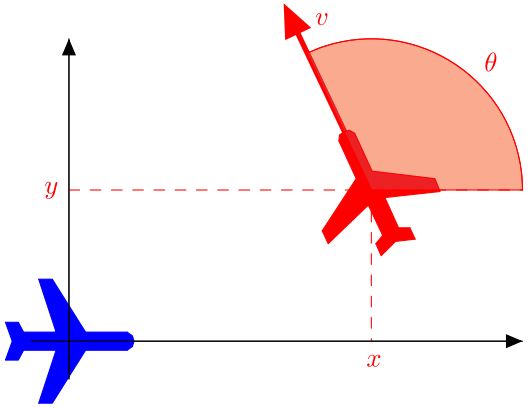

Fig. 2 Planar motion of the intruder with respect to a relative frame embedded in aircraft 1.

The next section discusses our approach to generate an expression for `monitor` [Eq. (3)]. We want the expression for `monitor` to ensure that the current and future behaviors of the pilot always preserve safety.

B. Safety Monitoring Using Differential Invariants

In Eq. (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 Eq. (4). The question-mark 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 Eq. (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 asterisk $(*)$ operator, in the exponent of Eq. (3), means that this sequence of 1) discrete control, 2) monitoring, and 3) 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.

To generate a safe and nonrestrictive monitor in Eq. (3), it is critical to effectively reason about the behavior of the solutions of the differential equations. Although a closed-form solution of Eq. (4) is known, handling it symbolically would require automated procedures to reason about transcendental functions, a problem that is undecidable in general. Instead, we proceed with a qualitative analysis of the solutions by means of differential invariants [24]: functions that remain constant when evaluated along any solution of the differential equations. In [25], 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 [25], we automatically generate the following nontrivial invariant function:

$$I_1(t) \equiv (v_2 \sin \theta(0))x(t) - (v_2 \cos \theta(0) - v_1)y(t) = I_1(0) \tag{5}$$

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 $x(0)$, $y(0)$, and $\theta(0)$.

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

$$I_2(t) \equiv -\omega_1\omega_2(x(t)^2 + y(t)^2) + 2v_2\omega_1 \sin \theta(t)x(t) + 2(v_1\omega_2 - v_2\omega_1 \cos \theta(t))y(t) + 2v_1v_2 \cos \theta(t) = I_2(0) \tag{6}$$

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 nontrivial helical trajectory (the radius of the circle changes over time) in the three-dimensional phase space (x, y, θ) (see Fig. 3).

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:

$$\texttt{safety} \equiv \forall\, t \geq 0, \qquad r(t) > p \tag{7}$$

Assuming $(\omega_1, \omega_2) = (0, 0)$, from Eq. (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:

[‡‡]The sine and cosine functions can be encoded algebraically as in [24].

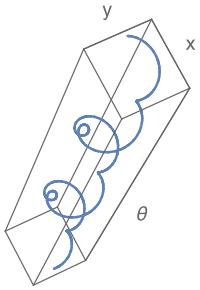

Fig. 3 Helical trajectory in the three-dimensional phase space (x,y,θ).

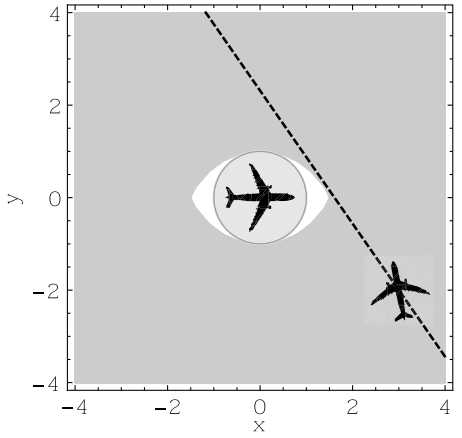

Fig. 4 Safe region (gray) relative to the ownship, centered in the disk of radius $p = 1$, when $(\omega_1,\omega_2) = (0,0)$.

$$|I_1(0)| = |I_1(t)| \tag{8}$$

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

$$\leq v_1|y(t)| + v_2 r(t)\left|\frac{x(t)}{r(t)}\sin\theta(0) - \frac{y(t)}{r(t)}\cos\theta(0)\right| \tag{10}$$

$$\leq v_1 r(t) + v_2 r(t) \tag{11}$$

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

$$\texttt{safety} \rightarrow |I_1(0)| \leq (v_1 + v_2)p \tag{12}$$

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

$$|I_1(0)| > (v_1 + v_2)p \rightarrow \texttt{safety} \tag{13}$$

Figure 4 depicts the safe region (gray) for $v_1 = 1$, $v_2 = 2$, $\omega_1 = \omega_2 = 0$, and $p = 1$. 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. For a given initial (relative) configuration $(x(0), y(0), \theta(0))$, if the line defined by the equation $I_1(t) = I_1(0)$ 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 $(0, 0)$. For instance, the black dashed line in Fig. 4 is a safe configuration. However, when the line $I_1(t) = I_1(0)$ intersects 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 nonequivalent inequality bounds, as in Eq. (11), are two common sources for false alarms. Hence, reducing false alarms requires more refined analysis, where one targets the sources of large overapproximations.

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

$$|I_2(0)| > 2v_1v_2 + 2p(v_2|\omega_1| + v_1|\omega_2|) + p^2|\omega_1\omega_2| \rightarrow \texttt{safety} \tag{14}$$

In this case, the safe region is a three-dimensional region in the configuration space (x, y, θ) described by Eq. (14). The projection of the safe region on the (x, y) plane is depicted in gray in Fig. 5. 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 $p = 1$.

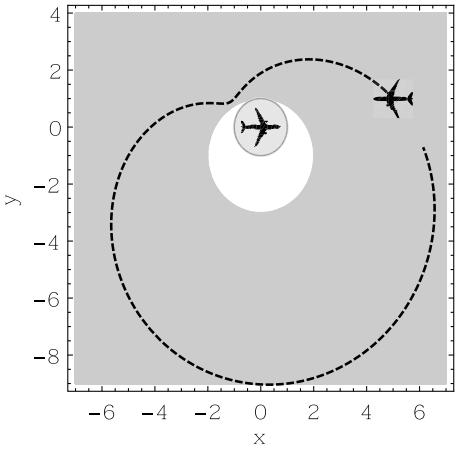

Fig. 5 Projection of the safe region (gray) relative to the ownship, centered in the disk of radius $p = 1$, when $(\omega_1,\omega_2) \neq (0,0)$.

The monitor of our collision-avoidance system [Eq. (3)] combines the safety bounds given in Eqs. (13) and (14) for the cases of zero and nonzero angular velocities, respectively:

$$\begin{aligned}\texttt{monitor} \equiv (\omega_1,\omega_2) = (0,0) \rightarrow I_1(0) > p(v_1+v_2) \wedge \\ (\omega_1,\omega_2) \neq (0,0) \rightarrow I_2(0) > 2v_1v_2 + 2p(v_2|\omega_1| + v_1|\omega_2|) + p^2|\omega_1\omega_2|\end{aligned} \tag{15}$$

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 d$\mathcal{L}$ formula, expressing that the safety property always holds for the whole system, is valid, i.e., it is true in all states:

$$\texttt{safety} \rightarrow [\texttt{cas}]\texttt{safety} \tag{16}$$

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

Theorem 1 states that, if two airplanes start at a safe state (where the separation is respected), and if the monitor defined in Eq. (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 [using an encoding of the transcendental functions (sine and cosine) with extra variables],[§§] 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 (4 GB of RAM and Intel Core 2 Duo 2.66 GHz).

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 and v_2; the intruder's angular velocity ω_2; and the desired separation p, the set of safe control inputs ω_1 for the ownship can be computed as those that satisfy 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, Fig. 6 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 [Eq. (15)]. The dashed trajectory shows the path of the ownship for $\omega_1 = -1$. This particular choice of ω_1 satisfies the monitor and is hence safe. Observe that the curved dashed line is entirely outside the potentially unsafe meshed zone.

When the ownship detects multiple intruders in its surroundings, the same constraints [Eq. (15)] can be used for each intruder to ensure separation by suitable coordination. Figure 7 shows a configuration with three aircraft where the meshed zone depicts the trajectories resulting from an unsafe command ω_1 being selected by the ownship. Observe how the meshed zone evolved from Fig. 6 where only one intruder was involved. The choices that are proven safe for the ownship are considerably reduced. The selected dashed trajectory ($\omega_1 = -1$) permits once again to evolve safely.

Figure 7 suggests that there exist scenarios where the ownship is "trapped"; that is, no proven safe choice exists for the ownship. In such cases, if all involved aircraft are equipped, the system can try to find an instantiation of commands $(\omega_1, \omega_2, \ldots)$ (within their respective physical limits) for which the pairwise safety condition [Eq. (15)] of each of the 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.

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 (Sec. IV.A); proof automation (Sec. IV.B); numerical issues (Sec. IV.C); and finally, scalability (Sec. 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.

[§§]The models and proof files are available online at http://www.lix.polytechnique.fr/~ghorbal/aerospace [retrieved 2014].

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

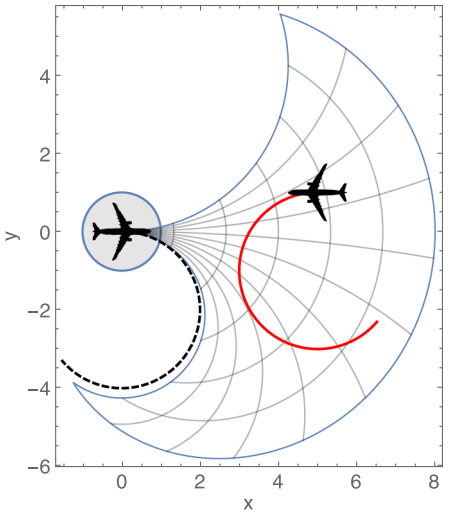

Fig. 6 The meshed zone depicts the projection of the trajectories that results when an unsafe command ω_1 is selected. The dashed trajectory shows a proven safe choice for the ownship.

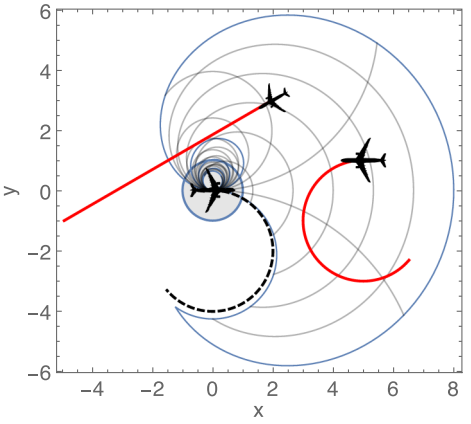

Fig. 7 The meshed zone, as in Fig. 6, accounts now for the additional intruder. The dashed trajectory shows a proven safe choice for the ownship.

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. Stochastic information can be incorporated into hybrid system models leading to stochastic hybrid systems, for which logic can be lifted [26]. However, the verification of stochastic hybrid systems is quite nontrivial. We therefore resort to nondeterministic choice as a simpler sound overapproximation of all possible behaviors. For instance, our hybrid system models would handle the windspeed uncertainty 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 decomposition ensures that we do not have to do complicated probability calculations every time we make a nondeterministic choice in our proof.

***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.

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

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 the joint probability bound Δ is essentially an integration question. At a high level, if Ω is a set of all possible joint parameter settings (e.g., $\boldsymbol{x} \in \mathbb{R}^n$), $\mathcal{E} \subset \Omega$ is the set of all joint settings that satisfy the local bounds (e.g., a box $\mathcal{E} = \{\boldsymbol{x} \in \mathbb{R}^n | l_i \leq x_i \leq u_i\}$), and p is a joint probability density function (e.g., a multivariate normal distribution), then we want to find a small Δ such that

$$\Delta \geq 1 - \int_{\mathcal{E}} p(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}$$

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 tradeoff, 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 Sec. III, but they are really estimated from a number of different noisy sensors, including radar and GPS positioning information broadcast by automatic dependent surveillance–broadcast (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. 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.

Second, 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 [19] 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 [27]. 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 anticorrelated. 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 [19], do not filter, or we need to 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: Aircraft Collision-Avoidance System 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 use 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 10th aircraft collision-avoidance system (ACAS X) [28]. This system, a successor to the traffic alert and collision-avoidance system (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-midair 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 a 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, [29]). 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 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.) 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, which is 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 Sec. 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 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 the a 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. (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.) This is because safety regions abstract away a lot of the details the MDPs had to wrestle with, and they use a symbolic polynomial representation that is richer than the tabular representation used by the MDP. Second, 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 [30].

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 and 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) evolutions 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 Sec. III), we want to automate the generation of the safety bounds (such as the safety monitor in Sec. III). Such directions have been recently explored within our group [31].

C. Numerical Issues

A computer cannot effectively perform real numbers computations. Real numbers are always truncated to fit a specific finite representation, which is 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 [32], 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 can neither 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 overapproximations because they lose the functional relations between the actuator (output) and the sensor 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 [5,33], 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 1980s (see [4] for a survey). Similar issues were noticed in other branches of computer science earlier, e.g., the curse of dimensionality observed by Bellman [34]. The framework has already pushed further the boundaries that previous approaches were limited to [8,11]. It allows for sound compositional verification of (linear and nonlinear) hybrid systems. That is, the system is decomposed into subsystems, and 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 [35] standard about the certification guidelines 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 [36] and abstract interpretation [5,37] have been shown to help verifying the complex software of current 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 aircraft 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 [36] and abstract interpretation [5,37]. Formal verification and validation techniques for hybrid systems such as theorem proving [7,8,11] are suitable, as we have illustrated in the collision-avoidance case study of Sec. III.

The benefit of formally verifying a collision-avoidance system is obvious: missing corner cases is unacceptable. Although exhaustive stress testing is infeasible, formal methods aim at proving the absence of mishaps, which is a highly desirable property. As a matter of fact, many collision-avoidance systems have been proposed (see [38] for an overview), but only few attempts have been made to formally verify such systems [23,39–41]. Nonverified collision-avoidance systems, such as those based on probabilistic modeling methods (e.g., [42,43]), 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., [44,45]) is too conservative. It assumes uniform distributions of events and resorts often to either large or nonflyable approximations of the dynamics. Likewise, the verification (not the design) of collision-avoidance systems based on controllable sets [23,46] uses either simplified (usually nonflyable) approximations, such as a succession of straight-line segments or instant turns as in [23]; or it falls back on nonexhaustive numerical testing when the computation is too complex [46]. Numerical computation of the reachable sets of hybrid systems by means of partial differential equations [47] has been applied to design and verify the safety of collision-avoidance systems for which an analytic solution is available. Such an approach is interesting, especially in sufficiently low dimensions, but it 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 [48].

Previous approaches of using theorem provers, such as PVS (in [39,40]), 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 roundabout maneuver modeled by Dubins curves [41]. The full capabilities of such a protocol, proposed first in [23], were formally verified and even refined with flyable entries and exits.

The proposed approach in this work is different form [23,41], in that it is not protocol-based. It hence allows for more flexible, yet flyable, collision resolutions. The verification does not consider an approximated, a continuous, nor a discretized trajectory, as in [23,39,40]. 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 [23,46,47], 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 [42–45], 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 [49,50]. In [51], the safety analysis of the aircraft autolander was performed by means of a 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 unmanned aerial vehicles (UAV) [18].

VI. Conclusions

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. The main challenges to extend the presented approach to formally verify real aerospace systems are sketched and different promising proposals of solutions are given. The complexity and nature of the aerospace systems require 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. With the astonishing advances in formal methods, and the surprisingly strong analysis that is possible today, it is believed that formal methods will become, and remain, a crucial ingredient in the design and certification of aerospace systems as supported by the recently published software considerations DO-178C. Indeed, 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.

Acknowledgments

This material is based upon work partially supported by the National Science Foundation (NSF) under NSF CAREER Award CNS-1054246, NSF EXPEDITION CNS-0926181, NSF CNS-0931985, and CNS-1035800. It is also supported by the U.S. Army Research Office under Award no. W911NF-09-1-0273 and the Federal Aviation Administration under grant number DTFAWA-11-C-00074.

References

[1] Trimble, S., "AUVSI: RQ-7 Likely Not to Blame for C-130 Collision," *Flight Daily News*, Aug. 2011.

[2] Johnson, C., "Review of the BFU Überlingen Accident Report," Dept. of Computing Science, Univ. of Glasgow, Glasgow, Scotland, 2004, www.dcs.gla.ac.uk/~johnson/Eurocontrol/Ueberlingen/Ueberlingen_Final_Report.PDF [retrieved Sept. 2014].

[3] 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

[4] Fix, L., "Fifteen Years of Formal Property Verification in Intel," *25 Years of Model Checking*, edited by Grumberg, O., and Veith, H., Springer–Verlag, Berlin, 2008, pp. 139–144.

[5] Delmas, D., and Souyris, J., "Astrée: From Research to Industry," *Static Analysis: Lecture Notes in Computer Science*, Vol. 4634, edited by Nielson, H., and Filé, G., Springer–Verlag, Berlin, 2007, pp. 437–451.

[6] 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," *AIAA Infotech@Aerospace 2010*, AIAA Paper 2010-3385, 2010.

[7] Platzer, A., "Differential Dynamic Logic for Hybrid Systems," *Journal of Automated Reasoning*, Vol. 41, No. 2, 2008, pp. 143–189. doi:10.1007/s10817-008-9103-8
[8] Platzer, A., "Logics of Dynamical Systems," *Logic in Computer Science (LICS'12)*, IEEE, Piscataway, NJ, 2012, pp. 13–24.
[9] Platzer, A., and Clarke, E. M., "Computing Differential Invariants of Hybrid Systems As Fixedpoints," *Computer Assisted Verification: Lecture Notes in Computer Science*, Vol. 5123, edited by Gupta, A., and Malik, S., Springer–Verlag, Berlin, 2008, pp. 176–189.
[10] Henzinger, T. A., "The Theory of Hybrid Automata," *Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science*, IEEE, Piscataway, NJ, 1996, pp. 278–292.
[11] Platzer, A., *Logical Analysis of Hybrid Systems: Proving Theorems for Complex Dynamics*, Springer–Verlag, Heidelberg, 2010.
[12] Platzer, A., and Quesel, J.-D., "KeYmaera: A Hybrid Theorem Prover for Hybrid Systems," *Automated Reasoning: Lecture Notes in Computer Science*, Vol. 5195, edited by Armando, A., Baumgartner, P., and Dowek, G., Springer–Verlag, Berlin, 2008, pp. 171–178.
[13] Loos, S. M., Platzer, A., and Nistor, L., "Adaptive Cruise Control: Hybrid, Distributed, and Now Formally Verified," *Formal Methods: Lecture Notes in Computer Science*, Vol. 6664, Springer–Verlag, Berlin, 2011, pp. 42–56.
[14] Loos, S. M., and Platzer, A., "Safe Intersections: At the Crossing of Hybrid Systems and Verification," *Intelligent Transportation Systems (ITSC'11)*, edited by Yi, K., IEEE, Piscataway, NJ, 2011, pp. 1181–1186. doi:10.1109/ITSC.2011.6083138
[15] Mitsch, S., Loos, S. M., and Platzer, A., "Towards Formal Verification of Freeway Traffic Control," *Proceedings of the 2nd International Conference on Cyber-Physical Systems (ICCPS'12)*, edited by Lu, C., IEEE, Piscataway, NJ, 2012, pp. 171–180.
[16] Platzer, A., and Quesel, J.-D., "European Train Control System: A Case Study in Formal Verification," *Formal Engineering Methods: Lecture Notes in Computer Science*, Vol. 5885, edited by Breitman, K., and Cavalcanti, A., Springer–Verlag, Berlin, 2009, pp. 246–265.
[17] Platzer, A., and Clarke, E. M., "Formal Verification of Curved Flight Collision Avoidance Maneuvers: A Case Study," *Formal Methods: Lecture Notes in Computer Science*, Vol. 5850, edited by Cavalcanti, A., and Dams, D., Springer–Verlag, Berlin, 2009, pp. 547–562.
[18] Loos, S. M., Renshaw, D. W., and Platzer, A., "Formal Verification of Distributed Aircraft Controllers," *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC'13)*, edited by Belta, C., and Ivancic, F., ACM, Philadelphia, April 2013, pp. 125–130.
[19] Mitsch, S., Ghorbal, K., and Platzer, A., "On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles," *Robotics: Science and Systems IX*, edited by Newman, P., Fox, D., and Hsu, D., Technische Universität Berlin, Berlin, June 2013, http://www.roboticsproceedings.org/rss09/p14.pdf [retrieved Sept. 2014].
[20] Kouskoulas, Y., Renshaw, D. W., Platzer, A., and Kazanzides, P., "Certifying the Safe Design of a Virtual Fixture Control Algorithm for a Surgical Robot," *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control (HSCC'13)*, edited by Belta, C., and Ivancic, F., ACM, Philadelphia, April 2013, pp. 263–272.
[21] 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*, Vol. 79, No. 3, 1957, pp. 497–516. doi:10.2307/2372560
[22] Bicchi, A., Marigo, A., Pappas, G., Pardini, M., Parlangeli, G., Tomlin, C., and Sastry, S., "Decentralized Air Traffic Management Systems: Performance and Fault Tolerance," *Proceedings of the IFAC International Workshop on Motion Control (MC'98)*, The International Federation of Automatic Control, 1998, pp. 259–264.
[23] 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. doi:10.1109/9.664154
[24] Platzer, A., "Differential-Algebraic Dynamic Logic for Differential-Algebraic Programs," *Journal of Logic and Computation*, Vol. 20, No. 1, 2010, pp. 309–352. doi:10.1093/logcom/exn070
[25] Ghorbal, K., and Platzer, A., "Characterizing Algebraic Invariants by Differential Radical Invariants," *Tools and Algorithms for the Construction and Analysis of Systems: Lecture Notes in Computer Science*, Vol. 8413, edited by Ábrahám, E., and Havelund, K., Springer–Verlag, Berlin, 2014, pp. 279–294.
[26] Platzer, A, "Stochastic Differential Dynamic Logic for Stochastic Hybrid Programs," *Automated Deduction: Lecture Notes in Computer Science*, Vol. 6803, edited by Bjørner, N., and Sofronie-Stokkermans, V., Springer–Verlag, Berlin, 2011, pp. 431–445.
[27] Kalman, R. E., "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering*, Vol. 82, No. 1, 1960, pp. 35–45. doi:10.1115/1.3662552
[28] Kochenderfer, M. J., Holland, J. E., and Chryssanthacopoulos, J. P., "Next-Generation Airborne Collision Avoidance System," *Lincoln Laboratory Journal*, Vol. 19, No. 1, 2012, pp. 17–33, https://www.ll.mit.edu/publications/journal/pdf/vol19_no1/19_1_1_Kochenderfer.pdf [retrieved Sept. 2014].
[29] Puterman, M. L., *Markov Decision Processes: Discrete Stochastic Dynamic Programming*, Wiley, Hoboken, NJ, 2009.
[30] von Essen, C., and Giannakopoulou, D., "Analyzing the Next Generation Airborne Collision Avoidance System," *Tools and Algorithms for the Construction and Analysis of Systems: Lecture Notes in Computer Science*, Vol. 8413, edited by Ábrahám, E., and Havelund, K., Springer–Verlag, Berlin, 2014, pp. 620–635.
[31] Mitsch, S., and Platzer, A., "ModelPlex: Verified Runtime Validation of Verified Cyber-Physical System Models," *Runtime Verification: Lecture Notes in Computer Science*, Vol. 8734, edited by Bonakdarpour, B., and Smolka, S. A., Springer–Verlag, Berlin, 2014, pp. 199–214.
[32] Bouissou, O., Goubault, E., Putot, S., Tekkal, K., and Vedrine, F., "HybridFluctuat: A Static Analyzer of Numerical Programs Within a Continuous Environment," *Computer Aided Verification: Lecture Notes in Computer Science*, Springer–Verlag, Berlin, 2009, pp. 620–626.
[33] Venet, A, "The Gauge Domain: Scalable Analysis of Linear Inequality Invariants," *Computer Aided Verification: Lecture Notes in Computer Science*, Vol. 7358, edited by Madhusudan, P., and Seshia, S., Springer–Verlag, Berlin, 2012, pp. 139–154.
[34] Bellman, R. E., *Dynamic Programming*, Princeton Univ. Press, Princeton, NJ, 1962.
[35] "Software Considerations in Airborne Systems and Equipment Certification," RTCA STD DO-178C, Washington, D.C., 2012.
[36] Miller, S. P., Whalen, M. W., and Cofer, D. D., "Software Model Checking Takes Off," *Communications of the ACM*, Vol. 53, No. 2, 2010, pp. 58–64. doi:10.1145/1646353
[37] Souyris, J., Wiels, V., Delmas, D., and Delseny, H., "Formal Verification of Avionics Software Products," *Formal Methods: Lecture Notes in Computer Science*, Vol. 5850, edited by Cavalcanti, A., and Dams, D., Springer–Verlag, Berlin, 2009, pp. 532–546.
[38] Kuchar, J. K., and Yang, L. C., "A Review of Conflict Detection and Resolution Modeling Methods," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 1, No. 4, 2000, pp. 179–189. doi:10.1109/6979.898217
[39] Dowek, G., Muñoz, C., and Carreño, V. A., "Provably Safe Coordinated Strategy for Distributed Conflict Resolution," *2005 AIAA Guidance Navigation, and Control Conference and Exhibit*, AIAA Paper 2005-6047, 2005.
[40] Galdino, A. L., Muñoz, C., and Ayala-Rincón, M., "Formal Verification of an Optimal Air Traffic Conflict Resolution and Recovery Algorithm," *Logic, Language, Information and Computation: Lecture Notes in Computer Science*, Vol. 4576, edited by Leivant, D., and de Queiroz, R., Springer–Verlag, Berlin, 2007, pp. 177–188.
[41] 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. doi:10.1007/s10703-009-0079-8
[42] Prandini, M., Hu, J., Lygeros, J., and Sastry, S., "A Probabilistic Approach to Aircraft Conflict Detection," *IEEE Transactions on Intelligent Transportation Systems*, Vol. 1, No. 4, 2000, pp. 199–220. doi:10.1109/6979.898224

[43] Lygeros, J., and Prandini, M., "Aircraft and Weather Models for Probabilistic Collision Avoidance in Air Traffic Control," *Proceedings of the 41st IEEE Conference on Decision and Control*, Vol. 3, IEEE, Piscataway, NJ, 2002, pp. 2427–2432.
[44] Lachner, R., "Collision Avoidance as a Differential Game: Real-Time Approximation of Optimal Strategies Using Higher Derivatives of the Value Function," *IEEE International Conference on Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation, 1997*, Vol. 3, IEEE, Piscataway, NJ, 1997, pp. 2308–2313.
[45] Bilimoria, K. D., Lee, H. Q., Mao, Z.-H., and Feron, E., "Comparison of Centralized and Decentralized Conflict Resolution Strategies for Multiple-Aircraft Problems," *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2000-4268, 2000.
[46] 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.
[47] Mitchell, I. M., Bayen, A. M., and Tomlin, C. J., "A Time-Dependent Hamilton-Jacobi Formulation of Reachable Sets for Continuous Dynamic Games," *IEEE Transactions on Automatic Control*, Vol. 50, No. 7, 2005, pp. 947–957.
doi:10.1109/TAC.2005.851439
[48] 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," *Computer Aided Verification: Lecture Notes in Computer Science*, Vol. 6806, edited by Gopalakrishnan, G., and Qadeer, S., Springer–Verlag, Berlin, 2011, pp. 379–395.
[49] 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," *Formal Methods: Lecture Notes in Computer Science*, Vol. 4085, edited by Misra, J., Nipkow, T., and Sekerinski, E., Springer–Verlag, Berlin, 2006, pp. 64–80.
[50] Umeno, S., and Lynch, N. A., "Safety Verification of an Aircraft Landing Protocol: A Refinement Approach," *Hybrid Systems: Computation, and Control: Lecture Notes in Computer Science*, Vol. 4416, edited by Bemporad, A., Bicchi, A., and Buttazzo, G., Springer–Verlag, Berlin, 2007, pp. 557–572.
[51] 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.
doi:10.2514/1.21562

M. Davies
Associate Editor