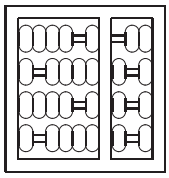


INSTITUTO DE COMPUTAÇÃO
UNIVERSIDADE ESTADUAL DE CAMPINAS

Formal Verification and Parameters Synthesis for Hybrid Real Systems

Adilson Luiz Bonifácio *Arnaldo Vieira Moura*

Technical Report - IC-05-04 - Relatório Técnico

March - 2005 - Março

Formal Verification and Parameters Synthesis for Hybrid Real Systems

Adilson Luiz Bonifácio* Arnaldo Vieira Moura†

Abstract

A hybrid system is given by a set of real-time reactive components, whose dynamic behavior results from the interplay of continuous and discrete events. Hybrid system have been modeled using formal methods and results have been obtained using the accompanying computational tools. Some such formal methods and tools are briefly presented, emphasizing their different characteristics. The Hybrid Automata formalism is one among such methods. A hybrid automaton is a finite state automaton where each state is extended to contain a description of a system dynamic profile. Transitions between states model a change in the system dynamics, triggered by discrete events. In this work, hybrid automata are used to model realistic hybrid systems stemming from segments of a subway mesh and parts of an air traffic control system. The models constructed are verified in this way using the support of computational tools. Moreover, some important model parameters are synthesized using the same tools. The verification certified that the systems operate safely. The synthesis sessions arrived at tighter values for some operational system parameters, while still guaranteeing a safe operation.

1 Introduction

Formal methods are being used with increasing emphasis in the modeling and verification of distributed systems, specially when hybrid components are part of the system, and when a system failure can cause severe or irreparable damage. In general, the time behavior of a hybrid system [HWKF02, Tra01] is obtained from a set of continuous dynamic profiles, controlled by the occurrence of discrete asynchronous events. These events signal a phase change in the system by substituting a continuous dynamic profile [HHWT95a, Ho95, BMCJAJ00a, BMCJAJ00b, BM00].

There are several approaches, based on finite state models, that can be followed when formally modeling hybrid systems. Statecharts [Har87], timed automata [AD94] and hybrid automata [AHH93] are among them. A common problem faced by these techniques, when implemented as computational aids, is the state space explosion problem. When several smaller component specifications are grouped together in order to model a larger system, the system state space may grow abruptly, impairing the performance of tools that visit all states when verifying system properties, such as when some form of model checking is

*Computing Department, University of Londrina, 86051-900, Londrina, PR
†Institute of Computing, University of Campinas, 13081-970 Campinas, SP

used. Special techniques [Bry92, CES86] were developed to help circumvent such problems. Some of these techniques gave rise to computational aids, as is exemplified by the Masaccio model [Hen00], that treats hybrid automata specifications.

This work focuses on hybrid automata [AHH93, HHWT95a, HHWT97, Ho95], as the vehicle for formally specifying and verifying real reactive systems. A hybrid automaton is, basically, a finite state machine in which a state contains a dynamic profile that describes a continuous time evolution for the system being modeled. Transitions between states, triggered by the occurrence of discrete events, model a change in the continuous profiles that describe the system after the transition takes place. Each system component is modeled by an independent hybrid automaton and the system behavior then being given by the product automaton [HHWT97, Ho95]. Synchronism across the system is obtained by the exchange of messages among the component automata [HHWT97, Ho95]. Hybrid automata are useful to model systems composed by a number of smaller components that can be described by a set of simple dynamic behaviors. This is in contrast to classical control theory where single systems with a much more complex dynamic behavior are studied.

As case studies, two real world systems are used: part of a subway mesh and part of an air traffic management system. Both systems are relatively complex ones, but still could be treated by the computational aids, when running close to their limits of memory and execution time. The subway mesh shows a simple structure, but with several subcomponents and sensors which induced an intense message exchange among sub-models, each sub-model capturing different aspect of the real system. The results obtained contributed to the validation of the overall system operation. The Traffic Alert and Collision Avoidance System (TCAS) is, essentially, an on-board algorithm responsible for detecting possible collision courses between two aircraft, while proposing possible collision avoidance maneuvers. The system modeled was comprised by two aircraft moving in opposite directions, thus capturing a fraction of the complexity of a full real system. The model was used to certify that the simplified collision detection and avoidance protocol was safe. In both cases, a linear hybrid approximation was defined and, for both model specifications and system verifications, the HyTech tool [HH94] was used.

The rest of this paper is organized as follows. Section 2 comments about some related techniques and tools in more details. The fundamental notions and definitions of hybrid automata are presented in Section 3, with some examples. Section 4 describes the subway system and the air traffic protocol in more details. Section 5 presents the hybrid automata models for these systems. Results are discussed in Section 6. The last section closes with some concluding remarks.

2 Related Works

This section briefly presents other approaches and tools, somewhat related to this work, used to model and verify hybrid systems stemming, primarily, from railway control problems and from air traffic management problems.

2.1 Techniques

Many of the models give rise to some form of a state transition graph to represent the operation modes of the system and the possible transitions between such modes [BCMD90, CMMC95, CEG$^+$01, CMM$^+$94, CES86]. The properties of interest, usually, are written using some form of temporal logical formalism, e.g. CTL, ITL or LTL [BCMD90, CMMC95]. The verification algorithm then visits the reachable states, checking which ones satisfy the desired properties. In order to cope with the state explosion problem, many implementations use binary decision diagrams (BDDs) [Bry92]. BDDs use an internal representation for logical formulas that favors the elimination of redundancies and the sharing of common substructures. Model checking and BDDs have been used with success in the specification and verification of some complex systems [CMM$^+$94, CES86].

Timed automata have been used to model and verify real-time hybrid systems. Using state reachability analysis and model checking the value range of some system parameters were narrowed. One limitation, in this case, was related to the timed automata formalism, which offers only perfect clocks and chronometers, and does not allow for a more elaborate description of dynamic behavior.

Hybrid automata [HH94, HHWT97, Ho95] is another formalism designed to model hybrid systems. This is the formalism used in this work and it is described in more details in the next section.

More recently, Masaccio [Hen00] has been proposed as another formal notation to model embedded hybrid systems. The approach is similar to hybrid automata, where the global system behavior being obtained by the cooperation of independent distributed components, which can describe both discrete and dynamic system behavior. New components can be created by serial or parallel composition, and can further be nested into a hierarchical design. This hierarchical structure is not present in Hybrid Automata model, which do not permit the nesting of structures. In these formalisms, discrete basic components, which model state transitions, and dynamic components, which model time continuous evolution, can be constructed using six basic operations of parallel and serial composition, variable and mode renaming, and variable and mode abstraction [Hen00].

2.2 Some Case Studies

A Mealy machine formalism has been used to model and verify the interlocking mechanism in railway systems [HPS01]. Both the hardware units and the communication protocol between units were modeled. The formal languages Lustre [HR99] and Mocha [AdAG$^+$01] were used to describe the system specifications. The verification addressed some security properties involving the correctness of the communication protocol. The modeling and the computational tool used did not allow for the synthesis of system parameters, in order to better tune the algorithm.

The RAISE formalism [Gro95] has also been used to specify and verify some control aspects of railway systems. The RAISE methodology uses a predicate language and algebraic specifications in order to cope with some real time system requirements. Dynamic properties of the system can then be verified by the RAISE tool. One of the advantages of the

formalism is that the state explosion problem is kept under control, to some extent. In this way, the model can deal with several compositions and many track switching mechanism, permitting a more realistic system to be tested. As with the Mealy formalism alluded to above, the RAISE methodology also does not allow for the synthesis of system parameters.

Another model for a railway interlocking mechanism has been constructed using the formalism of Communication Sequential Processes (CSP) [Win02]. CSP describes system interactions as intercommunicating processes. The global system synchronization is attained by the interchange of values through the system communication channels naturally modeled in the CSP formalism. The language used also supports the analysis of system security properties [Win02], by verifying that all reachable states satisfy them. The computational tool Failure Divergence Refinement (FDR) [Ros95] was able to verify the communication protocol, attesting that some security failures, such as derailment and collisions, are avoided. Again, parameter synthesis was not considered. Also, the model has no graphical representation that can serve as an aid when manipulating and analyzing the models.

The Short Term Conflict Alert (STCA) protocol, which is part of the Traffic Alert and Collision Avoidance System (TCAS), was designed to alert aircraft controllers for the possibility of air collisions. It has been studied [MKN$^+$00] using a discrete model that captures its behavior by means of a set of processes that communicate by event exchange. The overall system behavior was obtained by the composition of these individual processes. Moreover, the dynamic model of the system was extended using logical functions over continuous variables, in order to compose a hybrid model.

Models have been constructed using the timed automata formalism [AD94]. At Honeywell labs [CEG$^+$01] some hardware controllers were modeled using this formalism. The Label Transition System Analyzer (LTSA) [Mag99] is tool that offers a graphical animator for the clock variables in the timed automata notation. The input to the tool is a specification is written in the FSP formalism [MKN$^+$00]. Some parameter synthesis, such as for speed and altitude, were realized when this combination was used to model some aspects of an air traffic management system [MKN$^+$00].

Other works, more similar to this work, have been able to verify the conflict resolution properties of the TCAS protocols using the HyTech tool [MF01]. In that case, the SPIN tool was also used to describe non-timed models in the Promela language. The results were limited due to the state explosion problem, and similar results have already been obtained when both aircraft travel in the same airspace [BM00]. In any case, the state explosion problem has proved to be the most serious issue when using the HyTech tool to verify hybrid systems.

3 Hybrid Automata

Hybrid Automata are a general formalism, which has been proposed to model dynamical systems. A *hybrid automaton* [ACH$^+$94, AHWT97, AHH93, BM00, BMCJAJ00a, BMCJAJ00b, Ho95] is a system

$$H = (X, V, \mathit{flow}, \mathit{inv}, \mathit{init}, E, \mathit{jump}, \Sigma, \mathit{syn}),$$

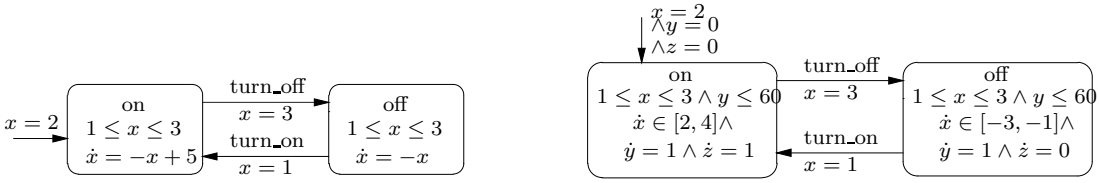
where:

1. $X = \{x_1, x_2, \ldots, x_n\}$ is a finite set of *variables*, where n is the *dimension* of H.
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 $X \cup \dot{X}$, where $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$[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 be always 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 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 $X \cup X'$, where $X' = \{x'_1, \ldots, x'_n\}$. The primed variable x' is used to indicate the (new) value of the corresponding variable x after a transition is taken.
8. Σ is a finite set of *event labels*, or just *events*.
9. The partial function *syn* maps transitions in E into events of Σ, and is used to synchronize transitions, when appropriate.

3.1 An example: a gas heater model

This model captures the simple dynamics of a gas heater [HHWT97], at shown in Figure 1(a). The real variable x, which evolves deterministically in time, is used to model the temperature of the container. The heater is turned off when x reaches 3 units, and it is turned back on when x falls to 1 unit. In this example, $n = 1$, $X = \{x\}$, and that $V = \{on, off\}$. The time evolution of x is computed from the initial conditions, the differential equations present in each mode and from the transitions between modes. 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 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 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 usual condition $x' = x$ not being shown in the figure. In this model, the value of the variables do not change when a transition is taken. It is also common to use guards, in the sense of [Dij75], to represent

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



(a) Gas heater

(b) Relaxation and Conditions

Figure 1: An example of hybrid automata

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 $x_1 = x_2$ to be enforced before the change, and the postcondition $x_1' = 2x_2$, that must hold after the 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 allow the synchronization between distributed component automata, as described in the sequel.

Given the internal requirements for mode *on*, in the example depicted in Figure 1(a), it is easy to verify that the predicate $2 \leq \dot{x} \leq 4$ is always satisfied. Similarly, it is easy to see that the condition $\dot{x} \in [-3, -1]$ is always satisfied in mode *off*. Using these facts, a non-deterministic relaxation of the previous model can be constructed, as shown in Figure 1(b). The latter model also uses two extra variables, y and z. Variable y is a perfect clock its time derivative being one. Variable z is a perfect stopwatch, since its time derivative is either zero (clock stopped) or one (clock running). These two extra variables can be used to describe external safety conditions, such as $(y \leq 60 \implies z \leq 40)$, stating that "during the first hour of operation, the heater shall not be on for more than 40 minutes" [HHWT97].

3.2 The Product Automaton

The model of a complex system comprises several individual component automata that operate concurrently. The synchronism of the global system is obtained by requiring simultaneity of phase transitions when they are labeled by the same discrete event, and using shared variables. The synchronism is automatically incorporated into the product automaton, the latter being defined as follows. Let H_1 and H_2 be hybrid automata. The *product automaton* of H_1 and H_2, indicated by $H_1||H_2$, is a system

$$H = (X_1 \cup X_2, V_1 \times V_2, flow, inv, init, E, jump, \Sigma_1 \cup \Sigma_2, syn),$$

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

1. $v_1 = v_1'$, $e_1 \notin E_1$, $e_2 = (v_2, v_2') \in E_2$ 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. $v_2 = v_2'$, $e_2 \notin E_2$, $e_1 = (v_1, v_1') \in E_1$ 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. $e_1 = (v_1, v_1') \in E_1$, $e_2 = (v_2, v_2') \in E_2$ and $\mathit{syn}_1(e_1) = \mathit{syn}_2(e_2)$. 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 the individual automata, in turn.

3.3 Regions and Trajectories

An *atomic linear predicate* is a closed 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 linear hybrid automaton satisfies the following restrictions:

(i) For every operation mode $v \in V$, the conditions *inv(v)* and *init(v)* are convex linear predicates.

(ii) For every operation mode $v \in V$, the continuous activity condition *flow(v)* is a convex linear predicate over the set $\dot{X}$ only.

(iii) For every transition $e \in E$, the phase change condition *jump(e)* is a convex linear predicate.

The last 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 rational constants ϵ_1 and ϵ_2.

A *convex region* of a hybrid automaton of dimension n is a convex polyhedron in $\mathbb{R}^n$. A *region* is a finite union of convex regions [HPP94]. 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 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 $a \in [inv(v)]$. The configuration (v, a) is *initial* if $a \in [init(v)]$. In the example depicted on Figure 1(a), configuration $(on, 0.5)$ is not an admissible one, while $(on, 2)$ is an initial configuration.

Let $q = (v, a)$ and $q' = (v', a')$ be two configurations of a hybrid automaton. The pair (q, q') is a *phase change* if $e = (v, v')$ is a transition in E and $(a, a') \in [jump(e)]$. The pair (q, q') is a *continuous activity* if $v = v'$, if there is a nonnegative real $\delta \in \mathbb{R}$ (the duration of the continuous activity) and if there is 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 hybrid automaton H 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 H. Such a trajectory is said to *start* at q_0. A configuration q' of H is *reachable from* a configuration q if q' is the last configuration of some trajectory of H starting at q. An *initial trajectory* of H is any trajectory of H starting at some initial configuration of H. A configuration q' of H is simply *reachable* if it is reachable from a initial configuration of H. 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 H. 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 φ_k contains all the configurations of H that are reachable via some trajectory that starts from a configuration in φ_0. When the process converges, 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.

The following theorem can be shown to hold [AHH93]: if H is a linear hybrid automaton and if $[\varphi]$ is any region of H, 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.

3.4 Safety Verification and Parameter Synthesis

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 H, 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 requirements 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 equation $\dot{\alpha} = 0$ in all operation modes and, further, by imposing the condition $\alpha' = \alpha$ over all 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 [HHWT97]. It follows that the original model is also safe with respect to the same requirements.

3.5 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 [HH94, HHWT95a, HHWT95b, HHWT97] 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 [HHWT97]. HyTech is one of the few 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 [AHH93, HH94, HHWT97, HHWT95b].

UPPAAL [LPY97, BLL+98, BLL+96b, BLL+96a, LPY01] 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. A number of other related works, dealing with hybrid systems, can be found in [HWT95, HWT96, Ho95].

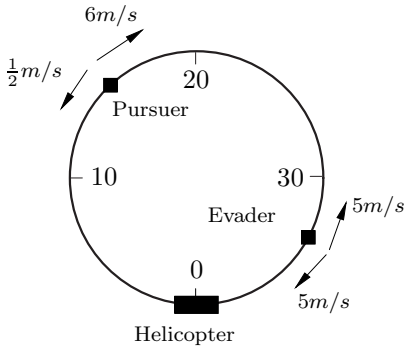


Figure 2: Pursuit game scenario

3.6 Another example: the pursuit game

As another example of hybrid automata, consider Figure 2. The scenario represents a two-person game between a pursuer and an evader. Both players are running on a 40 meters long circular track. The pursuer can travel in the clockwise direction with a speed of up to 6 meters per second, or in a counterclockwise direction, going at up to 0.5 meters per second. The evader is on a bicycle and travels at up to 5 meters per second in either direction. The pursuer can change direction instantaneously at any time. The evader, however, makes a decision whether to change its direction only at fixed points in time, separated by exactly two seconds. The goal of the evader is to avoid the pursuer. In order to achieve that, the evader must reach a rescue helicopter located at position zero on the track.

The game can be modeled by the hybrid automaton depicted in Figure 3. The variable p gives the position of the pursuer on the track, relative to position 0. Similarly, the variable e models the position of the evader. The clock variable t measures the time elapsed between consecutive choices of the evader. There are three operation modes, according the movement direction and the evader position: going clockwise, modeled by the mode labeled *clockwise*, going counterclockwise, modeled by the mode labeled *counter* and a rescued mode, labeled *rescued*. The evader evolves clockwisely according to the differential equations $\dot{e} = 5$ when it is going clockwise and it obeys the relation $\dot{e} = -5$ when it is going counterclockwise. The pursuer speed is non-deterministically given by a value in the interval $[-1/2, 6]$. This non-determinism models the fact that the pursuer can change direction instantly. The invariant conditions $0 \leq e$, $p \leq 40$ and the phase change condition appearing as self-loops, in the modes *clockwise* and *counter*, reflect the fact that the track is circular and 40 meters long. It is common to use guards [Dij75] to represent the phase change conditions. For example, the guard $(p = 40) \rightarrow (p' = 0)$ means phase change condition is given by predicate $p = 40$, which the variable p will assume the value 0 after the transition.

The evader's strategy is captured by the four transitions between the operation modes. These transitions implement a simple strategy. The evader compares the time necessary for both players to reach the helicopter, assuming movement in clockwise direction. The evader keeps moving in the clockwise direction if its time is less than the time computed

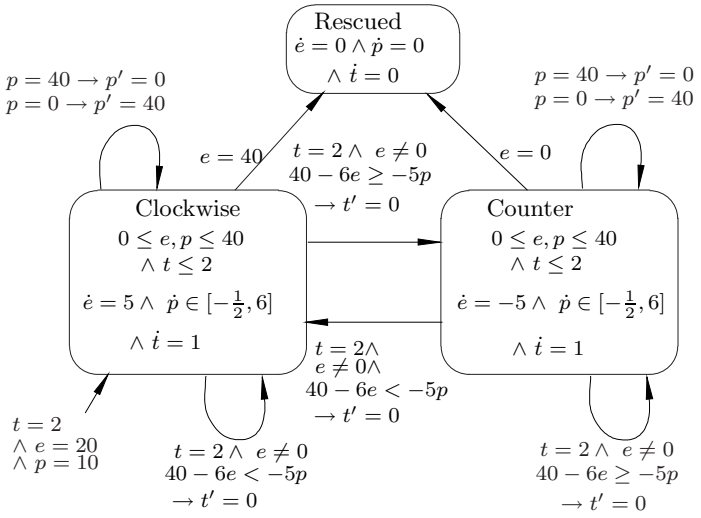


Figure 3: Pursuit game automaton

for the pursuer, else the evader inverts its direction.

The mode *rescued* signals that the evader reaches the helicopter. Therefore, the transition for this mode is triggered as soon as the evader reaches position 40 or position 0, when going in the clockwise or counterclockwise direction, respectively. When the evader reaches the helicopter the speeds of the evader and of the pursuer, as well as the clock variation, are zeroed, interrupting the game.

The evader is initially at position 20 directly opposite the helicopter, and the pursuer is between both, at position 10. From this initial situation, the evolution of the hybrid automaton would disclose that the evader would reach the helicopter before the pursuer could catch him. Therefore, the evader's strategy is efficient, in this case.

4 Case Studies: system descriptions

In this section the two real hybrid systems used as case studies are described. The next section presents the models constructed for each of them.

4.1 A Subway Control System

Subway systems are complex distributed systems that function under tight operational conditions and under great demand. Needless to say, any safety fault can result in intolerably high damages, both in terms of properties and lives. Hence, the need arises for a formal, rigorous, verification of all the system safety requirements. Part of the subway traffic is controlled using signaling electronic systems [FAM98], implemented on microprocessors. Recent technological advances in the area of micro-processed circuits make possible the use of more sophisticated functions in the control and supervision of the system behavior.

This more advanced technology permits a reduction of the interval between consecutive trains, thereby increasing the system capacity, while still maintaining adequate levels of security, promptness and reliability, and within an adequate cost. The use of more advanced technologies tightens further the system parameters and accrues the verification and safety problems, opening up the possibilities for introducing subtle errors. In these cases, a formal verification of the system safety is the more desirable [CJAJ98].

In this work, analysis and verification of safety aspects of a subway mesh are performed. The aim is, on the one hand, to model the system and to verify different properties of the model, validating its operation. On the other hand, the work here described also focuses on the synthesis of more appropriate and tighter values for some critical system parameters, in such a way as to improve the system overall performance. One of the main aspects of the modeling is to capture how and when the cooperation between the system parts takes place.

The functioning and the operation of the relevant essential features of a subway mesh that are necessary to understand the modeling phase are described next:

1. Track Circuits (tcs): these are segments where a train can be detected. In a track circuit, speed codes are transmitted to the trains. The enforcement of such codes guarantees the safety of the trains, given their positions in the track, and in accordance with their relative position with respect to other trains.

2. Switch Machines (SMs): switch machines are located on the tracks to permit trains to change tracks. The machines must be correctly positioned and locked, when a train passes through it. SMs are operated electrically, or manually in emergency conditions, and can be in one of two states: (i) normal; or (ii) reverse. In each state, the machine can be either electrically locked or electrically unlocked.

3. Terminal Zones: it is a yard comprising several track circuits, which are used for maneuvering.

4. Interlock Regions: these are segments where the SMs are found. They are also called "SMs regions".

5. Micro-processed Interlock: this equipment communicates with the SMs and the track circuits. It is part of the automatic control of the global system and it automatically supervises the safe movement of the trains.

Some functions of the subsystem that controls the movement of the trains are: (i) selection of the interlock control mode; (ii) issue commands to the SMs; (iii) impose speed restrictions; and (v) issue commands to reverse direction. Reversing direction allows a train to switch direction at a terminal zone and also allows for the rescue of blocked trains.

There are some basic safety requirements to prevent train collisions and derailments. These situations can occur, for example, by the wrong movement of a SM. Some safety conditions are: (i) occupy a tc only if there is no traffic in the opposite direction; (ii) occupy a tc only if it is empty; (iii) indicate the release of a tc when the train abandons it;

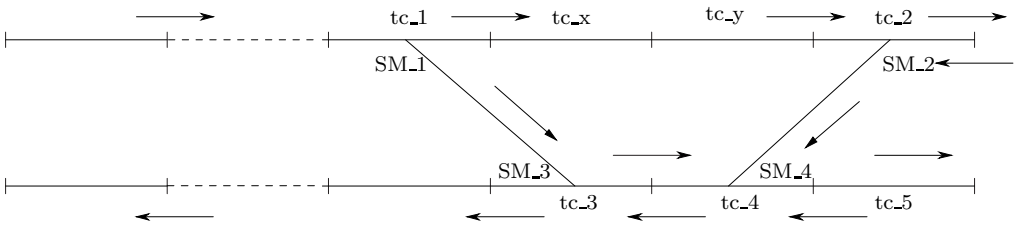


Figure 4: Tracks in a maneuvering yard

(iv) unlock a *SM* only when the corresponding *tc* is empty; (v) enforce the speed profiles; and (vi) reverse the traffic direction only when the *tc* is empty.

In the real system there can be several trains in the subway mesh. A map of a terminal track section is depicted in Figure 4, and it includes *tc*s and *SM*s, where each *tc* is about 200 meters in length. The occupation of a *tc* is provoked by a request from some train that needs to pass though the *tc* in order to arrive at its destination. When the train leaves the *tc*, the latter is released and it is, thus, ready to be occupied again by some train traveling on the same tracks. If the occupation is granted, adequate speed profiles are then imposed using suitable transmitters attached to the *tc*. The basic speed profiles, in kilometers per hour, for the system are: 0, 30, 60, 80 and 100. There is a predefined sequence of default speed profiles established along each track. During the system operation, the speed code of 0 km/h is applied to any circuit that cannot be occupied. Also, a circuit that contains a *SM* carries a standard speed profile of 30 km/h, and track circuits farther away from interlock regions can have speed profiles of up to 100 km/h. Moreover, the occupation of any particular *tc* forces a particular speed profile on the *tc*s that fall behind it. These *tc*s have a speed limit imposed upon them in agreement to their distances to the base *tc*. This scheme, called a *shadow*, is a safety condition imposed on the system operation to avert collisions.

When a train occupies a *tc* that contains a *SM*, the train must have, previously, requested the movement of the *SM* to the required position, either to *normal* or to *reverse*. After the train signals the positioning of the *SM*, the latter unlocks and starts to move to the required position. This takes some time, until the needle point locks in the correct place and the engine subsequently disengages. Enough time has to be allowed for this movement to complete, before the train is about to enter the *tc* where the *SM* is located.

4.2 An Air Traffic Management System

Air Traffic Management (ATM) systems 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 lives. Several faults are known to have occurred in aviation history, causing terrible accidents, due the unsafe operation of these systems [Sco00].

In a typical flight, after the aircraft enter en route, the Air Traffic Control (ATC) monitors the flight by radio [Hei98]. After an ATC acknowledges radar contact with the signaling aircraft, 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 conventional airline aircraft.

General aviation aircraft are allowed to fly under Visual Flight Rules (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, pilots are responsible for maintaining adequate separation from other aircraft. That is why these rules are sometimes called the "see and be seen" rules.

Instrument Flight Rules (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 an airspace sector [TPS97]. 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. However, 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.

Developments of modern control systems have coped with increasing the air traffic, while maintaining the necessary levels of flight safety. Since risks of airborne collision remains, 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) [Sco00]. 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, but it can not entirely eliminate all collision risks. As in any predictive system, it might itself induce a risk of collision [Law99].

The TCAS can enter in one of two levels of alertness [LL95]. 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

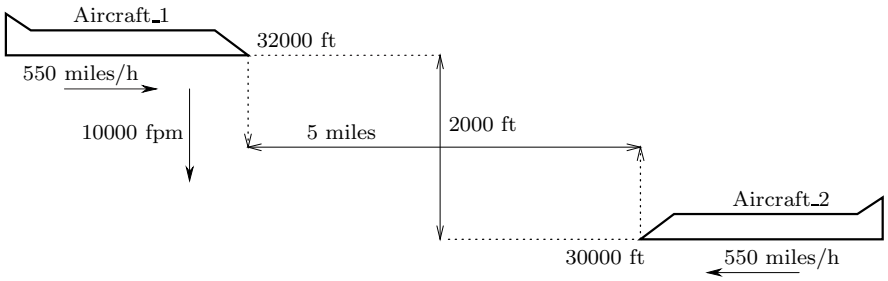


Figure 5: Two aircraft in the same airspace

also allows for the presence of reversal commands that may change the climb or 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 (fpm). 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 system monitors the vertical and horizontal planes. This work considers RAs only for the vertical plane [WK98].

The emphasis here 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 as to improve the system overall performance and to decrease the occurrence of unsafe situations. Other main aspect of the modeling is to capture the cooperation between the system parts.

Figure 5 depicts a situation with two aircraft in the same airspace. The changes of 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 ascent 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 5, the leftmost aircraft is traveling at a height of 32000

feet and the rightmost one is traveling at a height of 30000 feet, in the opposite direction. The vertical distance between the aircraft is 2000 feet[2]. The aircraft are supposed to travel at 550 miles per hour, when cruising. The speed of 490 miles per hour can also be applied in order to reduce the cruise speed[3]. These rates and values are based on real data for the Boeing 747 family[4].

5 Case Studies: system models

In this section we present the hybrid automata models for the two case studies.

5.1 The Hybrid Automata Models for the Subway Mesh

For safety reasons, usually, in a subway mesh it is required that the positioning and locking of any *SM* before the train invades the *tc* where the *SM* is located. In this work, however, the model deviates from this notion. Here, the system dynamic allows a train to enter the *tc* before the *SM* is locked.

The system modeled comprises the terminal track yard depicted in Figure 4. There are two trains traveling from left to right on the upper tracks and reversing direction on the lower tracks. The full system model shows nine individual hybrid automata: one for each train; one for each *SM*; and five other automata, one for each one of the five *tc*s shown.

The two train automata are the more complex of the nine automata. Each one has eight states that represent the various situations that arise when the train passes through the yard. One of the train models is shown in Figure 6. Variable aux_1 indicates the distance of the trains inside of the currently occupied *tc*, counting from the *tc* entrance point. Variables k and w are used to enforce the occupation rules. The automaton starts in the *Get* state. From this state, it moves to one of the *Taking* states, depending on the value of the k variable. This variable identifies which of the *tc*s is being occupied next. The two *Profile* states distinguish between the two speed codes imposed on the two *SM* regions. A positive speed indicates movement from left to right and a negative one signals movement in the opposite direction. When a *tc* is released, the *Get* state is reentered, and the cycle repeats.

Referring back to Figure 4, at the start of the cycle the train is about to reach the maneuvering yard, traveling from left to right on the upper tracks. At this point, it non-deterministically takes either tc_1 or tc_2, and starts to maneuver towards the lower tracks. If tc_3 is free, the train might choose to go down tc_1. After it enters tc_1, the train starts to travel at a speed that varies from 7 to 9 meters per second, until it crosses the 200 meters extension of tc_1, and is ready to occupy tc_3. If the latter is occupied, the train waits until it is released. Also, the train only occupies tc_3 if tc_4 and tc_5 are both free. Upon occupying tc_3, the train signals that it needs the positioning of SM_3 in the *reverse* state. Once that action is completed, the train crosses SM_3 and continues on tc_3. Before leaving tc_3, the train requests the occupation of tc_4. Upon being granted access to tc_4, the train signals for the movement of SM_4 to the *normal* state, and goes on to complete the crossing of

[2] 32000 feet is about 9754 meters; 30000 feet is about 9144 meters; and 2000 feet is about 610 meters.
[3] 550 miles per hour $\simeq$ 280 meters per second; and 490 miles per hour $\simeq$ 250 meters per second.
[4] http://www.boeing.com/commercial/747family/pf/pf_classics.html

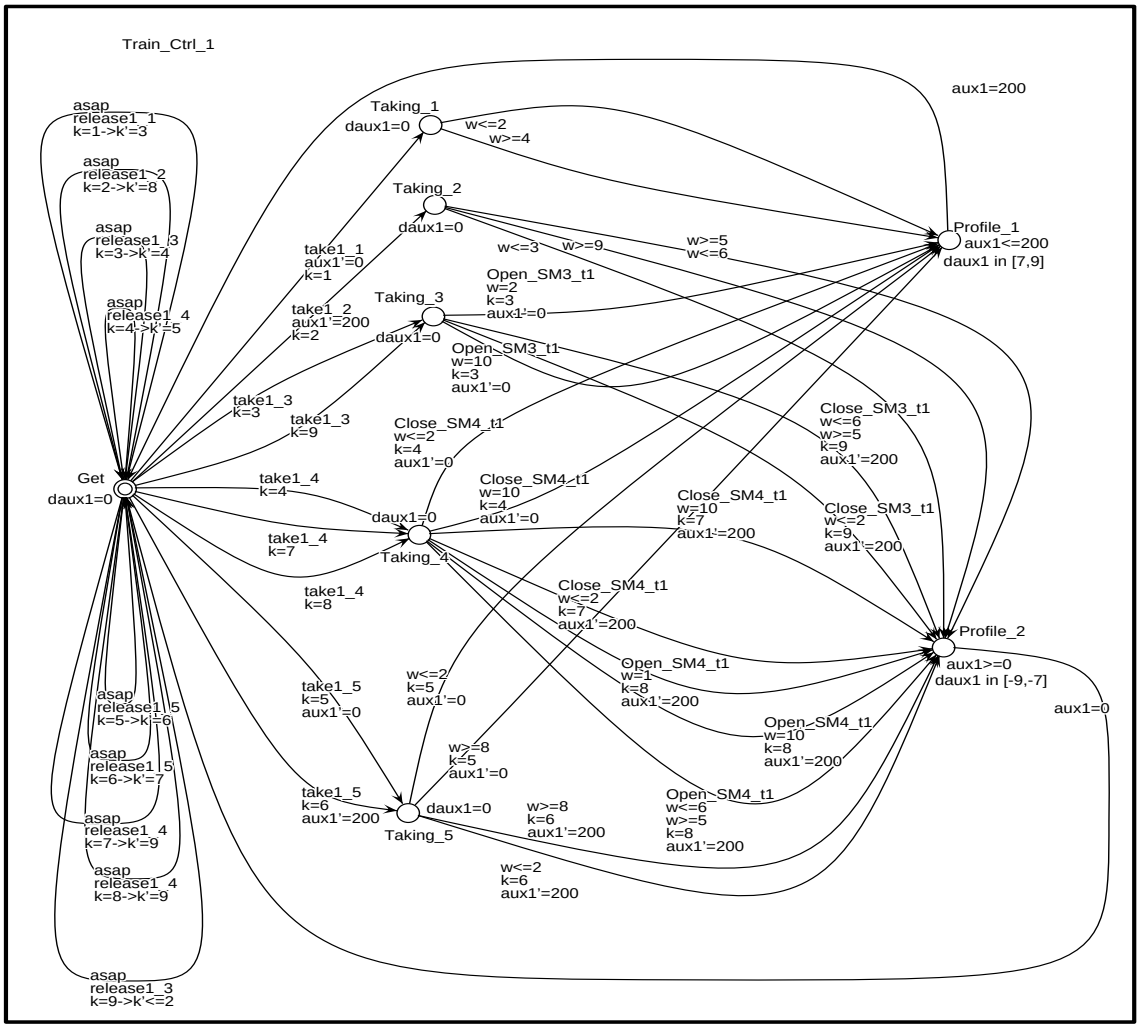


Figure 6: Train automaton

tc_4. When reaching tc_5 the train travels all its extension and, at its end, the train reverses direction. It, then, starts traveling from right to left with a speed between -9 to -7 meters per second. Upon reaching the end of tc_5, the train will only occupy tc_4 if it is already free. When entering tc_4, the train signals for the positioning of SM_4 in the *normal* state, proceeds to cross the SM_4 sector and travels all the way towards the end of tc_4. Next, it occupies tc_3 and signals for the movement of SM_3 to the *normal* state. After traveling all the extension of tc_3, the train abandons this circuit and returns to the initial point.

Alternatively, from the initial position, the train might non-deterministically decide to travel down tc_2. In this case, it will only occupy tc_4 if there is no train in tc_3 traveling in the opposite direction, nor there is another train occupying tc_4. When it reaches tc_4, the train signals for SM_4 to move to the *reverse* state. When this movement is completed, the train continues towards the left end of tc_4. Having reached it, the train frees tc_4 and enters tc_3, signaling for the movement of SM_3 to the *normal* position. After the train crosses tc_3, it returns to the initial position and restarts the cycle again.

The *SM* automaton is formed by four states as shown in Figure 7. Variables t_3 and t_4 are used to time the movement of the *SM*. The automaton is initiated in the *normal* state and the time it takes to complete a move, in either direction, is 15 seconds. There is a tolerance in this value, this imprecision being modeled by a nondeterministic choice of a clock rate in the interval $[0.8, 1]$. The discrete events *Open* and *Close* synchronize the train and the *tc* automata.

The model of a *tc* is illustrated in Figure 8. Each one has only two states, namely, *Release* and *Take*. The product of all the automata forms the product automaton that models the system.

5.2 The Hybrid Automata Models for the ATC System

Aviation texts, usually, do not treat measures in a uniform way. In general, the horizontal speed is given in miles per hour and the vertical rate is measured in feet per minute. The vertical distance is presented in feet and time is given in minutes or hours, while the horizontal distance is given in miles. See an example in Figure 5. 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[5].

Before the models are described the models in details, it should be mentioned that the TA signal was suppressed from the models treated here. When a conflict situation arises, a RA signal is issued directly. On the other hand, having more computer power available, it would have been 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 system modeled comprises two aircraft flying in opposite directions, as depicted in Figure 5. 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

[5] 1 foot $\simeq$ 0.3048 meters; 1 mile $\simeq$ 1829 meters; and 1 knot $\simeq$ 0.51 meters per second.

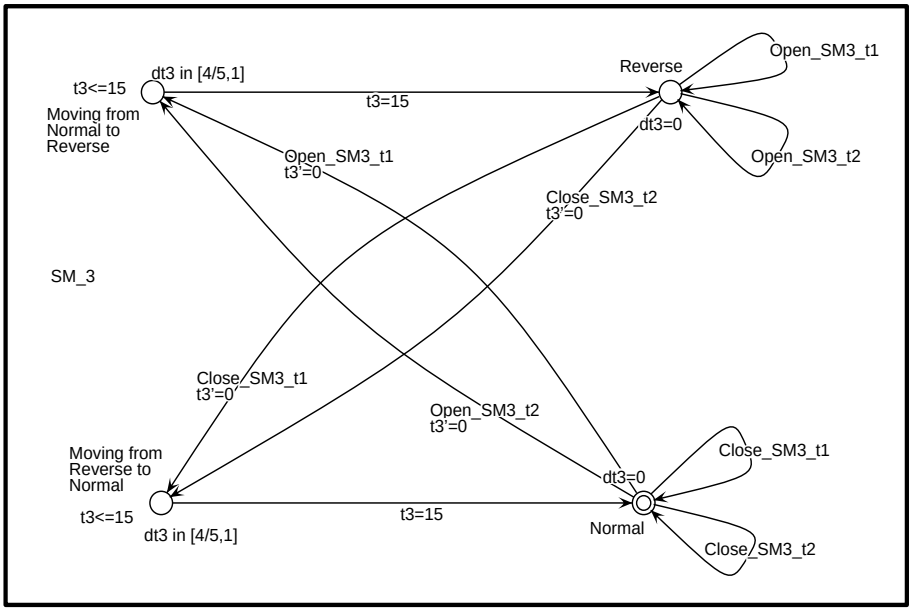


Figure 7: Switch machine automaton

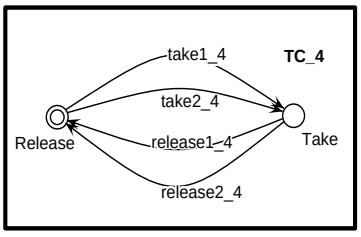


Figure 8: Track circuit automaton

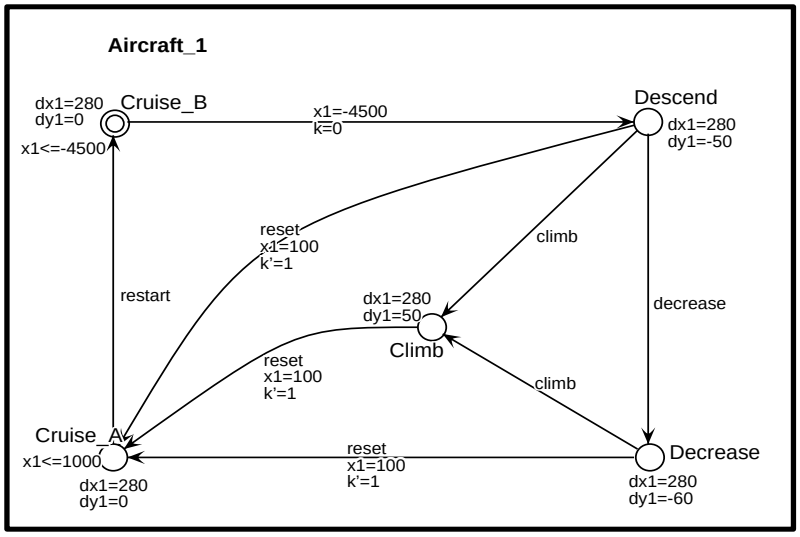


Figure 9: Model for the leftmost aircraft

is more complex than the automaton model for the other aircraft, which is flying in the opposite direction. The first aircraft, at some instants, may decide to engage in some kind of specific maneuver, while the second aircraft is assumed to remain cruising en route.

The full system model comprises three individual hybrid automata: one for each aircraft and another one for the system controller. The model for the leftmost aircraft is depicted in Figure 9 and the model for the rightmost aircraft is shown in Figure 10. Variable x_i indicates the horizontal distance and variable y_i indicates the vertical position of the aircraft, where index i indicates the automaton model for the corresponding 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 starts off in the *Cruise_B* mode, as shown in Figure 9. From this mode, at the mark of 4500 meters, it enters in the *Descend* mode and 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

[6]About 10000 feet per minute.

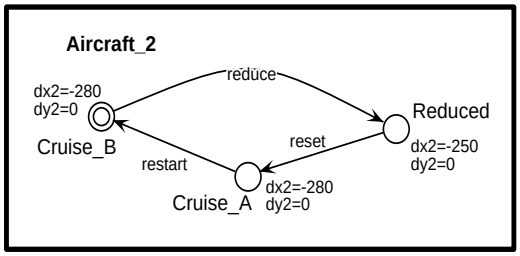


Figure 10: Model for the rightmost aircraft

aircraft have already passed by each other. It uses the *reset* event to synchronize with both aircraft models and the controller model. 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 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 10. 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 controller model implements a simplified version of the TCAS protocol and it is illustrated in Figure 11. 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

[7]About 550 miles per hour.

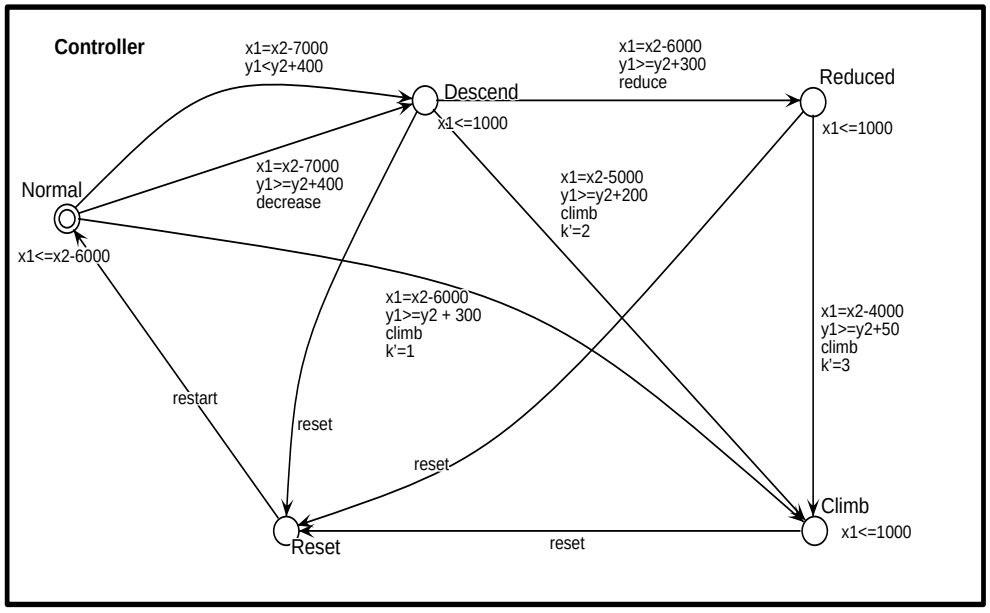


Figure 11: Controller automaton

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 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, thus moving the controller to the *Climb* mode. In the first case, a *reduce* event slows down the rightmost aircraft and puts

[8] About 1315 feet.
[9] About 985 feet.
[10] About 10000 feet per minute.

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 9. At this moment, the controller jumps to the *Reset* mode, where it waits until the aircraft have crossed and until 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 *Cruise_B* mode, restarting the cycle.

The automaton that governs the overall system behavior is obtained by calculating the product automaton of the three individual automata.

6 Computational Results

This section discusses the computational results obtained when running the system models described in Section 5.

Each system behavior was captured and analyzed by running a number of different experiments. Case studies of this kind have been able to reveal the limits of a safe operation of these systems. Some experiments can also be instrumental in obtaining tighter values for some system parameters. All computational results were obtained by running the HyTech tool[11] on a 2.4 GHz desktop Pentium IV PC, with 1 GB of RAM and 512MB of *swap* space. One of the major difficulties faced in the model development phase was caused by excessive resource consumption when running the tool. This problem was more acute when the HyTech tool attempted to compute the product automaton of more complex models. To overcome the memory usage problem, a careful *bottom-up* construction of the models was undertaken, always working at the tool limits. Memory usage was never a problem when running the parametric 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 models caused both the forward and backward analysis to fail. These observations indicate that the HyTech tool was operating close to its limit.

Certain internal abnormal conditions of the tool were raised, signaling possible multiplication overflow errors. The output message "*Will try hard to avoid library arithmetic overflow errors*" was one of them.

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

Table 1: Typical resource consumption

	Property	Analysis	Iteration	Memory (MB)	Time (s)
1	Unsafe	Backward	2 to 3	330 to 460	29 to 47
2	Safe	*Backward	> 5	$> 1.5GB$	> 1200
3	Unsafe	Forward	61 to 82	460 to 500	76 to 118
4	Safe	Forward	61	350 to 415	78 to 101
5	Unsafe	*Backward	> 7	$> 1.5GB$	> 1200
6	Safe	*Backward	> 7	$> 1.5GB$	> 1200
7	Unsafe	Forward	82	230 to 250	39 to 41
8	Safe	Forward	82	130 to 150	39 to 41

6.1 Results on the Subway Mesh Model

In this case, several experiments of safety property verification depleted all computational resources, and a number of compromise decisions had to be reached, in order to adjust the full model to the limits of the available hardware. The number of trains was limited to two, and only two speed profiles were allowed. In order to gain a perspective on the resource consumption during the experiments, Table 1 shows some typical data collected from these runs. The first column numbers the lines in the table. The second column indicates how the safety condition was characterized: by describing the safe region or by giving the region of undesirable, unsafe, configurations. The third column indicates the nature of the iteration process that converged: a forward analysis used the *Post* operator and a backward analysis used the *Pre* operator. The asterisk indicates that all computational resources were not enough to finish the corresponding verification. The fourth column shows the number of iterations that were necessary to reach convergence. The fifth column presents the approximate amount of RAM and swap memory that was used to complete the analyses. The sixth column gives the approximate running time for the corresponding experiment to reach completion. Some of these experiments were aborted due to insufficient machine resources. For example, from line three one reads that in one of the experiments the safety condition was modeled by describing the region of unsafe configurations, that the analysis proceeded forward, that the run needed 61 to 82 iterations to converge, that the execution time for this experiment was approximately of 76 up to 118 seconds, and used about 500MB of RAM.

The rest of this section reports on parameter synthesis for critical values of the subway mesh model. Due to machine resource restrictions, during synthesis, a simplification of the train automaton was used: since all synthesis experiments were local to a particular *tc*, only one *Profile* and one *Taking* state were maintained.

Table 2: Maximum distance after signaling the *SM*

Signal (m)	Maximum distance (m)	Time (s)	Speed (m/s)
0	675/4	15	[7,9]
20	755/4	15	[7,9]
125/4	800/4	15	[7,9]
40	835/4	15	[7,9]

```
Number of iterations required for reachability: 10
     train_dist >= 0
     & 4train_dist <= 675
```

Figure 12: Parametric analysis of latest point to signal

6.1.1 The maximum distance inside a *tc*

Here, the focus is on the maximum distance that the train can travel, inside the *tc*, before signaling for the *SM* to move, while still guaranteeing all safety conditions. The results obtained appear in Table 2. The first column indicates the distance, inside the track circuit, of the point where the train requests the *SM* to start moving. The second column shows the maximum distance that the train can reach, inside the track circuit, up until the moment when the *SM* locks. In the limit, if this distance measures all the 200 meters of a track circuit, the train could still pass through the *SM*. The third column indicates the time for the *SM* to complete its movement. In all of these simulations, the train speed was assumed to be in the interval between 7 and 9 meters per second, as the fourth column of the table shows.

From line 1 in Table 2, it can be seen that if the trains signal the *SM* to move immediately after entering the *tc*, the maximum distance it can travel inside the *tc*, before the *SM* locks, is 675/4 meters. Given that the *tc* total extension is 200 meters, it follows that the system operates safely if the trains always signal for the *SM* to move right upon entering the *tc*. Further, by consulting line 3 of Table 2, it can be seen that the largest mark on which the trains could signal the *SM* to move is the point located at 125/4 meters past the *tc* entrance, given that the *tc* extension is 800/4 meters.

Figure 12 illustrates the output of the HyTech tool for the experiment characterized by the first line in Table 2. The variable *train_dist* indicates the distance traveled by the train after entering the track circuit. The output of the tool is the clause $(train\_dist \geq 0) \wedge (train\_dist \leq 675/4)$. Disregarding the trivial condition $train\_dist \geq 0$, the output clause reduces to $train\_dist \leq 675/4$. In this analysis, the unsafe *SM* region was given as a target region on input to the tool. Hence, the output clause indicates the condition under which the unsafe *SM* region can be reached. That is, the *SM* region is unsafe when the

```
Number of iterations required for reachability: 10
     request <= train_dist
     & 4train_dist <= 4request + 675
```

Figure 13: Signaling point and maximum distance traveled

train has traveled any distance from 0 to 675/4 meters. Since the *SM* is at a distance of $800/4 = 200$ meters from the entrance of the *tc*, one can conclude that the operation is safe, provided that the train signals the *SM* upon entering the corresponding *tc*.

Another study was performed in order to disclose what is the safe relationship between the maximum distance traveled by the train inside the *tc* and the point where the train signals the *SM* to move. The result produced by the tool is shown in Figure 13. The variable *request* indicates the mark where the *SM* movement is requested. The variable *train_dist* indicates the distance traveled by the train inside of the track circuit. The output produced by the tool was the following conjunction $(request \leq train\_dist) \wedge (train\_dist \leq request + 675/4)$. This clause indicates what relationship must hold between the variable *request* and the variable *train_dist* for the unsafe region to be reached. Therefore, for a safe operation, the negation of this predicate must be observed, yielding the clause, $(request > train\_dist) \vee (train\_dist > request + 675/4)$. The first disjunct never occurs, since $train\_dist \geq request$ always holds, by construction. Hence, for a safe operation, the relationship $train\_dist > request + 675/4$ must always be observed.

6.1.2 Timing the *SM*

In this experiment, the objective was to determine how much slower the *SM*s could operate without compromising the safe functioning of the overall system. This parameter was synthesized using variable t_3 of the *SM* automaton.

Four experiments synthesized four different time instants for the *SM* movement. The results appear on Table 3. The first column indicates the distance inside of the track circuit where the train requests the *SM* to move. The second column shows that the maximum distance traveled by the train, up to the point when the *SM* locks, was fixed in 200 meters, corresponding to all the extension of a track circuit. The third column indicates which values for the *SM* operation time were synthesized, while still guaranteeing the system safety. Again, in all these experiments, the train speed was assumed to be in the interval between 7 and 9 meters per second, as indicated in the fourth column of the table.

By line 1 of Table 3, if it is required of the train that it always signals the *SM* upon entering the *tc*, then a slower *SM* could be used. Line 1, in Table 3, gives the value for this slower *SM*, namely, the *SM* would take 17.7 seconds to stop.

The output produced by the HyTech tool, for the case illustrated in the first line of the Table 3, is shown in Figure 14. The variable *SM_time* indicates the time needed by the *SM* to complete its movement. The relationship obtained indicates the values that cause the

Table 3: Parametric analysis of the *SM* time

Signal (m)	Distance (m)	Time (s)	Speed (m/s)
0	200	160/9	[7,9]
20	200	16	[7,9]
125/4	200	15	[7,9]
40	200	128/9	[7,9]

```
Number of iterations required for reachability: 5
     9SM_time >= 160
```

Figure 14: Parametric analysis for the *SM* movement

region of undesirable configurations to be reached. Negating this clause, the predicate that must be observed is revealed: $9SM\_time < 160$. Therefore, the clause that characterizes the safe operation is $SM\_time < 160/9$, as shown in the line 1, in Table 3.

The relationship that must hold between the *SM* operation time and the point where the train signals the *SM* to move was also synthesized. The result produced by the tool is presented in Figure 15. The variable *request* indicates the distance, counting from the beginning of the track circuit, where the *SM* movement is requested. The variable *SM_time* indicates the time needed for the *SM* to operate. The output produced by the tool indicates the relationship that must be observed in order for the region of unsafe configurations to be reached. Once the trivial condition $request \leq 200$ is discarded, the negation of the new clause reduces to $45SM\_time + 4request < 800$. As a result, the predicate that must hold is equivalent to $SM\_time < (800 - 4request)/45$.

6.1.3 *SM* timing and the point of signaling

The two previous sets of experiments synthesized values for *SM* timing and the farthest mark from which the train can signal the *SM* to move, both in isolation. Much richer information would result from the relationship that must hold between these two parameters, while still

```
Number of iterations required for reachability: 10
     request <= 200
     & 45SM_time + 4request >= 800
```

Figure 15: *SM* activation mark and its movement time

```
var
 final_reg, init_reg, reached: region;
init_reg:=
     loc[Ctrl_Train_1] = Get &
     loc[SM_3] = Normal &
     t3=0 & aux1=request;
final_reg :=
      loc[Ctrl_Train_1] = Get &
      aux1=train_dist &
      (loc[SM_3] = Normal |
      loc[SM_3] = Opening |
      loc[SM_3] = Closing);
reached:==
 reach forward from init_reg endreach;
print omit all locations
      hide non_parameters in
           reached & final_reg
      endhide;
```

Figure 16: Multiple parametric analysis

guaranteeing the system safety. Knowing these relationship would permit the simultaneous adjustment of these parameters.

In a first run, the mark where the train signals the *SM* to move was fixed right at the beginning of the corresponding *tc*. Figure 16 illustrates the input to the HyTech tool for this kind of parametric analysis. The region of initial configurations, *init_reg*, indicates that: (i) $train_1$ is in the imminence of occupying a *tc*; (ii) the SM_3 is in the normal position; (iii) the movement time for SM_3 starts zeroed; and (iv) the parametric analysis of the distance, measured inside tc_3, of the mark where the trains signals the *SM*, will be carried out using the predicate $aux_1 = request$. The final region, *final_reg*, describes the unsafe situations, which are: (i) the train is occupying tc_3; (ii) SM_3 is not in the *reverse* position, in other words, it is either in the *normal* position, or it is going from *normal* to *reverse*, or vice-versa; and (iii) the distance of the train, measured from the initial mark of tc_3, was parameterized using the condition $aux_1 = train\_dist$. The HyTech tool determines, automatically, the region which describes all the reachable configurations from the region *init_reg*. In this example, the *Post* operator was used to compute reachability, which implies a forward analysis. After obtaining the reachable region, the tool computes the intersection of this reachable region and the region described by *final_reg*. As the region *final_reg* describes the unsafe configurations for system operation, it is necessary to negate the predicate obtained in order to reveal the relationship which must hold between the parameters for the system to operate safely. The result produced by the tool appears in Figure 17. The variable

```
Number of iterations required for reachability: 10
     train_dist >= 0
     & 4train_dist <= 45SM_time
```

Figure 17: The *SM* timing and the distance traveled

```
Number of iterations required for reachability: 10
     request <= train_dist
     & 4train_dist <= 45SM_time + 4request
```

Figure 18: *SM* timing, signaling point and distance traveled

train_dist indicates the distance traveled by the train inside the *tc*. The variable *SM_time* indicates the *SM* operation time. The clause constructed by the tool is negated and the impossible condition $train\_dist < 0$ is discarded. What remains is the condition that must be observed for the system to operate safely: $SM\_time < (4/45) \times train\_dist$.

In a second run, the HyTech tool was programmed to obtain the relationship among the distance traveled by the train, the *SM* movement time and the mark where the train signals the *SM* to position itself, counting from the beginning of the *tc*. The result appears in Figure 18. The variable *request* indicates the distance of the mark point where the *SM* movement is requested. The variable *SM_time* indicates the *SM* operation time and the variable *train_dist* indicates the distance traveled by the train into the *tc*. The output produced by the tool is the relationship that must hold for the system to reach the region of unsafe configurations. Discarding of the infeasible clause $train\_dist < request$, the negation of the resulting condition gives $SM\_time < (4/45) \times (train\_dist - request)$. This condition ensures a safe operation for the segment of the subway mesh.

6.2 Results on the Air Traffic Protocol Model

Aviation texts, usually, do not treat measures in a uniform way. In general, the horizontal speed is given in miles per hour and the vertical rate is measured in feet per minute. The vertical distance is presented in feet and time is given in minutes or hours, while the horizontal distance is given in miles. See Figure 5. 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[12].

The ATC system modeled in this work comprises two aircraft flying in opposite directions, as depicted in Figure 5. The models capture several kinds of maneuvers that the

[12] 1 foot $\simeq$ 0.3048 meters; 1 mile $\simeq$ 1829 meters; and 1 knot $\simeq$ 0.51 meters per second.

```
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 19: Height parametric analysis

aircraft can perform in the same airspace. 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.

Figure 19 illustrates a typical input to the HyTech tool for a parametric analysis. The region of initial configurations, *init_reg*, indicates that: (i) *Aircraft_1* and *Aircraft_2* are in their respective start modes, *Cruise_B*; (ii) *Controller* is in its start mode, *Normal*; (iii) the initial positions for both aircraft are specified next, in meters[13]; and (iv) the auxiliary variable k is zeroed, which indicates the mode of origin when the *Climb* mode is entered. The final region, *final_reg*, captures an unsafe region according to the TCAS protocol. It comprises five lines: (i) *Aircraft_1* is in the *Decrease* mode, indicating a steep descent; (ii) *Aircraft_2* is in the *Reduced* mode, indicating a reduced horizontal speed; (iii) *Controller* is in the *Reduced* mode, reflecting the two conditions above; (iv) both aircraft are at the same position, passing by each other; and (v) the last condition parameterizes the vertical distance of the leftmost aircraft. The particular parameter values used in this exercise are typical. They could easily be changed in order to reflect different values.

The next line, in Figure 19, asks the HyTech tool to perform a forward analysis in order

[13] 6000 meters is about 3.3 miles; 9750 meters is about 32000 feet; and 9140 meters is about 30000 feet.

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

Figure 20: Parametric analysis before the controller action

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.

In the rest of this section, six case studies are presented, reporting on the synthesis of values for several critical parameters.

6.2.1 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 output of the HyTech tool for this experiment is shown in Figure 20. The parametric analysis shows that the first aircraft can reach a minimum height of 9482 meters[14] before the controller starts to act. Observe that this vertical distance is still above the cruising altitude of the second aircraft, indicating that the fly by is unsafe.

6.2.2 At critical point with increased descend

This scenario 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. Thus, the controller does not engage in climbing maneuvers, nor commands the second aircraft to reduce its cruising speed, as can be seen in Figure 9 and 11. The extra condition specified by $x_1 = x_2$ guarantees that the final region is focusing on the critical point. Again, the minimum height reached by the first aircraft was synthesized using the *height* parametric variable.

[14] About 31110 feet.

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

Figure 21: Parameterizing vertical separation and vertical distance

The HyTech tool synthesized the conditions that must be observed for this situation to occur. 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, 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[15]. 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 11. The tool, then, synthesized the maximum vertical separation between 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 21. It indicates that the maximum relative vertical distance between the aircraft, for this situation to occur, can be 431 meters[16]. 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 22 presents the output of the tool for this exercise, where the parameter *diff_horiz* indicates the desired minimum horizontal distance. The synthesis revealed that the command to increase 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[17], which is a little unsafer.

[15] About 1310 feet.
[16] About 1414 feet.
[17] About 28960 feet.

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

Figure 22: Parameterizing the horizontal separation and vertical distance

6.2.3 Reducing horizontal rate and no climbing

This experiment allows 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.

The same three situations as in the previous subsection were analyzed here. First, the HyTech tool synthesized the minimum vertical distance for the first aircraft, without parameterizing the coordinate values where it starts its increased descent. The value obtained was 8785 meters[18], which is a slightly safer value than the one revealed in the previous subsection, but the difference it is not significant for it. 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, the tool was run for the same scenario, where 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.

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

6.2.4 At the critical point and climbing

Next, the situation where the first aircraft is ordered to climb is analyzed. In this initial scenario, 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 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. All the evaluations are still being taken at the critical crossing point. Note that the value $k = 1$ is also part of the critical region. This guarantees that the automata are following the transitions as specified.

[18] About 28822 feet.

```
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
```

Figure 23: Horizontal separation and climbing

The result shows that the maximum height reached at the critical point, while the first aircraft is climbing, measures 10017 meters[19]. 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 11. 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 for the vertical separation parameter. Since the 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 23. The vertical separation is maintained at 300 meters. As can be seen, now a linear relationship 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[20]. 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 2000 feet and, as a consequence, the maneuver is deemed a safe one.

6.2.5 Increasing the descent and 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. The final region, here, is modified only by

[19] About 32865 feet.
[20] About 32588 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
```

Figure 24: Horizontal separation and vertical distance with increased descent, then climbing

changing the condition on the k variable. 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[21]. 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.

Next, the vertical separation of 200 meters was parameterized. The horizontal separation, in this case, remained at the 5000 meters mark. The 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 24 illustrates the output of the 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[22], still allowing for the minimum of 2000 feet required for a safe operation.

6.2.6 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. 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 HyTech tool computed that the first aircraft reaches a minimum height of 9615

[21] About 32163 feet.
[22] About 32060 feet.

meters[23] at the crossing point. Note that this value is approximately 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 6.2.3, 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 tool and specify a final region where the vertical distance reached by the first aircraft, at the crossing point, was exactly 32000 feet, the minimum required for safety, and then 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 was kept at 4000 meters. The tool showed that the vertical separation can be at most 36310/37, or about 97 meters, 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.

7 Conclusions

The daily operation of a subway mesh is a critical, real-time and reactive process. The same can be said about air traffic control software. Due to the potential in causing irreparable damages, these systems demand a rigorous validation procedure. This work is a step in this direction. Hybrid automata were used as mathematical model to build formal specifications of the real behavior of such systems. This formalism is well suited to model systems comprised of individual agents that manifest a continuous, dynamic behavior, regulated by the asynchronous occurrence of discrete events.

A subway mesh has all these hybrid characteristics. The continuous components are represented by the physical components of the mesh, and the discrete asynchronous events arise from the exchange of signals during communication sessions. Air Traffic Management (ATM) systems, when using the Traffic alert and Collision Avoidance System (TCAS) protocol, also 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 when participating aircrafts engage in a collision avoiding maneuver, operating under vigilance of the TCAS components.

In a first study, the formal models developed were used to verify some aspects about the operational behavior of a real subway mesh segment, representing a maneuvering yard. Operational parameters that dictate the trains relative positioning and movement, as well as some timing parameters of the track switching components were verified and synthesized.

[23] About 31545 feet.

In a second series of studies, formal models based on the TCAS protocol were developed, describing 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. The hybrid automata models were used to verify the overall system safety, when operating in a number of different scenarios.

The HyTech tool was used as computational support in the analysis and development of the models. It was helpful when analyzing and synthesizing several aspects of the linear models. Although hindered, at times, from insufficient computational resources, all the verification and synthesis experiments were successfully programmed and run on a typical 2.4 GHz desktop Pentium IV machine, with 1,5 GB of main memory.

Both systems studied in this work offer a great potential for extensions. Among additional aspects about the subway mesh operation that could be further analyzed using hybrid automata is the so called *shadow* parameter. The shadow is a safe distance, depending on the train speed, that must be observed by all trains approaching from behind. It would be interesting to analyze such a parameter, determining better adjustments for the train speed profile, thus improving the efficiency in the system operation, while still observing all the safety conditions. Another study that could be performed in the subway mesh refers to track circuits with variable length. With less intense traffic, the track circuit extension could be increased and, with mode demanding traffic, it could be shortened, thus improving the throughput in that region. Also, a more detailed description of the train movements inside stations would be desirable, for example, with respect to the opening of the automatic doors.

In the ATM case study, several other aspects deserve consideration. 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 able to analyze more complex models. In particular, the model could also consider aircraft passing through in the same direction and traveling in different layers. Another aspect relates to the horizontal separation parameters, not treated in this work.

References

[ACH+94] Rajeev A., C. Courcoubetis, N. Halbwachs, T. A. Henzinger, Pei-Hsin Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. In G. Cohen and J.-P. Quadrat, editors, *Proceedings of the 11th International Conference on Analysis and Optimization of Discrete Event Systems*, volume 199 of *Lecture Notes in Control and Information Science*, pages 331–351. Springer-Verlag, 1994.

[AD94] R. Alur and D. Dill. A theory if timed automata. *Theoretical Comp. Sci.*, 126:183–235, 1994.

[AdAG+01] Rajeev Alur, Luca de Alfaro, Radu Grosu, Thomas A. Henzinger, Minsu Kang, Freddy Mang, Rupak Majumdar, Christoph Meyer Kirsch, and Bow-Yaw Wang. Mocha: A model checking tool that exploits design structure. In

Proceedings of the 23rd International Conference on Software Engineering, 2001.

[AHH93] 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.

[AHWT97] 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.

[BCMD90] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential circuit verification using symbolic model checking. In *27th ACM/IEEE Design Automation Conference*, IEEE, pages 46–51, 1990.

[BLL+96a] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL in 1995. *Lecture Notes in Computer Science*, 1055:111–114, 1996.

[BLL+96b] 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, Dinamarca e Department of Computer Systems, Uppsala University, Suécia, December 1996.

[BLL+98] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson, Yi Wang, and Carsten Weise. New Generation of Uppaal. In *Int. Workshop on Software Tools for Technology Transfer*, June 1998.

[BM00] Adilson Luiz Bonifácio and Arnaldo Vieira Moura. Modeling and Parameters Synthesis for an Air Traffic Management System. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *Proceedings of the 3rd International Conference on Formal Methods in Computer Aided Design*, volume 1954 of *Lecture Notes in Computer Science*, pages 316–334, Austin, Texas, 1-3, November 2000. Springer-Verlag.

[BMCJAJ00a] 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, Escócia, 3-7, April 2000. IEEE Computer Society Press.

[BMCJAJ00b] Adilson Luiz Bonifácio, Arnaldo Vieira Moura, João Batista Camargo Jr., and Jorge Rady Almeida Junior. Formal Verification and Synthesis for an Air Traffic Management System. Technical Report 05, Computing Institute, University of Campinas, Campinas, Brazil, February 2000.

[Bry92] Randal E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. Technical Report CMU-CS-92-160, School of Computer Science, Carnegie Mellon University, July 1992.

[CEG+01] Darren Cofer, Eric Engstrom, Robert P. Goldman, David Musliner, and S. Vestal. Applications of model checking at Honeywell Laboratories. In M. B. Dwyer, editor, *Proceedings of the 8th International SPIN Workshop 2001*, volume 2057 of *Lecture Notes in Computer Science*, pages 296–303, Toronto, Canada, 20, May 2001. Springer-Verlag.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, April 1986.

[CJAJ98] João Batista Camargo Junior and Jorge Rady Almeida Júnior. Safety analysis case in the São Paulo Metro. Technical report, Departamento de Sistams Digitais e Engenharia da Computação, Escola Politécnica - Universidade de São Paulo, São Paulo, 1998.

[CMM+94] S. V. Campos, M. Minea, W. Marrero, E. M. Clarke, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. In *Proc. 15th IEEE Real-Time Systems Symp.*, pages 266–270. IEEE Comp. Soc. Press, Dec 1994. San Juan, Porto Rico.

[CMMC95] S. V. Campos, M. Minea, W. Marrero, and E. M. Clarke. Timing analysis of industrial real-time systems. In *Proc. Workshop on Insdustrial-strength Formal Specification Techniques*, pages 97–107. IEEE Comp. Soc. Press, Apr 1995. Boca Raton, FL, USA.

[Dij75] E. W. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *CACM*, 18(8):453–457, August 1975.

[FAM98] José Henrique Zaccardi de Freitas, Antonio Accurso, and Ivaldo Lopes Mathias. A evolução tecnológica da cmsp e o estado da arte de sistemas de sinalização baseado em comunicação. In *Revista Engenharia*, volume 529, pages 116–124, 1998. (In Portuguese).

[Gro95] CORPORATE The RAISE Language Group. *The RAISE Development Method*. Prentice-Hall, Inc., 1995.

[Har87] D Harel. Statecharts: a visual formalism for complex systems. *Sci. Comput. Prog.*, 8:231–274, 1987.

[Hei98] Kathryn T. Heimerman. Air traffic control modeling. National Academy Press, The MITRE Corporation, 1998. Como aparece no livro intitulado Frontiers of Engineering 1997.

[Hen00] Thomas A. Henzinger. Masaccio: A Formal Model for Embedded Components. In *Proceedings of the First IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 549–563. Springer-Verlag, 2000.

[HH94] Thomas A. Henzinger and Pei-Hsin Ho. HyTech: The cornell hybrid technology tool. *Workshop on Hybrid Systems and Autonomous Control*, October 1994.

[HHWT95a] 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, Itália, 5-7, December 1995. IEEE Computer Society Press.

[HHWT95b] 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.

[HHWT97] 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.

[Ho95] Pei-Hsin Ho. *Automatic Analysis of Hybrid Systems*. PhD thesis, Cornell University, August 1995.

[HPP94] N. Halbwachs, Y.-E. Proy, and Raymond P. Verification of linear hybrid systems by means of convex approximations. In *International Symposium on Static Analysis*, volume LNCS 864 of *Lecture Notes in Computer Science*, 1994.

[HPS01] Tomas Hlavaty, Libor Preucil, and Petr Stepan. Case study: Formal specification and verification of railway interlocking system. In *Proceedings of the 27th Euromicro Conference 2001: A Net Odyssey*, pages 258–264, Warsaw, Poland, 4-6, September 2001. IEEE Computer Society Press.

[HR99] N. Halbwachs and P. Raymond. Validation of synchronous reactive systems: from formal verification to automatic testing. In *ASIAN'99, Asian Computing Science Conference*, Phuket (Thailand), December 1999. LNCS 1742, Springer Verlag.

[HWKF02] Scott Hazelhurst, Osnat Weissberg, Gila Kamhi, and Limor Fix. A hybrid verification approach: getting deep into the design. In *Proceedings of the 39th conference on Design automation*, pages 111–116, New Orleans, Louisiana, USA, 10-14, June 2002. ACM Press.

[HWT95] 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, Belgica, July 1995. Springer-Verlag.

[HWT96] 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.

[Law99] John Law. Acas II programme. ACAS Programme Manager, January 1999.

[LL95] 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, Massachusetts 02139, 1995.

[LPY97] 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.

[LPY01] Magnus Lindahl, Paul Pettersson, and Wang Yi. Formal Design and Analysis of a Gearbox Controller. *Springer International Journal of Software Tools for Technology Transfer (STTT)*, 3(3):353–368, 2001.

[Mag99] Jeff Magee. Behavioral analysis of software architectures using ltsa. In *ICSE*, pages 634–637, 1999.

[MF01] Mieke Massink and Nicoletta de Francesco. Modelling Free Flight with Collision Avoidance. In *Proceedings of the Seventh International Conference on Engineering of Complex Computer Systems*, pages 270–281, Skövde, Sweeden, 11-13, June 2001. IEEE Computer Society Press.

[MKN+00] Jeff Magee, Jeff Kramer, Bashar Nuseibeh, David Bush, and Julia Sonander. Hybrid Model Visualization in Requirements and Design: A Preliminary Investigation. In *Proceedings of the Tenth International Workshop on Software Specification and Design*, volume 939 of *Lecture Notes in Computer Science*, pages 3–11, San Diego, California, 5-7, November 2000. IEEE Computer Society Press.

[Ros95] A. W. Roscoe. Modelling and verifying key-exchange protocols using csp and fdr. In *CSFW '95: Proceedings of the The Eighth IEEE Computer Security Foundations Workshop (CSFW '95)*, page 98. IEEE Computer Society, 1995.

[Sco00] Phil Scott. Technology and business: Self-control in the skies. *Scientific American*, pages 24–55, January 2000.

[TPS97] 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.

[Tra01] Boris Trakhtenbrot. Automata, circuits and hybrids: facets of continuous time. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 754–755, Hersonissos, Greece, 6-8, July 2001. ACM Press.

[Win02] Kirsten Winter. Model checking railway interlocking systems. In *CRPITS '02: Proceedings of the twenty-fifth Australasian conference on Computer science*, pages 303–310. Australian Computer Society, Inc., 2002.

[WK98] 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, Massachusetts.