Formal Verification and Synthesis for an Air Traffic Management System

Adilson Luiz Bonifácio *Arnaldo Vieira Moura*
João Batista de Camargo Jr.
Jorge Rady de Almeida Jr.

Relatório Técnico IC–00-05

Fevereiro de 2000

Formal Verification and Synthesis for an Air Traffic Management System

Adilson Luiz Bonifácio* Arnaldo Vieira Moura

João Batista de Camargo Jr.† Jorge Rady de Almeida Jr.‡

February, 2000

Abstract

The aim of this work is to apply formal specification techniques to model real-time distributed systems arising from real-world applications. The target system is an Air Traffic Management System (ATM), using the Traffic alert and Collision Avoidance System (TCAS) protocol. The formal models developed here are based on the notion of hybrid automata. Semi-automatic tools are used in the verification of the models, and some important system parameters are synthesized using a parametric analysis. All results were obtained on a 350MHz desktop PC, with 320MB of main memory.

Key-Words: Air Traffic Management, Analysis, Synthesis, Verification, Hybrid Systems.

1 Introduction

The use of formal specifications and verification techniques in real-world system development processes has become more and more important, especially when such processes focus on distributed systems that control critical applications, where an operational fault can cause irreparable damages. Among a wide variety of distributed systems, the ones involving reactive and real-time responses are the most difficult to model accurately. In addition, certain kind of critical applications, known as *hybrid systems*, present a dual nature, in the sense that the system behavior is described by continuous dynamic profiles, regulated by the intervention of discrete events [8, 7, 6]. This introduces further complications and subtleties in the modeling and analysis of such systems.

Recently, the mathematical theory of hybrid automata has been gaining momentum as a good alternative to specify and verify hybrid systems [21, 2, 13, 15, 17]. Essentially, hybrid

*Partial support from CAPES, DS - 44/97

†Polytechnic School, University of São Paulo, Av. Prof. Luciano Gualberto, Trav. 3, N. 158, São Paulo, Brazil, 05508-900

‡Polytechnic School, University of São Paulo, Av. Prof. Luciano Gualberto, Trav. 3, N. 158, São Paulo, Brazil, 05508-900

automata are finite automata whose states include a dynamic specification of the system evolution, and whose state transitions model an abrupt change in the dynamic behavior of the system. Each component of the distributed system is modeled by an appropriate automaton. The overall system behavior is, then, captured by the corresponding product automaton [15, 17]. The communication between the system components is regulated by the exchange of messages between the independent automata. Messages are exchanged during state transitions taking place in the individual automata. Synchronism between the individual automata is also attained by the use of a shared memory, to which all the component automata have access [15, 17]. The notion of hybrid automata is adequate to model systems composed of several distributed communicating components, each one with a simple dynamic behavior. In contrast, classic control theories deal with systems with a few distributed components whose dynamic features are more complex.

In this work, hybrid automata are used to model the complexities of a real-world distributed hybrid system, consisting of an Air Traffic Management System (ATM). The ATM system coordinates many different autonomous aircraft which compete for routes in a section of airspace. The Traffic alert and Collision Avoidance System (TCAS) protocol is used to help manage the traffic in the vicinity of a cruising aircraft. The TCAS is an on-board conflict detection and resolution algorithm. Its task is to monitor the traffic around the aircraft, detect possible threats and advise on how to resolve these conflicts, for vertical separations.

The models explored in this work consider two aircraft flying in the same airspace, in opposite directions. This scenario captures only a fragment of the complexities of a real ATM system. Even so, the results obtained contribute a positive step towards the validation and the verification of the functionalities of the actual system. In all cases studied, a safe system operation is the main property to be verified. At the moment of this writing, the authors are unaware of other direct applications of hybrid automata techniques either to verify safety properties for ATM systems [20], or to synthesize values for operational parameters of such systems.

The operation of the ATM system requires an intense exchange of messages and a complex activity of coordination between all participating agents. Due to the complexity of the resulting model, semi-automatic computational tools are used to effectively bear on the formal specifications. Most of the time, models for realistic systems result in an automaton of such complexity that the cooperation of these automatic tools becomes essential. Here, the HyTech tool [12, 14, 13] was used to analyze the models.

This report is organized as follows. Section 2 presents the notion of hybrid automata, illustrated by some simple examples. Section 3 describes the air traffic management system. The system operation is explained, together with some of its variants and special procedures. Section 4 discusses the models that were developed to verify and to validate the system operation. The synthesis of values for certain important parameters is also described here. The last section concludes with some final observations.

2 Hybrid Automata

In this section, the notion of a hybrid automaton is introduced.

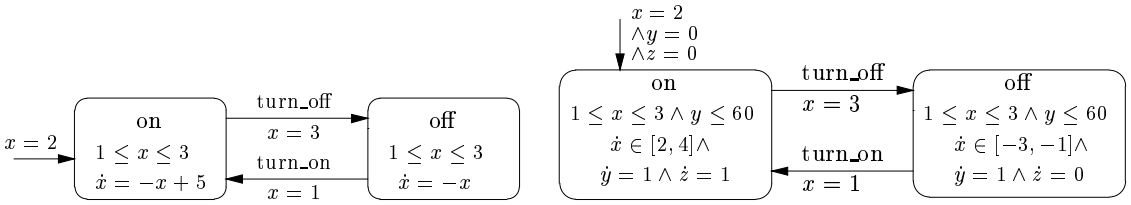
2.1 Basic Definitions

A *hybrid automaton* is a formal system $A = (X, V, \mathit{flow}, \mathit{init}, \mathit{inv}, E, \mathit{jump}, \Sigma, \mathit{syn})$, where:

1. $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of *variables*. The number n is the *dimension* of A.
2. V is a finite set of *operation modes*, or just *modes*.
3. For every mode $v \in V$, *flow(v)*, the *continuous activity condition* of mode v, is a predicate over the set of variables $X \cup \dot{X}$. Here, $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$.[1]
4. For every mode $v \in V$, *inv(v)*, the *invariant condition* for mode v, is a predicate over X. When in mode v, the invariant condition *inv(v)* must always be satisfied.
5. The *initial conditions* are given by the component *init*. For every mode $v \in V$, *init(v)* is a predicate over X. The automaton can start in mode v only if the initial condition *init(v)* is satisfied.
6. The multi-set E is formed by pairs (v, v'), where $v, v' \in V$ are operation modes. The multi-set describes the *transitions* of the automaton.
7. The component *jump* describes the *phase change conditions*. For every transition $e \in E$, *jump(e)* is a predicate over the set $X \cup X'$. Here, $X' = \{x'_1, \ldots, x'_n\}$. The primed variable x' is used to indicate the (new) value of the corresponding variable x after a transition.
8. Σ is a finite set of *event labels*, or just *events*. The partial function *syn* maps transitions in E into events of Σ.

A simple example [15] of a hybrid automaton is shown in Figure 1(a). This model captures the simple dynamics of a gas heater. The model uses a real variable x, which evolves deterministically in time, to model the temperature of a container. The heater is turned off when the temperature reaches 3 degrees, and it is turned back on when the temperature falls to 1 degree. The evolution of x is computed from the initial conditions, from differential equations present in the modes, and from the transitions between modes. In the formal set up for this example $n = 1$, $X = \{x\}$, and $V = \{\mathit{on}, \mathit{off}\}$. The continuous activity condition for mode *on* is given by the predicate $\dot{x} = -x + 5$, which describes an exponential rise of the temperature x. For mode *off* the continuous activity condition is $\dot{x} = -x$, describing an exponential fall in the temperature. For both operation modes, the invariant condition is $1 \leq x \leq 3$. The initial condition for mode *on* is $x = 2$, and for mode *off* it is `false`. The latter condition is not depicted in the figure. There are two transitions in Figure 1(a), namely (*on*, *off*) and (*off*, *on*). The transition (*on*, *off*) has a phase change

[1] For any variable x, the first derivative of x with respect to time is indicated by $\dot{x}$.

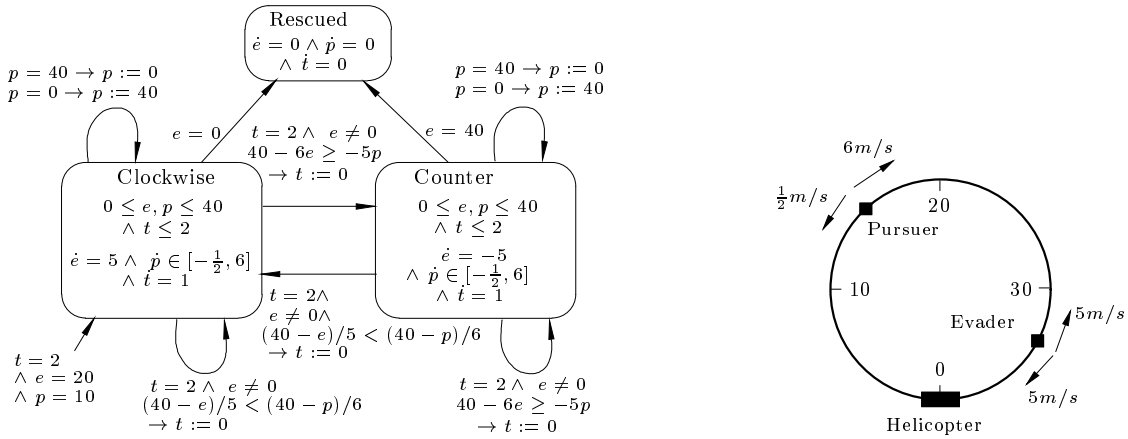


(a) Simple heater (b) Heater and safety requirement

Figure 1: Hybrid automata: an example

condition given by the predicate $x = 3 \wedge x' = x$, and the transition (*off*, *on*) has the phase change condition given by the predicate $x = 1 \wedge x' = x$. The trivial condition $x' = x$ is not shown in the figure. In this model, the value of the real variables do not change when a transition is taken. It is common to use "guards", in the sense of [10], to represent the phase change conditions in the graphical representation of hybrid automata. A guard such as $(x_1 = x_2) \rightarrow (x_1 := 2x_2)$ gives the precondition $x_1 = x_2$ and the postcondition $x_1' = 2x_2$, that are to be enforced during a phase change. These conditions are equivalent to the complete predicate $x_1 = x_2 \wedge x_1' = 2x_2 \wedge x_2' = x_2$. The set of discrete events for this example is $\Sigma = \{turn\_on, turn\_off\}$. The function *syn* associates the event *turn_on* to the transition (*off*, *on*), and associates the event *turn_off* to the transition (*on*, *off*). The presence of these events will allow for the synchronization between distributed automata, as will be described in the sequel.

Another interesting example [1] of a hybrid automaton is presented in Figure 2. This model describes a pursuit game. The scenario is depicted in Figure 2(b). There is a pursuer, in a car, chasing an evader on a 40 meters long circular track. The car can travel up to 6 meters per second in the clockwise direction, but only up to 1/2 meters per second when going counterclockwise, since it must use its reverse gear to travel in this direction. The evader is on a bicycle, and can travel at 5 meters per second in either direction. However, it may decide to change its direction of movenemt only at certain instants of time, separated by exactly two seconds. The goal of the evader is to avoid the pursuer, at all costs, by reaching a helicopter stationed at the zero mark on the track. In order to win the game, the evader must escape the pursuer, no matter what strategy the pursuer chooses to use. The game is modeled by the automaton shown in Figure 2(a). Variable p gives the position of the pursuer on the track. Similarly, variable e models the position of the evader. The clock t measures the delay between the choices made by the evader. There are three operation modes: going clockwise, modeled by the mode *clockwise*, going counterclockwise, modeled by the mode *counter*, and a rescued mode, named *rescued*. In the *clockwise* and *counter* modes, the variable e evolves according to the differential equations $\dot{e} = 5$ and $\dot{e} = -5$, respectively, and the variable p observes the differential inequality $\dot{p} \in [-1/2, 6]$. The



(a) Pursuit automaton

(b) Pursuit depicted

Figure 2: The pursuit game

nondeterministic choice for the value of $\dot{p}$ reflects the pursuer ability to instantly change its direction of movement. The clock t, of course, obeys the dynamic equation $\dot{t} = 1$, in both of these modes.

The invariant condition in the *clockwise* and *counter* modes is given by $0 \leq e, p \leq 40$. This, together with the upper state transitions, depicted as self-loops in these modes, reflect the fact that the track is circular with a 40 meters perimeter.

The lower self-loops and the transitions between the *clockwise* and *counter* modes describe the evader's simple strategy. It computes the time it will take for him and the pursuer to reach the escape helicopter, when both are moving in the clockwise direction. If his time is shorter, the evader moves clockwise; if it is longer, he goes counter-clockwise.

The *rescued* mode signals that the evader has reached the helicopter. The transition to this mode is taken when the evader reaches the helicopter, either when he is moving clockwise or when he is moving counter-clockwise.

The evader is initially set at position 20 on the track, directly opposite to the helicopter. The pursuer starts at position 10. The HyTech tool can automatically determine that, in this scenario, the evader has a winning strategy.

A number of other related works, dealing with models based on hybrid automata, can be found in [18, 16, 17].

2.2 The Product Automaton

The model of a complex system comprises several individual automata that operate concurrently. The synchronism of the global system is obtained by:

1. Requiring simultaneity of phase transitions when they are labeled by the same discrete event, and

2. Using shared variables.

The synchronism is automatically incorporated into the product automaton, to be described next. Let A_1 and A_2 are hybrid automata, where $X_1 \cap X_2 = \emptyset$. The *product automaton* of A_1 and A_2 is a system $A = (X_1 \cup X_2, V_1 \times V_2, \mathit{flow}, \mathit{init}, \mathit{inv}, E, \mathit{jump}, \Sigma_1 \cup \Sigma_2, \mathit{syn})$, where the *flow*, *init* and *inv* components are simple conjunctions:

$$\begin{aligned} \mathit{flow}((v_1, v_2)) &= \mathit{flow}(v_1) \wedge \mathit{flow}(v_2); \\ \mathit{inv}((v_1, v_2)) &= \mathit{inv}(v_1) \wedge \mathit{inv}(v_2); \\ \mathit{init}((v_1, v_2)) &= \mathit{init}(v_1) \wedge \mathit{init}(v_2). \end{aligned}$$

For the phase transitions, $e = ((v_1, v_1'), (v_2, v_2')) \in E$ if and only if one of the following conditions hold:

1. $v_1 = v_1'$, $e_2 = (v_2, v_2') \in E_2$ and $\mathit{syn}_2(e_2) \notin \Sigma_1$; $\mathit{jump}(e) = \mathit{jump}_2(e_2)$ and $\mathit{syn}(e) = \mathit{syn}_2(e_2)$.

2. $v_2 = v_2'$, $e_1 = (v_1, v_1') \in E_1$ and $\mathit{syn}_1(e_1) \notin \Sigma_2$; $\mathit{jump}(e) = \mathit{jump}_1(e_1)$ and $\mathit{syn}(e) = \mathit{syn}_1(e_1)$.

3. $e_1 = (v_1, v_1') \in E_1$, $e_2 = (v_2, v_2') \in E_2$, $\mathit{syn}_1(e_1) = \mathit{syn}_2(e_2)$. In this case, $\mathit{jump}(e) = \mathit{jump}_1(e_1) \wedge \mathit{jump}_2(e_2)$ and $\mathit{syn}(e) = \mathit{syn}_1(e_1)$. The synchronism is imposed by forcing the transitions labeled by the same event to occur simultaneously.

The product of several automata is obtained by accumulating the result of computing the product of each individual automaton, in turn.

2.3 Regions and Trajectories

An *atomic linear predicate* is an inequality between rational constants and linear combinations of variables with rational coefficients, such as $2x + 4y - 7z/2 \leq -10$. A *convex linear predicate* is a finite conjunction of atomic linear predicates, and a *linear predicate* is a finite disjunction of convex linear predicates.

A *convex region* of a hybrid automaton A of dimension n is a convex polyhedron in $\mathbb{R}^n$. A *region* is a finite union of convex regions. Given a predicate φ, the region determined by φ is denoted by $[\varphi]$, and is called the φ-region. A *configuration* of a hybrid automaton A is a pair $q = (v, a)$ consisting of an operation mode $v \in V$ and a vector $a = (a_1, \ldots, a_n)$ in $\mathbb{R}^n$. The configuration (v, a) is *admissible* if $a \in [\mathit{inv}(v)]$. The configuration (v, a) is *initial* if $a \in [\mathit{init(v)}]$. In the example depicted on Figure 1(a), the configuration $(\mathit{on}, 0.5)$ is not admissible and that the configuration $(\mathit{on}, 2)$ is initial.

Let $q = (v, a)$ and $q' = (v', a')$ be two configurations of A. The pair (q, q') is a *phase change* of A if $e = (v, v')$ is a transition in E and $(a, a') \in [\mathit{jump}(e)]$. The pair (q, q') is a *continuous activity* of A if $v = v'$, if there is a nonnegative real $\delta \in \mathbb{R}$ (the duration of the continuous activity) and if there is also a differentiable function $\rho : [0, \delta] \to \mathbb{R}^n$ (the curve of the continuous activity), such that the following requirements are satisfied:

1. Endpoints: $\rho(0) = a$ and $\rho(\delta) = a'$;
2. Admissibility condition: for all time instants $t \in [0, \delta]$, the configuration $(v, \rho(t))$ is admissible;
3. Invariant condition: $\rho(t) \in [inv(v)]$, for all time instants $t \in [0, \delta]$;
4. Continuous activity condition: if $\dot{\rho} : [0, \delta] \to \mathbb{R}^n$ is the first derivative of ρ, then, for all time instants $t \in [0, \delta]$, $(\rho(t), \dot{\rho}(t)) \in [flow(v)]$.

A *trajectory* of A is a finite sequence $q_0, q_1, \ldots, q_k$, of admissible configurations, where each pair (q_j, q_{j+1}) of consecutive configurations is either a phase change or a continuous activity of A. Such a trajectory is said to *start* at q_0. A configuration q' of A is *reachable from* a configuration q if q' is the last configuration of some trajectory of A starting at q. An *initial trajectory* of A is any of its trajectory that starts at an initial configuration. A configuration q' of A is simply *reachable* if it is reachable from an initial configuration of A. In the example illustrated in Figure 1(a), all admissible configurations are reachable.

Given a region φ, $Post(\varphi)$ is the region formed by all those configurations q' for which there exists a configuration $q \in [\varphi]$ such that q' is reachable from q through a continuous activity or of a phase change of A. Starting with $\varphi_0 = \varphi$, the iteration of this process will compute the regions $\varphi_{k+1} = Post(\varphi_k)$, for $k = 0, 1, \ldots$. If a region φ_k satisfies $\varphi_k = \varphi_{k+1}$, then the process converges and region φ_k contains all the configurations of A that are reachable via some trajectory that starts from a configuration in φ_0. When the process converges, region φ_k is denoted by $Post^\star(\varphi_0)$. A backward process can also be defined in a similar way, giving rise to a *Pre* operator.

2.4 Linear Hybrid Automata

A hybrid automaton A is a *linear hybrid automaton*, if it satisfies the following requirements:

1. For every operation mode $v \in V$, the continuous activity condition *flow(v)*, the invariant condition *inv(v)* and the initial condition *init(v)*, are convex linear predicates. Moreover, for every transition $e \in E$, the phase change condition *jump(e)* is a convex linear predicate.
2. For every operation mode $v \in V$, the continuous activity condition *flow(v)* is a predicate over the set $\dot{X}$ only.

The second requirement prohibits continuous activities such as $\dot{x} = x$. On the other hand, it still allows the use of stopwatches and clocks with bounded drift, the latter being described by dynamic conditions such as $\dot{x} \in [1 - \epsilon_1, 1 + \epsilon_2]$, for some constants ϵ_1 and ϵ_2. The heater automaton of Figure 1(a) is not a linear hybrid automaton, because the continuous activity condition is not a predicate over the set $\dot{X}$ only.

There are two techniques for replacing a nonlinear hybrid automaton by a linear hybrid automaton [15]. The first technique, called *clock translation*, replaces variables that cause nonlinearity by clocks. The second technique, called *linear phase-portrait approximation*, replaces nonlinear predicates by more relaxed linear predicates.

The idea of clock translation is that sometimes the value of a variable can be determined from a past value and the time that has elapsed since the variable was last attributed that value. The variable x of a hybrid automaton A is *clock-translatable* if the following two requirements hold:

1. Solvability: in each continuous activity condition *flow(v)*, all occurrences of x and $\dot{x}$ are within conjuncts of the form $\dot{x} = g^v(x)$, where $g^v : \mathbb{R} \to \mathbb{R}$ is an integrable function with constant sign over the interval determined by the invariant condition. And, in the invariant, initial, and phase change conditions, all occurrences of x and x' are within conjuncts of the form $x' = x$, or $x \sim c$, or $x' \sim c$, where $\sim$ is an inequality operator and c is a rational constant.

2. Initialization: for every operation mode v, the initial condition *init(v)* is of the form $x = c$, for some constant c. And, for every transition (v, v'), it is the case that $g^v = g^{v'}$ and, either $jump(v, v')$ implies $x' = x$ or $jump(v, v')$ implies $x' = c$, for some constant c.

In accordance with these conditions, the value of x is determined by: (i) the time since it was last reassigned to a constant; and (ii) the value of that constant. Therefore, all invariant, initial, and phase change conditions on the clock-translatable variable x can be translated into conditions over a clock t^x. The clock t^x is restarted whenever x is reassigned to a constant. If necessary, operation modes are split in order to accommodate reassignments of x to different constants.

In the linear phase-portrait approximation the idea is to relax all nonlinear *flow*, *inv*, *init*, and *jump* predicates. Each such nonlinear predicate p is replaced by a linear predicate p' such that p implies p'. The linear hybrid automaton of Figure 1(b) is a linear phase-portrait approximation of the heater system, shown in Figure 1(a). For the moment, ignore the new variables y and z. All the invariant, initial, and phase change conditions are linear. Only the continuous activity conditions needed to be relaxed. For the operation mode *on*, the invariant condition $1 \leq x \leq 3$ and the continuous activity condition $\dot{x} = -x + 5$ imply that this predicate can be relaxed to the linear condition $\dot{x} \in [2, 4]$. In the same way, on the operation mode *off*, the continuous activity condition can be relaxed to the linear condition $\dot{x} \in [-3, -1]$.

Observe that clock translation preserves the trajectories of a system and the linear phase-portrait approximation adds trajectories to a system. Hence, if a safety property is satisfied in the relaxed system, then the property is also satisfied in the original system. However, if the relaxed system violates a safety property, then it must be checked if the discovered error trajectory is a valid trajectory of the original system. If not, the analysis is inconclusive and the approximations must be refined, for example, by splitting operation modes.

The following theorem can be shown to hold [2]: if A is a linear hybrid automaton, and if $[\varphi]$ is any region of A, then the calculation of $Post^\star(\varphi)$ converges. This theorem supports the construction of (semi) automatic computational tools for the analysis of linear hybrid automata.

2.5 Safety and Parametric Analysis

A *safety requirement* is a set of predicates imposed on the system configurations. Usually, a safety requirement is described by the set of values that the system variables can attain. A configuration is *safe* if it satisfies all the safety requirements associated with the model; otherwise the configuration is *unsafe*. A model is *safe* if all its reachable configurations are safe; otherwise the model is *unsafe*. Given a safety requirement as a predicate φ over the configurations of A, the region $reach = Post^{\star}([init])$ is computed, and the intersection $reach \cap [\neg\varphi]$ is obtained. If the resulting region is empty, the safety requirements are satisfied and the system is safe; otherwise they are violated and the system is unsafe.

System parameters are symbolic constants which assume fixed values. A parametric analysis determines value intervals for certain system parameters in such a way as to guarantee that no safety requirement is violated. This process, therefore, synthesizes maximum and minimum values for these parameters. In a hybrid automaton, a parameter α can be represented by a variable whose value never changes. This condition can be attained by imposing the condition $\dot{\alpha} = 0$ in all operation modes and, further, by imposing the condition $\alpha' = \alpha$ over all state transitions of the automaton. A value $a \in \mathbb{R}$ is *safe for a parameter* α if no unsafe configuration is reachable when the initial condition $\alpha = a$ is adjoined to the all operating modes of the automaton.

Returning to the heater example, depicted in Figure 1(a), a safety requirement for the heater could be: "in the first hour of operation, the heater should not be on by more than 2/3 of the time". Figure 1(b) depicts a relaxed version of that automaton, where the new clock variable y measures the total elapsed time and the new stopwatch variable z measures the time the heater stays in operation. It can be demonstrated that the relaxed version is safe with respect to these requirements [15]. It follows that the original model is also safe with respect to the same requirements.

2.6 Computational Tools

In general, the modeling of realistic systems results in a product automaton of such complexity that the verification phase can only be successfully done with the aid of (semi) automatic computational tools. The software HyTech [13, 15, 14, 12] was developed as a tool to aid in the analysis and verification of linear hybrid automaton models. The model is described to the tool via an input file that comprises two parts. The first part describes the individual hybrid automata. The product automaton is computed automatically by the tool. The second part is a sequence of analysis commands.

The HyTech tool can automatically verify that the relaxed heater model illustrated in Figure 1(b) satisfies all the posed safety requirements [15]. In the same vein, in the pursuit game of Figure 2, the tool can also verify that the evader has a winning escape strategy [1].

Today, HyTech is one of the only tools that supports semi-automatic analysis of linear hybrid automata models. It has been used in a variety of diverse situations and further examples can be found in [2, 15, 14, 12].

UPPAAL [19, 4, 5, 3] is another example of an automatic verification tool. It is based on timed-automata, a more restricted notion than that of linear hybrid automata. The

former notion can only work with clocks, and does not support the use of more complex differential equations to describe dynamic system profiles.

3 Description of an Air Traffic Management System

Air Traffic Management Systems (ATM) are very complex and critical systems, that operate under tight conditions and are composed of several distributed components. This makes it hard for such systems to be formally verified. On the other hand, the need for a formal, rigorous verification of all the system safety requirements is unquestionable, since any safety fault can result in intolerably high damages, both in terms of properties and lifes. Several faults are known to have occurred in aviation history, causing terrible accidents, due the unsafe operation of these systems [23].

3.1 The Air Traffic Control

In a typical flight, after the aircraft enter en route, the Air Traffic Control (ATC) monitors the flight by radio [11]. After an ATC acknowledges radar contact with the signaling plane, it starts to transmit values for heading and altitude, based on the aircraft present coordinates. Depending on the flight plan, one aircraft can switch many times from one en route controller to another, during the course of its flight. En route controllers are assigned to specific geographic areas, and they work to maintain the safe separation of passing aircraft that cross their sector of airspace. While all airline aircraft are controlled every step of the way, the same level of positive control does not always apply to all other aircraft. Some aircraft can, and often do, fly in uncontrolled airspace, outside the ATC range. In general, these uncontrolled air spaces are areas below the cruise lanes used by airline aircraft.

General aviation aircraft are allowed to fly under visual flight rules, or VFR, when weather and visibility are good. In these conditions, they do not have to file a flight plan, and they do not have to be in touch with air traffic control, unless they choose to operate in or out of an airport that has a control tower. Under VFR rules, pilots are responsible for maintaining adequate separation from other aircraft, and that is why these rules are sometimes called the "see and be seen" rules.

Instrument flight rules, or IFR, on the other hand, are the rules under which general aviation aircraft must fly in bad weather and low visibility. In this case, pilots must be in contact with the ATC and must file a flight plan. They also must be "instrument certified", meaning that they are proficient at navigating and flying their aircraft using cockpit instruments only, without the benefit of good visibility. Airline flights always operate under instrument flight rules, regardless of weather.

Recent technological advances made it possible to implement sophisticated functions, such as navigation and aircraft separation, in the system components responsible for the control and surveillance of a sector of airspace [24]. This more advanced technology permits a reduction in the number of collision, while efficiently maintaining the throughput of the airspace with adequate levels of security and reliability. On the other hand, the use of more advanced technologies tightens further the system parameters and accrues the verification

and safety problems, opening up the possibilities for the introduction of subtle errors. In these cases, a formal verification of the system safety is even more desirable.

3.2 The Traffic alert and Collision Avoidance System

Over the years, air traffic has continued to increase. The developments of modern air traffic control systems have made it possible to cope with this increase, while maintaining the necessary levels of flight safety. However, the risk of airborne collision remains. That is why the concept of an airborne collision avoidance system has been considered, giving rise to the initial development of the Traffic alert and Collision Avoidance System (TCAS) [23]. TCAS is a protocol used to monitor air traffic in the vicinity of flying aircraft. It provides the pilot with information about neighboring aircraft that may pose a collision threat and, also, it advises on how to resolve these conflicts. TCAS significantly improves flight safety. However, it can not entirely eliminate all collision risks. As in any predictive system, it might itself induce a risk of collision [20].

The TCAS system can enter in one of two levels of alertness [22]. In the first level, the system issues a Traffic Advisory (TA) signal to inform the pilot of a potential threat. When operating in this level of alertness, it does not provide any suggestions on how to resolve the situation. If the risk of collision increases, a Resolution Advisory (RA) signal is issued, and the system also suggests a maneuver that is likely to resolve the conflict. The TCAS system also allows for reversal commands and it may change the climb/descend advise during a conflict. This feature was added to the protocol in order to compensate for the nondeterminism in the pilot response. That is, the pilot may choose not to follow the TCAS advises thereby rendering the original maneuver unsafe. The TCAS detects this situation and changes the RA, if necessary. Obviously, this nondeterminism renders the necessity of a formal verification of the overall system safety even more necessary.

Aircraft separation standards vary according to circumstances. Above 29000 feet, when the aircraft are cruising at high speed, the standard is five miles of horizontal radar separation and 2000 feet of vertical separation. Below 29000 feet, the vertical separation is reduced to 1000 feet while the horizontal radar separation remains at five miles. When aircraft are moving at much slower speeds, as when they depart or approach an airport, the standard is three miles of horizontal radar separation and 1000 feet of vertical separation.

The TCAS protocol is designed to ensure collision avoidance between any two aircraft, with an approximation speed of up to 1200 knots and vertical rates as high as 10000 feet per minute. The controller action begins when the aircraft horizontal separation is 5 miles. The standard value for vertical climbs or descends is 10000 fpm. In emergencies it can be as high as 12000 fpm.

The TCAS system monitors the vertical plane, and only a few works, today, consider the parallel separation between the aircraft. In the future, the TCAS system might produce RAs for both the horizontal and the vertical planes [25, 9].

Aviation texts, usually, do not treat measures in a uniform way. For example, the horizontal speed may be given in miles per hour and the vertical rate may be measured in feet per minute. Or, the vertical distance may be presented in feet and time may be given in minutes or hours, while the horizontal distance is given in miles. See Figure 3. In this

work, however, all computed measures are converted to the standard metric system, so as to maintain a uniformity throughout. Hence, the horizontal and vertical distances are given in meters and the time is measured in seconds[2].

3.3 A Case Study with Two Aircraft

The emphasis of this work is on the verification and synthesis of some safety requirements of an air traffic management system. The aim is to validate the system operation, and to synthesize more appropriate and tighter values for some critical system parameters, in such a way to improve the system overall performance and to decrease the occurrence of unsafe situations. One of the main aspects of the modeling is to capture the cooperation between the system parts.

Figure 3 depicts a situation with two aircraft in the same airspace. The changes of

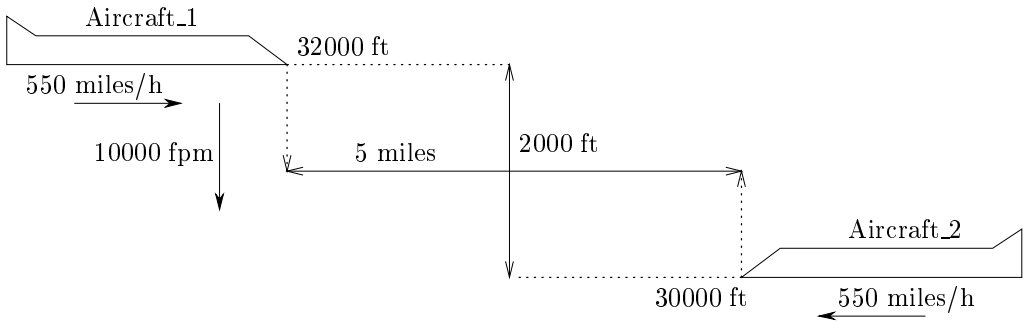


Figure 3: Two aircraft in the same airspace

attitude for the aircraft are provoked by the controller that monitors the vertical and horizontal separations between the aircraft. Based on the position between the two aircraft, the on-board TCAS controller may command the leftmost aircraft to climb or to descend. It may also increase its descent rate, if the situation is critical. The rightmost aircraft is assumed to remain in a leveled cruise. The controller may, however, reduce its horizontal speed in order to avert collisions.

In the scenario depicted in Figure 3, the leftmost aircraft is traveling at a height of 32000 feet and the rightmost one is traveling in the opposite direction, at a height of 30000 feet. The vertical distance between the aircraft is 2000 feet[3]. The aircraft are supposed to travel at 550 miles per hour, when cruising. The speed of 490 miles per hour[4] can also be applied in order to reduce the cruise speed. These rates and values are based on real data for Boeings 707 and Boeings 747[5].

[2] 1 foot $\simeq$ 0.3048 meters, 1 mile $\simeq$ 1829 meters and 1 knot $\simeq$ 0.47 meters per second.
[3] 32000 feet is about 9754 meters; 30000 feet is about 9144 meters; and 2000 feet is about 610 meters.
[4] 550 miles per hour $\simeq$ 280 meters per second; and 490 miles per hour $\simeq$ 250 meters per second.
[5] http://www.air-transport.org/public/Handbook/

The TA signal was suppressed from the models treated here and, when a conflict situation arises, a RA signal is issued directly. On the other hand, it is relatively easy to include more details of the TCAS protocols in the models, or to include more aircraft in the scenario. This would result in a more complex product automaton. The HyTech tool, however, was already operating close to its limits, running on a 350MHz desktop PC with 320MB of main memory. Certain internal limits of the tool, signaled by multiplication overflow errors, were also being reported. In order to accommodate a more complex and more complete model, it would be necessary to modify the source code of the tool and port it to a more powerful hardware platform.

4 Parameter Synthesis

The system modeled in this work comprises two aircraft flying in opposite directions, as depicted in Figure 3. The models capture several kinds of maneuvers that the aircraft can perform in the same airspace. The automaton model for the aircraft that is flying from left to right is more complex than the model for the other aircraft, which is flying in the opposite direction. This is because, at some instants, the first aircraft may decide to engage in some kind of specific maneuver, while the second aircraft is assumed to remain cruising en route. Clearly, in order to cope with such behavior, more operation modes are necessary to specify a model for the first aircraft. The full system model comprises three individual hybrid automata: one for each aircraft and another one for the system controller.

All the case studies reported here focus on the overall system safety. For each case, an analysis of the model is conducted and some parametric synthesis of critical values, such as the aircraft relative height and horizontal separation, is also performed.

In the next subsection, the hybrid automata models are presented. Each subsequent subsection describes a particular scenario that was studied.

4.1 The Hybrid Automata Models

The model for the leftmost aircraft is depicted in Figure 4, and the model for the rightmost aircraft is shown in Figure 5. Variable x_i indicates the horizontal distance and variable y_i indicates the vertical position of the aircraft. Variable k is used to extract information about the direction of flight, as will be explained later. The $x_1 = x_2$ horizontal mark is the position where both aircraft cross, and it is called the *critical point*. Note that the absolute position of the critical point varies, since it depends on the relative horizontal speed of both aircraft. Hence, the critical point is not always at the zero horizontal mark. A negative value for the horizontal position is measured to the left of the zero horizontal mark, while positive values indicate distances to the right of this point. At the beginning, both aircraft are symmetrically positioned at 6000 meters from the zero mark. The leftmost aircraft is cruising at 9750 meters and the rightmost one is cruising at 9140 meters.

The automaton for the leftmost aircraft, shown in Figure 4, starts off in the "Cruise_B" mode. From this mode, at the mark of 4500 meters, it enters in the "Descend" mode and

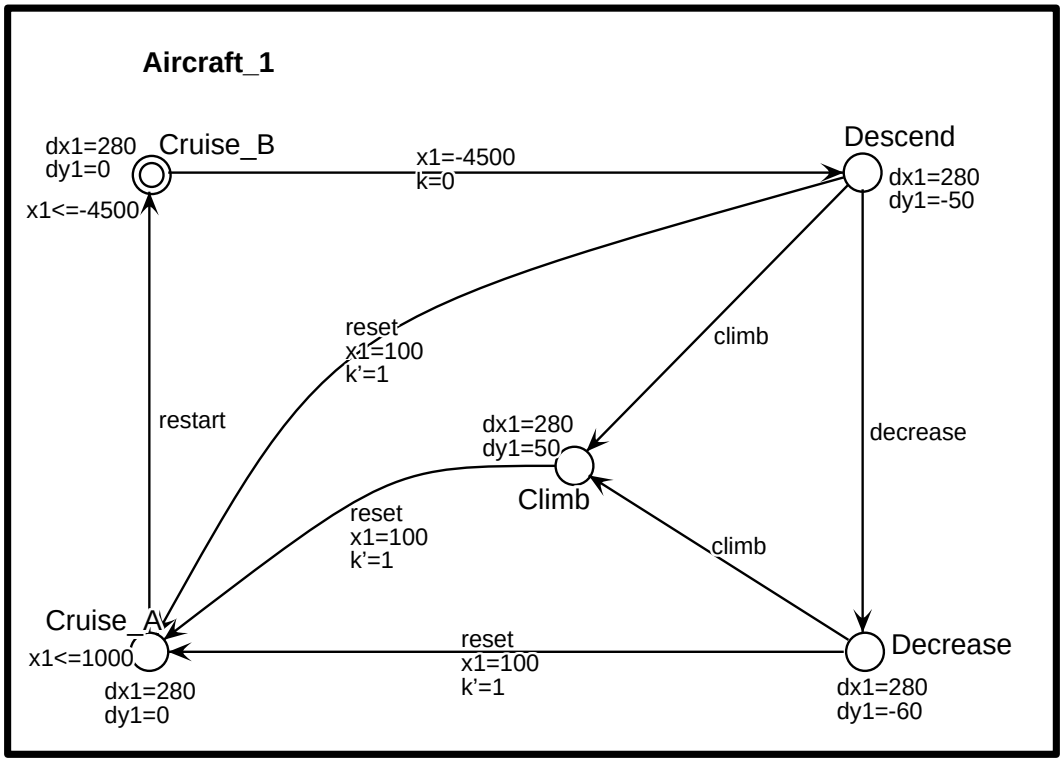


Figure 4: A model for the leftmost aircraft

starts to lose altitude at a rate of 50 meters per second[6]. From this point, a first nondeterministic alternative calls for a scenario of no interaction between both aircraft. This is captured by the transition that goes directly from the "Descend" mode to the "Cruise_A" mode. This transition happens at 100 meters from the critical point (indicating that both aircraft have already passed by each other) and it uses the "reset" event to synchronize with the models for the other aircraft and the controller. While in the "Descend" mode, the leftmost aircraft faces two other nondeterministic alternatives. It might receive a command to climb, moving into the "Climb" mode, synchronizing on the "climb" event, or it might receive a command to increase its descending rate, entering the "Decrease" mode and synchronizing on the "decrease" event. At the "Decrease" mode, there are two nondeterministic choices. Either, as a result of the interaction of both TCAS, the aircraft receives a counter-order to start climbing up, moving to the "Climb" mode, or it might fly by the second aircraft while still descending. In the first case, the leftmost aircraft will fly by the second aircraft while still climbing up. In any case, from the "Climb" mode or from the "Decrease" mode, a transition to the "Cruise_A" mode is taken, at a horizontal position of 100 meters, signaling that the fly by has already happened. In both cases, a "reset" event is issued, in order to synchronize this move with the model that describes the behavior of

[6] About 10000 feet per minute.

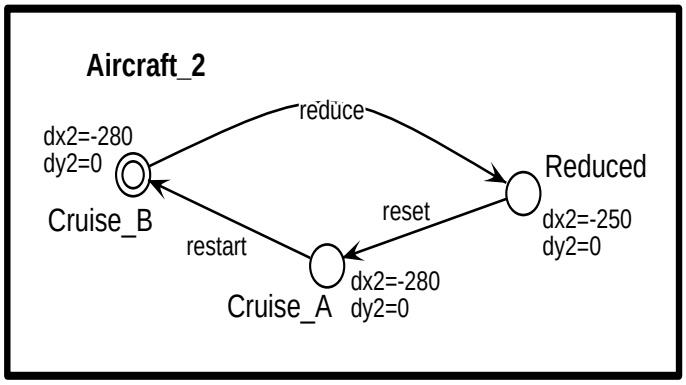


Figure 5: Aircraft_2 automaton

the rightmost aircraft. Finally, at the "Cruise_A" mode, the first aircraft assumes a leveled flight and returns to the start mode no later than when it passes by the 1000 meters mark, to the right of the critical point. The horizontal speed for this aircraft is kept constant at 280 meters per second[7].

The second aircraft is modeled by a simpler automaton, composed of three modes, as shown in the Figure 5. The automaton starts off in the "Cruise_B" mode, positioned at 6000 meters to the right of the zero horizontal mark and cruising at 9140 meters. Initially, the aircraft horizontal speed is 280 meters per second, traveling from right to left. From this mode the aircraft might, upon detecting the "reduce" event, move to the "Reduced" mode, where its horizontal speed decreases to 250 meters per second. The aircraft travels at this speed until it detects the "reset" event and synchronizes with the automaton that controls the first aircraft, moving to the "Cruise_A" mode. It remains at this mode until it reaches the critical point, when both aircraft cross each other. Next, the automaton returns to the start mode, upon detecting the "restart" event.

The model of the controller implements (a simplified version of) the TCAS protocol, and is illustrated in Figure 6. The automaton starts off in the "Normal" mode, and it may stay there until the horizontal separation between the two aircraft drops to 6000 meters. But earlier, at 7000 meters of horizontal separation, a nondeterministic choice occurs. Either the controller remains in the "Normal" mode, or it switches to the "Descend" mode. In the latter case, a synchronizing "decrease" event occurs if the vertical separation between the aircraft is at least 400 meters[8]. This event forces the first aircraft to increase its rate of descent.

If the nondeterministic choice is not exercised at 7000 meters, the controller continues in the "Normal" mode, monitoring the horizontal and vertical separation between both aircraft. When the horizontal distance drops to 6000 meters, and if the vertical separation

[7] About 550 miles per hour.
[8] About 1315 feet.

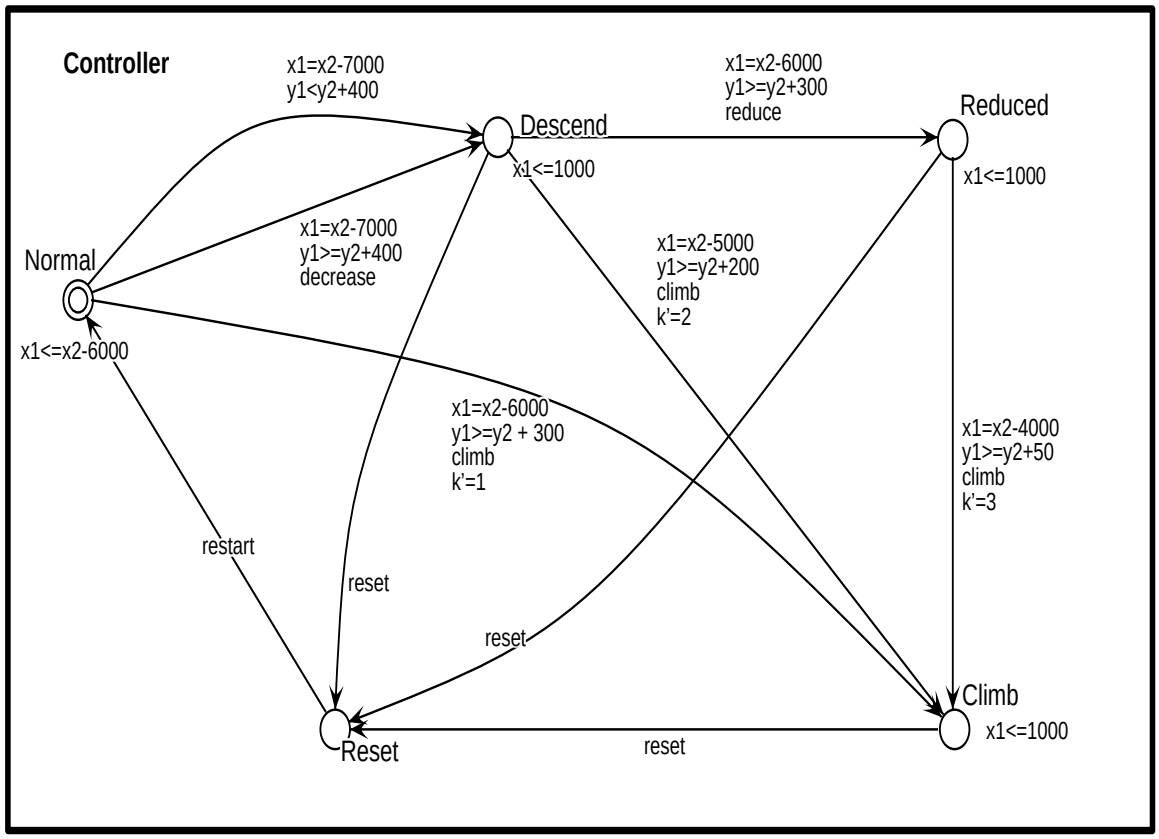


Figure 6: Controller automaton

is at most 300 meters[9], a "climb" event forces the first aircraft into a 50 meters per second[10] climbing trajectory, since it is not safe to continue the descent and cross the second aircraft route. As a result of the transition, the controller reaches the "Climb" mode.

If the nondeterministic choice is taken at the mark of 7000 meters, then the controller may stay in the "Descend" mode until the aircraft have crossed each other and the (originally) leftmost one has moved up to 1000 meters to the right of the zero point. In the "Descend" mode, one of three other nondeterministic choices will prevail. First, the controller may issue a "reset" event and move into the "Reset" mode, where it stays until the first aircraft has passed the zero mark by 100 meters. Or the controller might decide to act when the horizontal separation between both aircraft has reached the mark of 6000 meters, their vertical separation being at least 300 meters, or the controller might act when the horizontal separation between the aircraft has reached the mark of 5000 meters, with the vertical separation between them being at least 200 meters. In the latter case, a "climb" event is issued, forcing the first aircraft in an ascending trajectory, and moving the con-

[9] About 985 feet.
[10] About 10000 feet per minute.

troller to the "Climb" mode. In the first case, a "reduce" event slows down the rightmost aircraft and puts the controller into the "Reduced" mode.

After the controller reaches the "Reduced" mode a similar scenario evolves, with the controller monitoring the vertical and horizontal separation between both aircraft. The controller may stay in this mode until the aircraft have crossed and the first one has passed the zero point by 100 meters. Or, at a horizontal separation of 4000 meters, if the vertical separation between both aircraft is at least 50 meters, a "climb" event is issued and the controller passes to the "Climb" mode.

From the moment that the "Climb" mode is entered, the controller stays there until a "reset" event happens, when the first aircraft has passed the zero point by 100 meters. See also Figure 4. At this moment, the controller jumps to the "Reset" mode. In this mode, the controller waits until the aircraft have crossed and the (originally) leftmost one has traveled a distance not exceeding 1000 meters from the zero point. At the "Reset" mode, the automaton waits for the "restart" event and moves on to the "Cruse_B" mode, restarting the cycle.

The automaton that governs the overall system behavior is obtained by calculating the product of the three individual automata. The code for the automata, in the HyTech language, is listed in Appendix A.

4.2 Case Studies

The system behavior was captured and analyzed by running a number of different experiments. All case studies discussed in the sequel focus on establishing the safety of the system operation. To this end, several parametric synthesis were run. Experiments of this kind can reveal the limits of a safe system operation and can also be instrumental in obtaining tighter values for a number of system parameters.

All experiments were run using the HyTech computational support tool[11]. Figure 7 illustrates a typical input to the HyTech tool for a parametric analysis. The region of initial configurations, *init_reg*, indicates that:

1. *Aircraft_1* and *Aircraft_2* are in their respective start modes, "Cruise_B";
2. The *Controller* is in its start mode, "Normal";
3. The initial position for both aircraft are specified next, in meters[12];
4. The auxiliary variable k is zeroed. This variable indicates the mode of origin when the "Climb" mode is entered.

The particular parameter values used in this exercise are typical. They could easily be changed in order to reflect different values. An example of a final region, *final_reg*, capturing an unsafe region according to the TCAS protocol, is described next in Figure 7. It comprises five lines:

[11] http://www-cad.eecs.berkeley.edu/ tah/HyTech/

[12] 6000 meters is about 3.3 miles; 9750 meters is about 32000 feet; and 9140 meters is about 30000 feet.

```
var
     final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controller] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Decrease &
     loc[Aircraft_2] = Reduced &
     loc[Controller] = Reduced &
     x1=x2 &
     y1<=height;

reached:=

reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Figure 7: Height parametric analysis

1. *Aircraft_1* is in the "Decrease" mode, indicating a steep descent;
2. *Aircraft_2* is in the "Reduced" mode, indicating a reduced horizontal speed;
3. The *Controller* is in the "Reduced" mode, reflecting the two conditions above;
4. Both aircraft are at the same position, that is, they are passing by each other;
5. The last condition parameterizes the vertical distance of the leftmost aircraft.

The next line, in Figure 7, asks the HyTech tool to perform a forward analysis in order to compute the *reached* region. The last four lines specify a parametric print out of the region of points belonging to *reached* and *final_reg* regions. This intersection will contain all unsafe points for the system, since *final_reg* describes the region of unsafety. By negating the region calculated by the tool, it is possible to obtain the region of safe values for the *height* parameter.

All computational results were obtained by running the HyTech tool on a typical 350MHz Pentium II PC, with 320MB of main memory. Memory usage was never a problem when

running the experiments. Also, for each experiment, the time consumed to complete the analysis was always quite reasonable, in the range of a few seconds, with the *Post* operator converging after a few iterations. In contrast, any attempt to increase the number of aircraft caused both the forward and backward analysis to fail. These observations indicate that the HyTech tool was operating close to its limit, and a careful construction of the models had to be undertaken, always working at these limits.

In all experiments, command line options were used in order to avoid library overflow errors, caused by multiplication of large integers. The output message "`Will try hard to avoid library arithmetic overflow errors`", produced when running the HyTech tool, indicates the use of command line options to avoid such errors. Another point to be mentioned is that all backward analyses failed due, again, to multiplication overflow.

In the rest of this section, six case studies are presented, reporting on the synthesis of values for several critical parameters. More details about each of the experiments can be found in Appendix B.

4.3 The minimum height before the controller action

This study focuses on the minimum height reached by the first aircraft, in the worst scenario and before the controller starts to enforce the TCAS protocol. To capture this situation, the final region is specified by positioning the automaton for the first aircraft in the "Descend" mode, the automaton for the second aircraft in the "Cruise_B" mode, and the automaton for the controller in the "Normal" mode. Note that, because of the synchronizing "reset" event, it is not possible for the controller model to complete a full cycle before the aircraft models also restart at the initial mode. The input conditions can be read from the Appendix B, Section B.1.1, where the variable *height* indicates the parametric value for the vertical distance reached by the first aircraft. The output of the HyTech tool for this experiment is shown in Figure 8. The parametric analysis shows that the first aircraft can reach a minimum height of 9482 meters[13] before the controller starts to act. Observe that this vertical distance is still above the cruising altitude of the second aircraft. This indicates that the fly by is unsafe.

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 66375

Time spent = 0.63 sec total
```

Figure 8: Parametric analysis before the controller action

[13] About 31110 feet.

4.4 At critical point with increased descend

This case study investigates the safety condition at the critical point when the first aircraft has taken the decision to increase its descend, while the other aircraft is still unaffected. The situation is specified by letting the automaton for the first aircraft enter the "Decrease" mode, while keeping the second automaton in the initial "Cruise_B" mode. The automaton for the controller moves to the "Descend" mode, forced by the "decrease" event, and stays there. That is, the controller does not engage in climbing maneuvers, nor commands the second aircraft to reduce its cruising speed. See Figure 4 and 6. The extra condition specified by $x_1 = x_2$ guarantees that the final region is focusing on the critical point. The specification of this final region can be seen in Appendix B, Section B.2.1. Again, the minimum height reached by the first aircraft was synthesized using the *height* parametric variable.

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 61750

Time spent = 0.63 sec total
```

Figure 9: First aircraft increases descend

The HyTech tool synthesized the conditions that must be observed for this situation to occur. Figure 9 illustrates the output of the tool for this experiment. As can be seen, the minimum height value reached by the first aircraft is 8821 meters, or 28940 feet. From the point of view of the second aircraft, which is cruising at an altitude of 9140 meters, or 30000 feet, this value is inferior to the minimum acceptable distance of 2000 feet, as required by the protocol. However, from the point of view of the first aircraft, which is now below 29000 feet, the synthesized minimum value of 28940 feet is still within the acceptable interval of up 1000 feet for the vertical separation between both aircraft.

In the same situation, that is, with the three automata reaching the same final modes, another study was conducted. Note that the controller automaton passes from the "Normal" to the "Descend" mode when the horizontal distance between both aircraft is 7000 meters, and the vertical distance between them is at least 400 meters[14]. In this second exercise, the 400 meters separation was parameterized by the *diff_height* variable. This was accomplished by changing the condition $y_1 \geq y_2 + 400$ into $y_1 \geq y_2 + diff\_height$, in the controller automaton, shown in Figure 6. The HyTech tool, then, synthesized the maximum vertical separation bwtween both aircraft, at the moment when the first aircraft starts its increased descent, given that this change of behavior takes place exactly when the aircraft are 7000 meters apart. The result produced by the tool appears in Figure 10. It indicates that the maximum relative vertical distance between the aircraft, for this situation to occur, can be

[14] About 1310 feet.

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 4
     7diff_height = 3020   & 7height >= 61750

Time spent = 0.15 sec total
```

Figure 10: Parameterizing vertical distances

431 meters[15]. The minimum height reached by the first aircraft, at the crossing point, is still 8821 meters.

It is also as easy to parameterize the horizontal separation between both aircraft, while maintaining the minimum vertical separation at 400 meters between them, both measured at the point when the first aircraft starts to increase its descent. Figure 11 presents the output of the HyTech tool for this experiment, where the parameter *diff_horiz* indicates the desired minimum horizontal distance. The synthesis revealed that the command to increase

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 4
     diff_horiz = 6648   & 7height >= 61794

Time spent = 0.15 sec total
```

Figure 11: Parameterizing the horizontal and vertical distances

the descent should be issued no later than the point where the horizontal separation is 6648 meters. In this case, observe that the height reached by the first aircraft is now slightly higher, reaching 8827 meters[16], which is a little unsafer.

4.5 Reducing horizontal rate and no climbing

The next experiment will allow for the second aircraft to reduce its horizontal speed. For this to occur, the final region is specified as in the previous case, except that the automata for both the second aircraft and the controller are now allowed to synchronize on the "reduce" event, and proceed to reach the "Reduced" mode. For a full specification, see Appendix B, Section B.3.1.

The same three situations as in the previous subsection were analyzed here. First, Figure 12 presents the output of the HyTech tool when synthesizing the minimum vertical

[15] About 1414 feet.
[16] About 28960 feet.

distance for the first aircraft, without parameterizing the coordinate values where it starts its increased descent. The value obtained was 8785 meters[17], which is a slightly safer value

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     371height >= 3259250

Time spent = 0.62 sec total
```

Figure 12: Vertical distance with reduced horizontal rate

than the one revealed in the previous section, but the difference it is not significant. Or either, the reduction in the horizontal speed for the second aircraft must be more substantial in order to have a greater influence upon the system safety conditions.

Next, Figure 13 shows the output of the HyTech tool, for the same scenario, and when the 300 meters minimum vertical separation is also synthesized, using the parametric value *diff_height*. This variable measures the vertical separation between the aircraft, when the second aircraft receives the command to reduce its speed. The horizontal separation at this point is maintained at 6000 meters. The result shows that the vertical separation can be relaxed to 324 meters, a higher value.

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7diff_height <= 2270   & 371height >= 3259250

Time spent = 0.18 sec total
```

Figure 13: Vertical separation with reduced horizontal rate

In contrast, when trying to parameterize the minimum horizontal separation, the HyTech tool was unable to converge, due to library overflow errors caused by multiplication operations. Neither a backward nor a forward analysis could be completed, in this case.

4.6 At the critical point and climbing

Next, the situation where the first aircraft is ordered to climb is analyzed. In this initial case study, the first aircraft does not increase its descent, nor does the second aircraft reduces its cruising speed. The final region for the models is given by requiring the automaton for the first aircraft to reach the "Climb" mode directly from the "Descend" mode. The

[17] About 28822 feet.

automaton for the second aircraft remains in the initial "Cruise_B" mode, and the controller automaton moves directly from the "Normal" mode to the "Climb" mode. See Appendix B, Section B.4.1. All the evaluations are still being taken at the critical crossing point. Note that, here, the value $k = 1$ is also part of the critical region. This guarantees that the automata are following the transitions as specified. That is, the first aircraft is descending, but not at an increased rate, and the second aircraft is not reducing its horizontal speed. See also Figure 4 and 6.

Figure 14 illustrates the output of the HyTech tool when only the vertical height for the first aircraft is parameterized. The result shows that the maximum height reached at the

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 70125

Time spent = 0.66 sec total
```

Figure 14: Vertical distance while climbing

critical point, while the first aircraft is climbing, measures 10017 meters[18]. Note that this height is greater than the initial cruising height of 32000 feet. This shows that aborting the descent, in this case, could be premature. The next experiments provide tighter values for these parameters.

First, the vertical separation of 300 meters is parameterized, on the transition from the "Normal" mode to the "Climb" mode, in Figure 6. The parameter *diff_height* replaces the value 300 meters in the condition $y_1 \geq y_2 + 300$. The value of 342 meters is returned by the HyTech tool for the vertical separation parameter, as shown in Figure 15. Since the

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7diff_height <= 2395   & 7height >= 70125

Time spent = 0.18 sec total
```

Figure 15: Vertical separation while climbing

horizontal distance between both aircraft was maintained at 6000 meters, the maximum vertical distance reached by the first aircraft while climbing was still the same 10017 meters.

When parameterizing the horizontal separation between both aircraft, at the point where the climb command is issued, the results returned by the tool are as shown in Figure 16.

[18] About 32865 feet.

The vertical separation is maintained at 300 meters. As can be seen, now a linear relation-

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 8
     28height >= 5diff_horiz + 250500   & diff_horiz <= 9000
     & diff_horiz >= 5528

Time spent = 0.21 sec total
```

Figure 16: Horizontal separation and climbing

ship holds between the horizontal separation, when the maneuver is aborted by the climb command, and the vertical distance reached by the first aircraft, at the crossing point. The condition $diff\_horiz \leq 9000$ is trivially observed, given that the first aircraft starts to descend when the horizontal separation between them is exactly 9000 meters. The linear relationship is, then, reduced to

$$(28height \geq 5diff\_horiz + 250500) \wedge (diff\_horiz \geq 5528).$$

The last clause, $diff\_horiz \geq 5528$, indicates that the minimum horizontal separation between the aircraft can be reduced to 5528 meters. In that case, the other clause says that the maximum climbing point, for the first aircraft, can reach 9933 meters[19]. This shows that, even if the first aircraft goes into a descent and then the maneuver is aborted at its minimum possible horizontal separation distance, the first aircraft can still reach a point higher than its original cruising altitude. In the worst case, the vertical separation between both aircraft, at the crossing point, is greater than 32000 feet and, as a consequence, the maneuver is deemed a safe one.

4.7 Increasing the descent, then climbing

This is the same situation as in the previous subsection, except that the first aircraft has already engaged in a steeper descent route. Here, the final region is modified only by changing the condition on the k variable. It now reads $k = 2$, as can be seen from Appendix B, Section B.5.1. Note that the vertical separation at the point where the normal maneuver is interrupted by the climbing command is now at 200 meters, while the horizontal separation between both aircraft is set at 5000 meters, as dictated by the TCAS protocol.

The maximum height reached by the first aircraft, while climbing in this situation, is 9803 meters[20], as can be deduced from Figure 17. Observe, that this height is still greater than the initial cruising height of 32000 feet, but the difference is now smaller, when compared to the height reached in the maneuver studied in the previous experiment.

[19] About 32588 feet.
[20] About 32163 feet.

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 68625

Time spent = 0.66 sec total
```

Figure 17: Vertical distance after increasing descent and climbing

Parameterizing the vertical separation of 200 meters, produces the results depicted in Figure 18. The horizontal separation, in this case, remained at 5000 meters mark. The

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7diff_height <= 1520   & 7height >= 68625

Time spent = 0.18 sec total
```

Figure 18: Vertical separation with increased descent, followed by a climb

relaxed maximum value obtained was 217 meters, at the point where the climb command is issued.

As in the previous test cases, the horizontal separation between the aircraft, when the climb command is issued, was also synthesized. The vertical separation was maintained at 200 meters. Figure 19 illustrates the output of the HyTech tool for this case. The relation $\mathit{diff\_horiz} \leq 7000$ is trivially observed, since the first aircraft starts its descent already at the 7000 meters mark. Ignoring this clause, the conjunction computed by the tool reduces to the linear relationship

$$(56\mathit{height} \geq 11\mathit{diff\_horiz} + 494000) \wedge (\mathit{diff\_horiz} \geq 4840).$$

Hence, the maneuver can be aborted as late as when the horizontal separation between the aircraft reaches 4820 meters. Even in this extreme case, the height reached by the first aircraft, at the crossing point, is 9772 meters[21], still allowing for the minimum of 2000 feet required for a safe operation.

4.8 Increasing the descent, reducing the horizontal rate and aborting

In the last experiment, the first aircraft starts its descent, at a vertical rate of 50 meters per second, then it maneuvers to increase its downward vertical rate to 60 meters per second.

[21] About 32060 feet.

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     56height >= 11diff_horiz + 494000   & diff_horiz >= 4840
     & diff_horiz <= 7000

Time spent = 0.18 sec total
```

Figure 19: Horizontal separation and vertical distance with increased descent, followed by a climb

The second aircraft, sensing the presence of the other aircraft, reduces its horizontal rate from 280 meters per second to 250 meters per second. When both aircraft are 4000 meters apart, the maneuver is aborted by the controller, and the first aircraft receives a climb command. The final region shown by these conditions is described in Appendix B, Section B.6.1.

The HyTech tool computes that the first aircraft reaches a minimum height of 9615 meters[22] at the crossing point, as shown in Figure 20. Note that this value is approximately

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     371height >= 3567250

Time spent = 0.66 sec total
```

Figure 20: Parametric height analysis: aborting after descending and reducing

1500 feet above the cruising altitude of the second aircraft. By the TCAS protocol, this would configure an unsafe operation scenario. Note also that, as described in Subsection 4.5, if the maneuver is not aborted, the first aircraft will reach an altitude of 1200 feet below the cruising altitude of the second aircraft, which is, in this case, a safe altitude. This is because a minimum vertical separation of 2000 feet is specified when cruising at altitudes above 29000 feet, and this minimum is reduced to 1000 feet when cruising below 29000 feet. One alternative would be to use the HyTech tool and specify a final region where the vertical distance reached of the first aircraft, at the crossing point, was exactly 32000 feet, the minimum required for safety, and parameterize the horizontal separation required to reach that final region. This would yield the minimum safe value for that parameter.

In a final case study, the vertical separation between the two aircraft was also parameterized, at the moment when the "climb" event is issued, while the horizontal separation

[22] About 31545 feet.

was kept at 4000 meters. Figure 21 shows the output of the HyTech tool for this exercise. The output shows that the vertical separation can be at most 36310/37, or about 97 meters,

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     371diff_height <= 36310   & 371height >= 3567250

Time spent = 0.19 sec total
```

Figure 21: Parametric vertical separation analysis: aborting after descending and reducing

for the "climb" event to be issued. The vertical separation at the crossing point was still an unsafe 9615 meters, or about 31546 feet.

When, the HyTech tool was used to compute the relationship between the horizontal separation, at the point where the "climb" event occurs, and the vertical distance reached by the first aircraft at the crossing point, it was unable to complete the computation, due to the presence of multiplication library overflow errors.

5 Conclusions

The air traffic control protocol is a critical, real-time and reactive system. In addition, a possible system malfunction may cause enormous damages. The development of such systems demands the application of a rigorous validation and verification procedure. This work describes a step in this direction, using the software HyTech as (semi) automatic formal verification tool.

Hybrid automata is the mathematical approach used to construct the models for the various real system agents studied in this work. The hybrid automata formalism allows for such agents to manifest a continuous, dynamic behavior, regulated by the asynchronous occurrence of discrete events. An Air Traffic Management (ATM) system, when using the Traffic alert and Collision Avoidance System (TCAS) protocol, exhibit all these characteristics. The cruising aircraft comprise the cooperating continuous dynamic agents of the system. The discrete asynchronous events arise from the exchange of commands, issued by the TCAS protocol, when some participating aircraft engage in a collision avoiding maneuver.

The formal models developed in this work were used to synthesize some parameter values for the operational behavior of two aircraft, cruising in opposite directions. The aircraft were allowed to perform different maneuvers, under the guidance of the TCAS protocol. Using the hybrid automata models, built from the TCAS protocol description, the HyTech tool synthesized some critical values for the vertical distance, and the vertical and horizontal separation between the cruising aircraft. The values obtained indicated which distances offer a safe operational scenario and, in some cases, indicated tighter values for some of these parameters.

The HyTech tool also proved very easy to use, once the hybrid automata models were in place. Changing or adding new parametric variables was easily achieved from one case study to the next one. Describing new final regions, in order to capture different aspects of the system behavior, presented no difficulties either.

Although hindered, at times, when the complexity of the models reached the limits of the tool, most of the case studies were successfully planned and run on a typical desktop PC. More realistic cases studies, involving more aircraft and contemplating more complex maneuvers, could be attempted by adapting and extending the source code of the HyTech tool in order to make it run in larger and faster machines.

Related work, studying the safety of (parts of) a subway system, can be found in [8, 6, 7].

References

[1] R. Alur, T. Henzinger, and H. Wong-Toi. Symbolic analysis of hybrid systems. In *Proceedings of the 36th IEEE Conference on Decision and Control*, pages 702–707, 1997. Invited survey.

[2] Rajeev Alur, Thomas A. Henzinger, and Pei-Hsin Ho. Automatic symbolic verification of embedded systems. In *Proceedings of the 14th Annual IEEE Real-Time Systems Symposium*, pages 2–11. IEEE Computer Society Press, 1993.

[3] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL in 1995. *Lecture Notes in Computer Science*, 1055:111–114, 1996.

[4] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson, Wang Yi, and Carsten Weise. New generation UPPAAL. Technical report, BRICS, Dept. of Computer Science, Aalborg University, Denmark and Department of Computer Systems, Uppsala University, Sweden. This techincal report can be found at the URL address http://www.docs.uu.se/docs/rtmv/uppaal.

[5] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, and Wang Yi. UPPAAL — a tool suite for automatic verification of real–time systems. Technical Report RS-96-58, BRICS, Aalborg University, DENMARK and Department of Computer Systems, Uppsala University, Sweden, December 1996.

[6] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., and Jorge Rady Almeida Junior. Análise e Verificação de Segmentos de Via de uma Malha Metroviária. In *Proceedings of the II Workshop on Formal Methods*, pages 13–22, Florianópolis, Brazil, October 1999. (In Portuguese).

[7] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., and Jorge Rady Almeida Junior. Análise, Verificação e Síntese de Segmentos de Via de uma Malha Metroviária. Technical Report 18, Computing Institute, University of Campinas, Campinas, Brazil, August 1999. (In Portuguese).

[8] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., and Jorge Rady Almeida Junior. Formal Parameters Synthesis for Track Segments of the Subway Mesh. In *7th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems*, pages 263–272, Edinburgh, Scotland, April 2000. IEEE Computer Society Press.

[9] Brenda D. Carpenter and James K. Kuchar. Probability-based collision alerting logic for closely-spaced parallel approach. In *35th Aerospace Sciences Meeting & Exhibit*, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Crambridge, MA 02139, January 1997. Reno, NV.

[10] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *CACM*, 18(8):453–457, August 1975.

[11] Kathryn T. Heimerman. Air traffic control modeling. National Academy Press, The MITRE Corporation, 1998. As appears in the book entitled Frontiers of Engineering 1997.

[12] Thomas A. Henzinger and Pei-Hsin Ho. HyTech: The cornell hybrid technology tool. *Workshop on Hybrid Systems and Autonomous Control*, October 1994.

[13] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: the next generation. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 56–65, Pisa, Italy, 5-7, December 1995. IEEE Computer Society Press.

[14] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. A user guide to HyTech. In E. Brinksma, W.R. Cleaveland, K.G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS 95: Proceedings of the First Workshop on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1019 of *Lecture Notes in Computer Science*, pages 41–71. Springer-Verlag, 1995.

[15] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HYTECH: A model checker for hybrid systems. In O. Grumberg, editor, *CAV'97: Proceedings of the Ninth International Conference on Computer-Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 460–463. Springer-Verlag, 1997.

[16] Thomas A. Henzinger and Howard Wong-Toi. Using HyTech to synthesize control parameters for a steam boiler. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165 of *Lecture Notes in Control and Information Science*, pages 265–282. Springer-Verlag, 1996.

[17] Pei-Hsin Ho. *Automatic Analysis of Hybrid Systems.* PhD thesis, Cornell University, August 1995.

[18] Pei-Hsin Ho and Howard Wong-Toi. Automated analysis of an audio control protocol. In P. Wolper, editor, *Proceedings of the 7th International Conference On Computer*

Aided Verification, volume 939 of *Lecture Notes in Computer Science*, pages 381–394, Liege, Belgium, July 1995. Springer-Verlag.

[19] Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a NUTSHELL. *International Journal on Software Tools for Technology Transfer*, 1(1):134–152, December 1997.

[20] John Law. Acas ii programme. ACAS Programme Manager, January 1999.

[21] Michael D. Lemmon, Devin X. He, and Ivan Markvsky. Supervisory hybrid systems. *IEEE Control Systems*, pages 42–55, August 1999.

[22] John Lygeros and Nancy Lynch. On the formal verification of the tcas conflict resolution algorithms. Technical report, Laboratory of Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139, 1995.

[23] Phil Scott. Technology and business: Self-control in the skies. *Scientific American*, pages 24–55, January 2000.

[24] Claire Tomlin, George J. Pappas, and Shankar Satry. Conflict resolution for air traffic management: a study in multi-agent hybrid systems. In *IEEE Conference on Decision and Control*, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720, 1997.

[25] Lee F. Winder and James K. Kuchar. Evaluation of vertical collision avoidance maneuvers for parallel approach. In *AIAA Guidance, Naviation, and Control Conference*, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Crambridge, MA 02139, August 1998. Boston, MA.

A The System Automatons

```
var x1, x2, y1, y2   -- vertical and horizontal position of both aircraft
        : analog;
    k                -- it controls the taken climb
        : discrete;
    height, diff_height, diff_horiz   -- variable used to sinthesize
        : parameter;

-- ---------------------------------------------

automaton Aircraft_1
synclabs: decrease, climb, reset, restart;
initially Cruise_B;
loc Cruise_B: while x1<=-4500 wait { dy1=0, dx1=280 }
when k=0 & x1=-4500 goto Descend;
loc Decrease: while True wait { dy1=-60, dx1=280 }
when x1=100 sync reset do { k'=1 } goto Cruise_A;
when True sync climb goto Climb;
```

```
loc Descend: while True wait { dy1=-50, dx1=280 }
when x1=100 sync reset do { k'=1 } goto Cruise_A;
when True sync climb goto Climb;
when True sync decrease goto Decrease;
loc Climb: while True wait { dy1=50, dx1=280 }
when x1=100 sync reset do { k'=1 } goto Cruise_A;
loc Cruise_A: while x1<=1000 wait { dy1=0, dx1=280 }
when True sync restart goto Cruise_B;
end -- Aircraft_1

automaton Aircraft_2
synclabs: reduce, reset, restart;
initially Cruise_B;
loc Cruise_B: while True wait { dy2=0, dx2=-280 }
when True sync reduce goto Reduced;
loc Reduced: while True wait { dy2=0, dx2=-250 }
when True sync reset goto Cruise_A;
loc Cruise_A: while True wait { dy2=0, dx2=-280 }
when True sync restart goto Cruise_B;
end -- Aircraft_2

automaton Controller
synclabs: decrease, climb, reduce, reset, restart;
initially Normal;
loc Normal: while x1<=x2-6000 wait {  }
when y1<y2+400 & x1=x2-7000 goto Descend;
when y1>=y2 + 300 & x1=x2-6000 sync climb do { k'=1 } goto Climb;
when y1>=y2+400 & x1=x2-7000 sync decrease goto Descend;
loc Descend: while x1<=1000 wait {  }
when True sync reset goto Reset;
when y1>=y2+300 & x1=x2-6000 sync reduce goto Reduced;
when y1>=y2+200 & x1=x2-5000 sync climb do { k'=2 } goto Climb;
loc Climb: while x1<=1000 wait {  }
when True sync reset goto Reset;
loc Reduced: while x1<=1000 wait {  }
when True sync reset goto Reset;
when y1>=y2+50 & x1=x2-4000 sync climb do { k'=3 } goto Climb;
loc Reset: while True wait {  }
when True sync restart goto Normal;
end -- Controller
```

B Parameters Synthesis

B.1 The minimum height before the controller action

B.1.1 Analysis

```
var
   final_reg, init_reg, reached: region;
```

```
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Descend &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] =  Normal &
     y1<=height;

reached:=
         reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

B.1.2 Result

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 66375

Time spent = 0.63 sec total
```

B.2 At critical point with increased descend

B.2.1 Analysis

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Decrease &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] =  Descend &
     x1=x2 &
     y1<=height;

reached:=
         reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
```

```
          reached & final_reg
      endhide;
```

B.2.2 Result: first aircraft increases descend

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 61750

Time spent = 0.63 sec total
```

B.2.3 Result: parameterizing vertical separation and vertical distance

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 4
     7diff_height = 3020   & 7height >= 61750

Time spent = 0.15 sec total
```

B.2.4 Result: parameterizing the horizontal separation and vertical distance

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 4
     diff_horiz = 6648   & 7height >= 61794

Time spent = 0.15 sec total
```

B.3 Reducing horizontal rate and no climbing

B.3.1 Analysis

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Decrease &
     loc[Aircraft_2] = Reduced &
     loc[Controler] =  Reduced &
     x1=x2 &
     y1<=height;

reached:=
         reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
```

```
          reached & final_reg
      endhide;
```

B.3.2 Result: vertical distance with reduced horizontal rate

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     371height >= 3259250

Time spent = 0.62 sec total
```

B.3.3 Result: vertical distance and vertical separation with reduced horizontal rate

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7diff_height <= 2270   & 371height >= 3259250

Time spent = 0.18 sec total
```

B.3.4 Result: vertical distance and horizontal separation with reduced horizontal rate

```
     Library arithmetic overflow errors in mutiplication occurs.
```

B.4 At the critical point and climbing

B.4.1 Analysis

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Climb &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] =  Climb &
     x1=x2 &
     k=1 &
     y1<=height;

reached:=
         reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
```

```
          reached & final_reg
      endhide;
```

B.4.2 Result: vertical distance while climbing

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 70125

Time spent = 0.66 sec total
```

B.4.3 Result: vertical distance and vertical separation while climbing

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7diff_height <= 2395   & 7height >= 70125

Time spent = 0.18 sec total
```

B.4.4 Result: vertical distance and horizontal separation while climbing

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 8
     28height >= 5diff_horiz + 250500   & diff_horiz <= 9000
     & diff_horiz >= 5528

Time spent = 0.21 sec total
```

B.5 Increasing the descent, then climbing

B.5.1 Analysis

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Climb &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] =  Climb &
     x1=x2 &
     k=2 &
     y1<=height;

reached:=
         reach forward from init_reg endreach;
```

```
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

B.5.2 Result: vertical distance after increasing descent and climbing

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7height >= 68625

Time spent = 0.66 sec total
```

B.5.3 Result: vertical distance and vertical separation with increased descent, followed by a climb

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     7diff_height <= 1520   & 7height >= 68625

Time spent = 0.18 sec total
```

B.5.4 Result: vertical distance and horizontal separation with increased descent, followed by a climb

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     56height >= 11diff_horiz + 494000   & diff_horiz >= 4840
     & diff_horiz <= 7000

Time spent = 0.18 sec total
```

B.6 Aborting after increasing the descent and reducing the horizontal rate

B.6.1 Analysis

```
var
   final_reg, init_reg, reached: region;
init_reg:=
     loc[Aircraft_1] = Cruise_B &
     loc[Aircraft_2] = Cruise_B &
     loc[Controler] = Normal &
     x1=-6000 & x2=6000 & y1=9750 & y2=9140 & k=0;
final_reg :=
     loc[Aircraft_1] = Climb &
     loc[Aircraft_2] = Reduced &
     loc[Controler] =  Climb &
     x1=x2 &
```

```
     k=3 &
     y1<=height;

reached:=
         reach forward from init_reg endreach;

print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

B.6.2 Result: parametric height analysis aborting after descending and reducing

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     371height >= 3567250

Time spent = 0.66 sec total
```

B.6.3 Result: parametric vertical separation analysis aborting after descending and reducing

```
Will try hard to avoid library arithmetic overflow errors
Number of iterations required for reachability: 7
     371diff_height <= 36310   & 371height >= 3567250

Time spent = 0.19 sec total
```

B.6.4 Result: vertical distance and horizontal separation after increasing descent, reducing and climbing

```
     Library arithmetic overflow errors in mutiplication occurs.
```