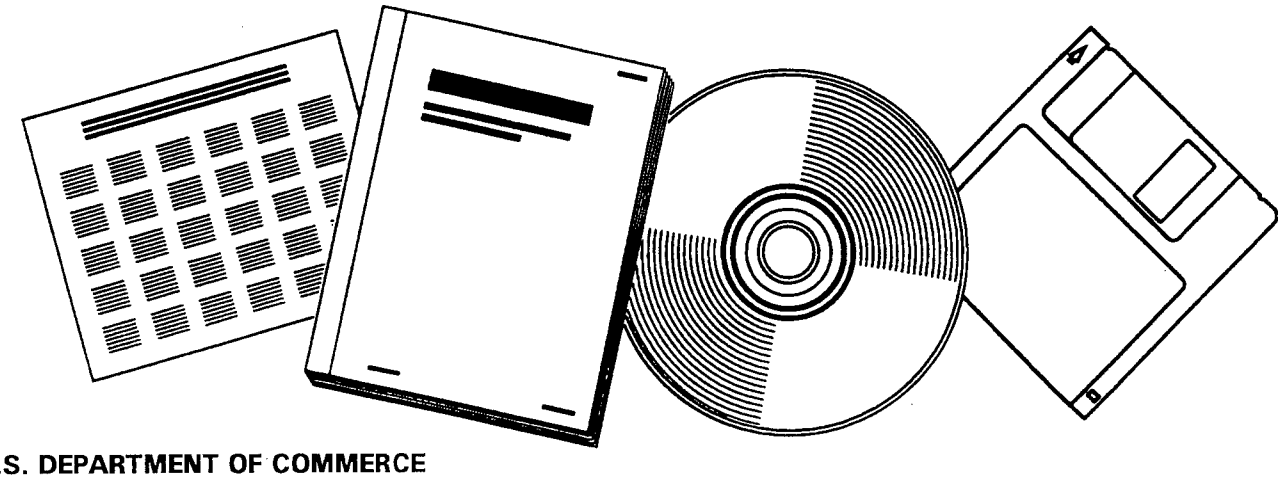
PB98-119142

MODELLING AND VERIFICATON OF ADVANCED VEHICLE CONTROL SYSTEMS USING TIMING BASED COMPUTING TECHNIQUES

9 JAN 98



U.S. DEPARTMENT OF COMMERCE
National Technical Information Service

Final Research Report

Project Number: MITR8-3

Project Title: Modelling and Verification of Advanced Vehicle Control Systems Using Timing-Based Computing Techniques

Principal Investigator: Nancy Lynch

Sponsor: US Department of Transportation University Transportation Centers Program

DISCLAIMER

Exhibit C

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle
Modelling and Verification of Advanced Vehicle Control Systems Using Timing-Based Computing Techniques

5. Report Date
1/9/98

6. Performing Organization Code

7. Author(s)
Nancy Lynch

8. Performing Organization Report No.

9. Performing Organization Name and Address
Massachusetts Institute of Technology

10. Work Unit No. (TRAIS)

11. Contract or Grant No.
DTRS95-G-0001

12. Sponsoring Agency Name and Address
New England (Region One) UTC
Massachusetts Institute of Technology
77 Massachusetts Avenue, Room 1-235
Cambridge, MA 02139

13. Type of Report and Period Covered
Final Year 8 Project
9/1/95-2/28/98

14. Sponsoring Agency Code

15. Supplementary Notes
Supported by a grant from the US Department of Transportation University Transportation Centers Program

16. Abstract
Please see attached.

17. Key Words
Please see attached.

18. Distribution Statement

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
		20	60K

Form DOT F 1700.7 Reproduction of form and completed page is authorized

Keywords

automated transportation systems, distributed computing, safety, hybrid I/O automata, parallel composition, invariant, levels of abstraction, simulation relation, personal rapid transit, deceleration maneuver, acceleration maneuver, vehicle protection system, automated highways, platoon, aircraft collision avoidance

Abstract

We have been working on hybrid system modelling, with automated transportation systems as our target application. We have developed a hybrid automaton model that we call the *hybrid I/O automaton (HIOA)* model [1], and have developed decomposition and proof methods for this model. HIOAs allows description of both discrete and continuous system components – using discrete mathematics notation for the discrete parts and continuous mathematics notation for the continuous parts of the system. The proof methods for HIOAs are based on techniques used previously for distributed computer systems: parallel composition, invariant assertions, and levels of abstraction.

Our notion of composition is based on sharing actions or sharing values of certain variables. Our invariants and simulation relations may involve real-world quantities like position, velocity, etc., in addition to the usual discrete quantities. We have designed our model and methods to incorporate control theory techniques as parts of proofs of invariants and simulation relations. The model clearly separates the use of control theory and computer science reasoning methods, while allowing them to be used in combination.

We have applied our model and methods to many automated transportation system settings, including controlled deceleration and acceleration maneuvers, platoon maneuvers (as in the California PATH project), vehicle protection systems (Raytheon), and aircraft collision avoidance (TCAS, CTAS). In each case, we have modelled both the discrete and continuous system components, at least at a high level of abstraction. We have obtained results giving proofs of safety properties. These results are typically conditional results, saying things like: "under certain assumptions about the behavior of the vehicles, safety is guaranteed".

[1] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control* (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of *Lecture Notes in Computer Science*, pages 496–510. Springer-Verlag, 1996.

Original Goal Statement

The goals of this project were (a) to develop a suitable formal framework for reasoning about advanced vehicle control systems such as those arising in the Raytheon Personal Rapid Transit project and the California PATH automated highway project, and (b) to use this framework to obtain useful results about several typical transportation scenarios. We planned to do this using powerful techniques arising in computer science, in particular, in the study of timing-based distributed computer systems.

The framework was to be built upon a particular (non-finite-state) *timed automaton* model that we had already developed, and that has been used successfully for modelling and verifying timing-based computer communication systems. The framework was to permit description of both real world components and computer components of advanced vehicle control systems. It was to support modular description and reasoning, using a variety of verification and analysis tools. The scenarios we planned to consider involve, among other things, attaining and maintaining safe speeds and inter-vehicle distances, implementing typical vehicle maneuvers, tracking specified vehicle trajectories, and protecting against catastrophes. The results we sought about the scenarios included proofs of safety and performance properties.

This work was intended to provide designers of advanced vehicle control systems with a framework they could use for describing, verifying and analyzing their designs. It was also intended to provide a thorough understanding of the chosen scenarios and how they fit into a complete transportation system design.

We planned to begin by working with transportation specialists Prof. Shankar Sastry at the University of California at Berkeley, and Roy Johnson and Steve Spielman at Raytheon Company, to identify typical transportation problems. For example, we planned to consider:
1. Problems of slowing down a vehicle sufficiently so that it attains a safe speed before reaching a particular segment of the roadway.
2. Problems of accommodating lane changes.
3. Problems of merging and splitting roadways.
4. Problems of joining and splitting "platoons" of vehicles.
5. Problems of tracking a specified vehicle trajectory.
6. Problems of resolving conflicts among several different planned vehicle maneuvers.
7. Problems of protecting against catastrophes (e.g., crashes), even in the presence of a wide range of unpredictable/faulty behavior on the part of the system and the environment.
8. Problems of "handing off" control of vehicles from one computer to a nearby computer, in a distributed computer system.
9. Problems of routing traffic for maximum system throughput.

Our background for carrying out this research is as follows. For the past 16 years, Prof. Lynch's Theory of Distributed Systems (TDS) group at M.I.T. has been one of the leading research groups working on distributed and real time systems and algorithms. Typical problems we have studied include problems of communication, resource allocation, consensus, process control, concurrency control, and synchronization. Our work fits roughly into three categories: (a) formal modelling and verification methods, (b) algorithms and impossibility results, and (c) applications.

I. Background

The high-level goal of our project was to produce some new and better tools for establishing safety and performance properties – especially safety properties – for automated transportation systems. Some examples of systems to which these tools were intended to apply are platoons of cars on highways (as occurring, for example, in the California PATH project [18], automated transit systems (as in Raytheon's Personal Rapid Transit project), and aircraft collision avoidance systems (for example, the TCAS collision-avoidance system [4]. Traditional approaches to establishing safety and performance properties involve setting up a computer model, simulating a large number of scenarios, and determining if the system seems to behave properly in those scenarios. But there is a problem with this approach: The fact that the system behaves right in the chosen scenarios does not imply that it will always behave right; it is not possible to simulate all possible scenarios.

To address this problem, we are using techniques that have recently become very important in computer science, in particular, in the area of *distributed computing.* Distributed computing systems are collections of interacting components, for example, active entities called "processes", communication channels, and various kinds of memory modules. As normally considered in computer science, distributed systems maintain discrete data and act in discrete steps. Unlike automated transportation systems, they exhibit no interesting continuous behavior (except for the passage of time).

Like automated transportation systems, distributed computing systems can be quite complicated to understand. For example, they contain many interacting components, all going at their own speeds, some of the components may fail, and no component "knows" precisely what the rest of the system is doing.

Nevertheless, it is important to verify correctness properties of such systems, in particular, to verify what is known in computer science as *safety properties.* Safety properties in computer science are very similar to safety properties in transportation systems – they say that some undesirable event does not occur. For example, typical safety properties for a distributed system say that a communication protocol does not deliver the wrong data, or reorder the messages, or deliver the same messages twice, and that a shared memory does not get corrupted. Other examples of safety properties are performance properties, for example, bounds on message delivery time.

As for transportation systems, it is not obvious how one can verify safety properties for distributed computing systems. Again, it is possible to simulate the system, but simulation alone does not test exhaustively for correctness.

The computer science methods that have arisen to cope with this problem involve modelling

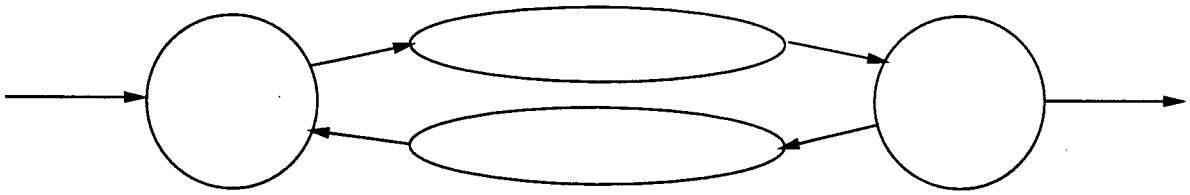


Figure 1: Structure of Alternating Bit Protocol.

the systems carefully, using automaton (state machine) models based on discrete mathematics. (These state machines are not the same as the finite-state automata used elsewhere in computer science – in particular, they can be infinite-state.) Automaton models for a distributed system are used as the basis for formal, mathematical reasoning about the system's behavior. Theorems can be proved, asserting that the modelled system's behavior satisfies certain conditions. These are statements about what the system will do in *all circumstances*, not in just a chosen few as in a simulation. Since there can be infinitely many different possible executions, this yields a qualitative improvement in information.

Among the leading formal models for distributed systems are two that were developed in our research group: I/O automata [14, 15] and timed I/O automata [13, 16]

The key ideas that make this approach work are three methods of imposing structure on complex distributed systems: *parallel composition*, *invariant assertions*, and *levels of abstraction*. *Parallel composition* allows a system to be described formally in pieces, with a formally-defined parallel composition operator describing how they are combined. The standard composition operators are *substitutive*, which means it possible to understand the behavior of a system without looking inside components to see how they are implemented. Parallel composition allows formal reasoning about the system in pieces.

For example, Figure 1 depicts the component structure of the well-known Alternating Bit Protocol. The figure shows two communicating processes and two one-way channels connecting them. The Alternating Bit Protocol is designed to achieve reliable communication over an unreliable channel. It is possible to understand something about how the protocol works by understanding the behavior of the individual pieces. (E.g., the low-level channels do not reorder messages, though they may lose or duplicate them.)

Invariants are properties that are true of all reachable systems states. Key safety properties for distributed systems are usually formulated as invariants. Also, important facts about the behavior of a distributed system that help to guarantee that the safety properties are true are usually described as invariants. Invariants have the nice property that they can be

proved using mathematical induction on the length of a system execution.

For example, in the Alternating Bit Protocol, the protocol sends messages repeatedly, tagged with 0 or 1, alternating for successive messages. This introduces the possibility of ambiguity, because the same tag is used for many messages. An important invariant for this system is that the 0 tags all occur consecutively in the channels, and likewise for the 1 tags; the 0s and 1s are not interleaved in a complicated way. This allows the protocol to sort out the messages correctly.

Levels of abstraction involve considering a high-level view of a complex system, which is somehow easier to understand than the complete, detailed system. The detailed system is related formally to the high-level view, using a relationship known as a *simulation relation*. As for invariants, simulation relations can be formally proved to hold using mathematical induction. Using levels of abstraction, a system can be developed in many layers, introducing more and more detail, more optimizations, etc.; this discipline of system development is known in computer science as "successive refinement".

For example, the Alternating Bit Protocol can be viewed as an implementation of a single abstract queue (sequence) of high-level messages. Or, it can be viewed as an implementation of a similar protocol that uses successively increasing (unbounded) sequence numbers, which in turn implements a queue.

These methods have worked extremely well for distributed systems. The state of the art in distributed computing is that it is now possible to model very complicated distributed systems/algorithms using state machines decomposed using parallel composition and levels of abstraction, and to prove safety properties using invariants and simulation relations. These methods have been widely applied in areas such as communication, distributed databases, and fault-tolerant computing. Lynch's book [9], for example, describes these methods and some of their applications.

During approximately the past four years, these computer science ideas have been pushed into a new research area that is known among computer scientists as *hybrid systems*. A hybrid system is a system consisting of a combination of discrete components (e.g., computers, protocols) and continuous components (e.g., planes, trains, and automobiles; machines in factories, nuclear reactors). Such systems are becoming more and more common, as more and smarter computer automation is being introduced into many physical applications. Safety properties, in the sense that transportation people mean by safety, are often important for such systems.

Various hybrid automaton models have arisen, allowing modelling of both discrete and continuous components. A variety of proof techniques have begun to be developed, some along

the lines discussed above – composition, invariants, abstraction; others arising from control theory, and others employing exhaustive-searching techniques. These techniques have begun to be applied to some examples, mostly toy examples.

II. Our Project

For about the past four years, members of our group have been working on hybrid system modelling, with automated transportation systems as our target application. We have developed a hybrid automaton model that we call the *hybrid I/O automaton (HIOA)* model [12], and have developed decomposition and proof methods for this model. HIOAs allows description of both discrete and continuous system components – using discrete mathematics notation for the discrete parts and continuous mathematics notation for the continuous parts of the system. The proof methods for HIOAs are based on those described above for distributed computer systems: parallel composition, invariant assertions, and levels of abstraction.

Our notion of composition is based on sharing actions or sharing values of certain variables. Our invariants and simulation relations may involve real-world quantities like position, velocity, etc., in addition to the usual discrete quantities. We have designed our model and methods to incorporate control theory techniques as parts of proofs of invariants and simulation relations. The model clearly separates the use of control theory and computer science reasoning methods, while allowing them to be used in combination.

We have applied our model and methods to many automated transportation system settings, including controlled deceleration and acceleration maneuvers, platoon maneuvers (as in the California PATH project), vehicle protection systems (Raytheon), and aircraft collision avoidance (TCAS, CTAS). In each case, we have modelled both the discrete and continuous system components, at least at a high level of abstraction. We have obtained results giving proofs of safety properties. These results are typically conditional results, saying things like: "under certain assumptions about the behavior of the vehicles, safety is guaranteed".

III. Deceleration and Acceleration Maneuvers

Our first project involved analyzing some toy examples involving deceleration and acceleration of vehicles on tracks [19, 20, 10].

A. Deceleration Maneuvers

We first considered the simple problem of ensuring that the speed of a vehicle on a straight track is in a given range, $[v_{min}, v_{max}]$ of speeds, when the vehicle reaches a particular track position x_f [19, 20]. We assume that the vehicle starts at position 0, with velocity v_s, which is greater than the maximum target velocity v_{max}. In the setup we consider, a controller for

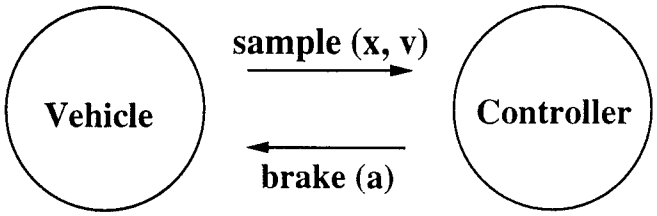


Figure 2: Interaction between vehicle and controller.

the vehicle is able to issue a $brake(a)$ command, $a < 0$, which causes the vehicle to decelerate at an unknown, possibly-varying rate in $[a - \epsilon, a]$. We assume that the controller learns the position x and velocity $\dot{x}$ of the vehicle every d time units. See Figure
reffg: vehicle-controller Certain restrictions on the constants are needed in order for the problem to be solvable.

There are many possible controller strategies. For example, one of those we model is as follows: The controller initially sets the acceleration to aim to reach v_{max} exactly when $x = x_f$. However, the vehicle might actually decelerate faster. Rather than changing the acceleration repeatedly, the controller leaves the acceleration alone until such time as the velocity actually becomes less than v_{max}. Thereafter, at each sample point, the controller resets the acceleration to aim so that $\dot{x} = v_{max}$ at the next sample point. This is a somewhat arbitrary choice of strategy, though the decision not to change the acceleration too frequently was based on discussions with Raytheon developers.

For this simple strategy, we proved that the require range is in fact reached. The proof splits up into two arguments, one to show the upper bound v_{max} and one for the lower bound v_{min}. The key to each proof turns out to be an invariant assertion. We show these below, in order to illustrate how the proof method works. First, for the upper bound, we use:

Invariant 1: In all reachable states,
if $x \leq x_f$ then $x_f - x \geq \frac{v_{max}^2 - \dot{x}^2}{2acc}$.

Invariant 1 says that, at any point during the execution, there is always enough remaining distance to reach v_{max}, even if the deceleration is the slowest possible. The acc here denotes the latest acceleration set by the controller – the actual acceleration $\ddot{x}$ may be smaller.

Formally, an invariant like this one is a property of system states that is claimed to be true in all reachable states of the system. Typically, invariants are proved by induction on the length of an execution. Here, since the system is a hybrid system, an execution consists of both discrete and continuous steps. However, we are still able to prove the result by induction.

in this case on the total number of steps, both continuous and discrete. Basically, we show that discrete steps and continuous steps (trajectories) both preserve the invariant. For this, we use a combination of continuous and discrete arguments, where the discrete arguments use algebraic and logical deduction, and the continuous arguments use simple calculations involving derivatives. A general theorem about the HIOA model shows how the two kinds of arguments can be pasted together cleanly, to give a correctness result for the entire system.

Second, for the lower bound, we use:

Invariant 2: In all reachable states, $\dot{x} \geq v_{min}$.

Invariant 2 must be strengthened in order for us to prove it by induction. In particular, in turns out that we have to say something about what is guaranteed between sample points:

Invariant 3: In all reachable states,
$\dot{x} + (acc - \epsilon)(\textit{next-sample} - \textit{now}) \geq v_{min}$.

Invariant 3 says that, at all times, even between sample points, the velocity is such that it is guaranteed to stay above v_{min} until next sample point. More precisely, it says that if the current velocity is modified by allowing the strongest deceleration (that is, the minimum acceleration) consistent with the currently-set acceleration acc (namely, $acc - \epsilon$), until the next sample point, then the result will still be at least v_{min}.

Although this last statement sounds as if it is talking about future behavior, in fact every quantity mentioned in Invariant 3 is part of the actual state of the automaton model. For example, the "time of the next sample point" is modelled by a state variable *next-sample*.

For intuition, consider two special cases: If $\textit{next-sample} = \textit{now}$, it means that we are at the end of a sample interval, about to sample again. In this case, the inequality reduces to $\dot{x} \geq v_{min}$, as needed. At the other extreme, if $\textit{next-sample} = \textit{now} + d$, it means that we are at the beginning of a sample interval, having just set the acceleration. In this case, the inequality reduces to: $\dot{x} + (acc - \epsilon)d \geq v_{min}$. The extra term of $(acc - \epsilon)d$ is the leeway that we require in order to ensure that the velocity doesn't degrade too badly during the sample interval. Again, we prove this by induction.

We treated several variants of the deceleration problem. In addition to composition to describe the combination of vehicle and controller, and invariants, some of the variants used levels of abstraction. This was used for replacing high-level descriptions of controllers by more detailed implementations.

B. Acceleration Maneuver

In order to explore the use of levels-of-abstraction methods in hybrid systems, we carried out a three-level analysis of a toy vehicle acceleration maneuver [10]. The goal of the maneuver

is to cause a vehicle, starting at velocity 0 at time 0, to attain a velocity of b (or as close to b as possible) at a later time a. The vehicle is assumed to provide accurate sampled data every d time units. The vehicle is assumed to be capable of receiving control signals, one immediately after each vehicle data output. Each control signal can set an "acceleration variable", acc, to an arbitrary real number. However, the actual acceleration exhibited by the vehicle need not be exactly equal to acc – instead, we assume that it is defined by an integrable function whose values are always in the range $[acc - \epsilon, acc]$. We can think of this uncertainty as representing, say, uncertainty in the performance of the vehicle's propulsion system.

The vehicle interacts with a controller, presumably a computer. In our work, we describe a particular controller and analyze the behavior of the combination of the vehicle and controller. One conclusion we draw is that the velocity of the vehicle at time a is in the range $[b - \epsilon d, b]$. That is, the uncertainty in setting acc combines multiplicatively with the sampling period to yield the uncertainty in the final velocity of the vehicle. More strongly, we obtain a range for the velocity of the vehicle at each time in the interval $[0, a]$.

We prove this fact using invariants and levels of abstraction (in particular, simulation methods), based on hybrid I/O automata [12]. Many of the pieces of the proofs use standard continuous methods, such as solving algebraic and differential equations. The entire proof represents a smooth combination of discrete and continuous methods.

The point of this exercise is to demonstrate some simple uses of levels of abstraction in reasoning about hybrid control problems. We use levels of abstraction here for two purposes: (a) to express the relationship between a derivative-based description of a system and an explicit description, and (b) to express the relationship between a system in which corrections are made at discrete sampling points and a system in which corrections are made continuously. The uncertainty in the acceleration is treated at all three levels of our example, and is integrated throughout the presentation.

We do not contribute anything new in the way of techniques for continuous mathematics; for example, we use standard methods of solving differential equations. Our contributions lie, rather, in the smooth combination of discrete and continuous methods within a single mathematical framework, and in the application of standard methods of discrete analysis (in particular, invariants and levels of abstraction) to hybrid systems. Our methods are particularly good at handling uncertainties and other forms of system nondeterminism.

IV. Platoon Safety

In [3, 1, 8], we have proved safety properties for a collection of maneuvers for platoons of vehicles on automated highways, as used, for example, in the California PATH project [18

In the PATH design, platoons of cars travel on a public highway, under automated control. Within a platoon, cars can be a mere 2 meters apart. They can travel at about 30 meters/sec. Platoons join and split apart based on destinations and desire to need to utilize the highway "bandwidth".

The basic problem is to ensure that the relative velocity of any collision is at most v_{allow}, a constant that is approximately 3 meters/sec. There are two good situations: First, if two vehicles are very close together, as they would normally be when they are travelling as part of the same platoon, then if the first one brakes, the cars collide quickly, before the relative velocity has a chance to grow too large. Second, if two vehicles are far apart, then the second has time to slow down or stop if the first one brakes.

Questions arise in intermediate situations, which occur, for example, during a platoon joining maneuver. Also, the two good cases described above make the most sense for the first collision – if there is a chain reaction, it is not so clear when these good cases apply.

We first analyzed the relative velocity of the first collision only [3, 1]. For this case, we proved an invariant guaranteeing safety:

Invariant: Either $\dot{x_1} + v_{allow} \geq \dot{x_2}$
or $x_1 - x_2 \geq \frac{\dot{x_1}^2 + v_{allow}^2 - \dot{x_2}^2}{2a}$.

Here, a is the greatest possible deceleration (the largest negative acceleration). This invariant says that either the relative velocities are already close enough, or the distance between the vehicles is great enough to allow the second to slow down sufficiently.

Again, we proved this using induction. We also showed that this condition is optimal – if it is not satisfied, then the cars may crash with higher velocity. (In fact, they *will* crash, if the first car brakes as strongly as possible.) This optimality condition is also proved using induction.

We then considered the case of multiple collisions (possibly chain reactions)[8]. This case is complicated, and we still have only partial results. We considered emergency braking maneuvers within a single platoon, where cars have possible-different braking strengths. For simplicity, we assume that each car brakes at its own constant rate. Also, for simplicity, we assume that cars all have equal masses, and that the coefficient of restitution is 1 (i.e., totally elastic collisions). Even with these simplifying assumptions, we have not yet obtained a complete solution – some complicated situations arise.

Even for the case of 2 cars only, repeated collisions are possible: Suppose that car 1 has a stronger brake than car 2 but car 2 starts with a higher velocity. Then the cars can collide and exchange velocities (this happens because they have the same masses and the restitution

is 1), making 1 faster. But this is just temporary – soon, 1's stronger brake slows it down sufficiently so car 2 is again faster, and they collide again. This can happen repeatedly.

We were able to obtain close bounds on the ranges of the parameters (for initial positions and velocities and the braking strengths) that ensure that all collisions are safe (relative velocity $\leq v_{allow}$). The sufficient condition for safety looks like:

Either M and $(\dot{x_1} - \dot{x_2})^2 - 2(a_1 - a_2)(x_1 - x_2) \leq v_{allow}^2$
or $\sim M$ and $\dot{x_2}^2 - \frac{a_2}{a_1}\dot{x_1}^2 + 2a_2(x_1 - x_2) \leq v_{allow}^2$.

Here, M expresses that the next collision occurs while the cars are both moving. We have also proved that this is essentially optimal.

For more than 2 cars, we have only partial results. Using a simulation, we have produced some very complex bad examples involving 3 vehicles. For more than about 4 cars, we have a partial negative result, showing that, for the simple case above (constant braking, elastic collisions), there are some platoons of that size, with normal parameter values, that are unsafe. This does not mean that large platoons could not be safe; however, it says that such safety would have to depend on something we do not model, e.g., bumpers that crush, or a smart non-constant braking strategy, or some restrictions on the braking capabilities within a platoon.

V. Vehicle Protection Systems

We also modelled vehicle protection systems of the sort used in the Raytheon Personal Rapid Transit (PRT) system [21, 5, 6].

The Raytheon system involves vehicles on a system of tracks, with Y-shaped merges and diverges. Vehicles travel at about 30 mph, with inter-vehicle distances supposedly large enough to prevent collisions, even in the face of "brick-wall" stopping of any of the vehicles. The problem is to ensure various safety conditions, including absence of collisions, safe merges, and prevention of overspeed.

The Raytheon solution, borrowed from mechanical protection systems for trains, uses a Vehicle Protection System (VPS), separate from the main control system. In this system, various *protectors* monitor the vehicles, looking for dangerous situations (e.g., near-collisions, near-overspeed). If such a situation arises, the protector causes the brakes of at least one of the involved vehicles to be applied, in time to prevent the safety violation. This strategy works to ensure safety even if the main control system is faulty.

We developed a general model for vehicle protection systems, and defined and proved general conditions under which they work correctly. This includes consideration of multiple protectors, some of which depend on the results of the others. We proved composition theorems

describing why the combination works correctly. Figure 3 illustrates a physical system with multiple protectors.

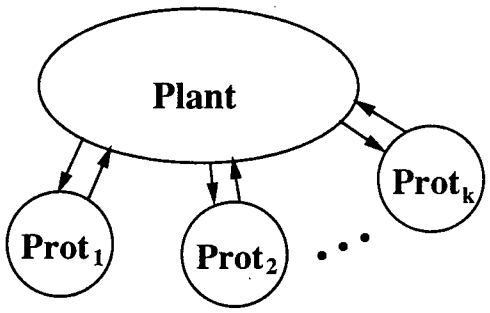


Figure 3: Plant and multiple protectors.

We applied this model and its theorems to prove correctness for a Raytheon-style system, with protectors for overspeed, and for collision on straight tracks and on merge. These protectors are highly interdependent, so we needed to use our composition results. This work made heavy use of composition and used some invariants. It also used levels of abstraction in an interesting way – to relate a vehicle protector implementation to an "abstract protector" in the general model.

VI. Aircraft Collision Avoidance

In [7] we have modelled, and have begun analyzing, the TCAS aircraft collision-avoidance system (TCAS II-7) [4].

The purpose of TCAS is to detect close encounters between aircraft in flight, and help in resolving them safely. Each plane carries its own TCAS system, which detects other planes when they arrive within a distance of approximately 12 miles. The TCAS system provides climb/descend advisories to the pilot. The advisories provided to the pilots of different aircraft must be "compatible". For example, for two aircraft, the advisories must break symmetry, in the sense that one plane should be advised to climb and the other to descend. TCAS uses sensor data providing altitude and range information to help decide which should do which. It also uses mode-S numbers of transponders as unique identifiers, for priority, in order to break symmetry. TCAS also attempts to avoid altitude crossings, and to minimize "reversals" (whereby TCAS changes an advisory already given). In particular, for two aircraft, TCAS II-7 allows a single reversal of direction, by the higher priority plane, if it sees that the physical parameters have changed.

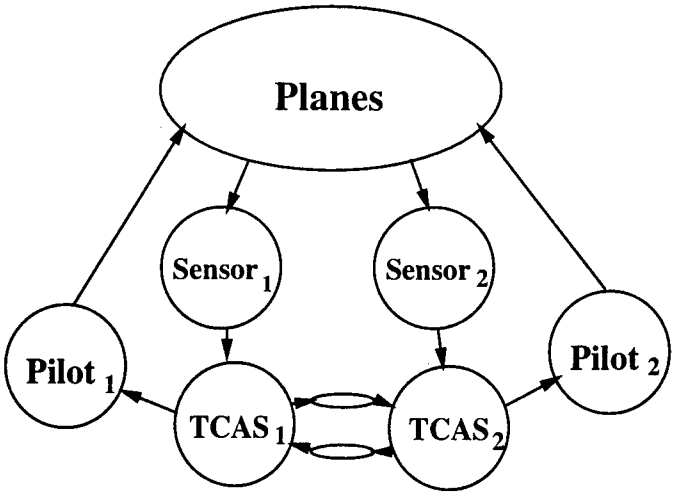


Figure 4: Structure of the TCAS system model.

The TCAS system is quite complex, and is highly safety-critical. Some kind of formal verification is obviously highly desirable.

We have defined automaton models for all the TCAS system components [7]. Some of these models are very abstract; for example, the pilot is just modelled in terms of his response time delay, and the threat detection subsystem is just modelled in terms of guaranteed detection within a certain range. Others of these models are quite detailed, in particular, the resolution advisory generator. The structure of the system is depicted in Figure 4.

We have begun using the model to prove theorems about the behavior of TCAS in flight. For example, we are working on a proof of a conjecture, which we can paraphrase as: Suppose that two planes are both TCAS-equipped, and velocities and accelerations are in normal bounds, and no equipment fails, and pilots follow advisories within at most a short delay. Then TCAS guarantees that the planes remain safely separated (in *all* situations, not just some chosen for simulation). Moreover, in this execution, no reversals occur.

After completing this proof, we will analyze other cases. For example, we will examine the case where pilots delay longer than in the "good" case above, in following their advisories, but they still follow the advisories correctly within the longer delay. We are also interested in cases where a pilot does not follow the advisory but just keeps going in the direction he was following originally. Other interesting cases involve some equipment failure, or more than two planes. The model provides a formal basis for analyzing these cases as well. It provides the ability to make (and prove) definitive conditional claims about what happens

when parameters are in various ranges.

We have also carried out a preliminary analysis of the Center TRACON Automation System (CTAS) [2, 17].

VII. Conclusions and Recommendations

We have adapted some very powerful techniques that have been developed in computer science for modelling and reasoning about complex systems – automaton modelling, parallel composition, invariant assertions, and levels of abstraction (simulation relations) – to the area of hybrid systems, in particular, to automated transportation systems. This work has yielded good models, at various levels of abstraction, and many informative and interesting safety results.

We believe that this project has been extremely successful in contributing new and usable methods for increasing safety assurance for automated transportation systems. We would like to see these methods moved toward practice in transportation system validation. We are happy to do whatever we can to make the methods more usable, and to publicize them widely.

In the future, we plan to complete an analysis of the multiple-collision case for platoons. We also plan to complete a fairly elaborate project on proving properties of TCAS, in order to produce a convincing example to demonstrate the power of our methods to air-traffic control system designers (and validators). We will also complete a definitive paper on the underlying HIOA model and its proof methods. Finally, we will work on the development of computer tools to assist in the application of our model and methods to automated transportation systems and other safety-critical hybrid systems.

See URL http://theory.lcs.mit.edu/tds/trans.html for more information about our project.

References

[1] Michael S. Branicky, Ekaterina Dolginova, and Nancy Lynch. A toolbox for proving and maintaining hybrid specifications. Presented at *HS'96: Hybrid Systems*, October 12-16, 1996, Cornell University, Ithacs, NY.

[2] Darren Cofer, John Maloney, Rosa Weber, George Pappas, Shankar Sastry, John Lygeros, and Nancy Lynch. Formal specification and analysis of the center-TRACON

automation systems (CTAS). Final Report SST-C97-002, Honeywell Technology Center, September 30 1997. Prepared for Langley Research Center and NEXTOR: FAA Center of Excellence in Aviation Operations Research.

[3] Ekaterina Dolginova and Nancy Lynch. Safety verification for automated platoon maneuvers: A case study. In Oded Maler, editor, *Hybrid and Real-Time Systems* (International Workshop, HART'97, Grenoble, France, March 1997), volume 1201 of *Lecture Notes in Computer Science*, pages 154–170. Springer-Verlag, 1997.

[4] Radio Technical Commission for Aeronautics. Minimum operational performance standards for TCAS airborn equipment. Technical Report RTCA/DO-185, RTCA, September 1990. Consolidated Edition.

[5] Carolos Livadas. Formal verification of safety-critical hybrid systems. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, September 1997. Also, Technical Report MIT/LCS/TR-730.

[6] Carolos Livadas and Nancy A. Lynch. Formal framework for modeling and verifying safety-critical hybrid systems. In *Hybrid Systems Computation and Control*, Berkeley, California, April 1998. To appear.

[7] John Lygeros and Nancy Lynch. On the formal verification of the TCAS conflict resolution algorithms. In *36th IEEE Conference on Decision and Control*, San Diego, CA, December 1997. To appear. Extended abstract.

[8] John Lygeros and Nancy Lynch. Conditions for safe deceleration of strings of vehicles. In *Hybrid Systems Computation and Control*, Berkeley, California, April 1998. To appear.

[9] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, March 1996.

[10] Nancy Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. In *Proceedings of the Third AMAST Workshop on Real-Time Systems*, pages 1–22, Salt Lake City, Utah, March 1996.

[11] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O automata. Technical Memo MIT/LCS/TM-544, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, December 1995. Also, [12

[12] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control* (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of *Lecture Notes in Computer Science*, pages 496–510. Springer-Verlag, 1996. Also, [11].

[13] Nancy Lynch and Frits Vaandrager. Forward and backward simulations — Part II: Timing-based systems. *Information and Computation*, 128(1):1–25, July 1996.

[14] Nancy A. Lynch and Mark R. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, Vancouver, British Columbia, Canada, August 1987.

[15] Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, September 1989. Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands. Technical Memo MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, November 1988.

[16] Nancy A. Lynch and Frits W. Vaandrager. Action transducers and timed automata. *Formal Aspects of Computing*, 8(5):499–538, 1996.

[17] George Pappas, Shankar Sastry, John Lygeros, and Nancy Lynch. Modeling, specification, and safety analysis of CTAS. Final report, University of California, September 30 1997. Prepared for Langley Research Center and NEXTOR: FAA Center of Excellence in Aviation Operations Research.

[18] Pravin Varaiya. Smart cars on smart roads: problems of control. *IEEE Transactions on Automatic Control*, AC-38(2):195–207, 1993.

[19] H. B. Weinberg. Correctness of vehicle control systems: A case study. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, February 1996. Also, MIT/LCS/TR-685.

[20] H. B. Weinberg and Nancy Lynch. Correctness of vehicle control systems: A case study. In *17th IEEE Real-Time Systems Symposium*, pages 62–72, Washington, D. C., December 1996.

[21] H. B. Weinberg, Nancy Lynch, and Norman Delisle. Verification of automated vehicle protection systems. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems*

III: Verification and Control (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of *Lecture Notes in Computer Science*, pages 101–113. Springer-Verlag, 1996.

Bibliography of Papers Funded by the Contract

H. B. Weinberg. Correctness of vehicle control systems: A case study. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, February 1996. Also, MIT/LCS/TR-685.
URL http://theory.lcs.mit.edu/tds/papers/Weinberg/SMThesis.html.

H. B. Weinberg and Nancy Lynch. Correctness of vehicle control systems: A case study. In 17th IEEE Real-Time Systems Symposium, pages 62–72, Washington, D. C., December 1996.
URL http://theory.lcs.mit.edu/tds/papers/Weinberg/RTSS96.html.

H. B. Weinberg, Nancy Lynch, and Norman Delisle. Verification of automated vehicle protection systems. In R. Alur, T. Henzinger, and E. Sontag, editors, Hybrid Systems III: Verification and Control (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of Lecture Notes in Computer Science, pages 101–113. Springer-Verlag, 1996.
URL http://theory.lcs.mit.edu/tds/papers/Weinberg/DIMACS95.html.

Carolos Livadas. Formal verification of safety-critical hybrid systems. Masters thesis. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, September 1997. Also, MIT/LCS/TR-730.
URL http://theory.lcs.mit.edu/~clivadas/papers/MEngThesis.html.

Carolos Livadas and Nancy A. Lynch. Formal Verification of safety-critical hybrid systems. In Hybrid Systems Computation and Control, Berkeley, California, April 1998. To appear. URL not available at this time.

Ekaterina Dolginova and Nancy Lynch. Safety verification for automated platoon maneuvers: A case study. In Oded Maler, editor, Hybrid and Real-Time Systems (International Workshop, HART'97, Grenoble, France, March 1997), volume 1201 of Lecture Notes in Computer Science, pages 154–170. Springer-Verlag, 1997.
URL http://theory.lcs.mit.edu/~katya/safety.html.

Michael S. Branicky, Ekaterina Dolginova, and Nancy Lynch. A toolbox for proving and maintaining hybrid specifications. Presented at HS'96: Hybrid Systems, October 12-16, 1996, Cornell University, Ithacs, NY.
URL http://theory.lcs.mit.edu/~katya/toolbox.html.

Nancy Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. In Proceedings of the Third AMAST Workshop on Real-Time Systems, pages 1–22, Salt

Lake City, Utah, March 1996.
URl http://theory.lcs.mit.edu/tds/three-level.html.

Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In R. Alur, T. Henzinger, and E. Sontag, editors, Hybrid Systems III: Verification and Control (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of Lecture Notes in Computer Science, pages 496–510. Springer-Verlag, 1996.
URl http://theory.lcs.mit.edu/tds/papers/Lynch/LSVW.html.

Nancy Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. In R.Alur, T.Henzinger, and E.Sontag, editors, Hybrid Systems III: Verification and Control (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of Lecture Notes in Computer Science, pages 449–463. Springer-Verlag, 1996.
URl http://theory.lcs.mit.edu/tds/papers/Lynch/transit-survey.html.

Nancy Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. Technical Memo MIT/LCS/TM-545, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, December 1995.
URl http://theory.lcs.mit.edu/tds/papers/Lynch/transit-survey.html.

John Lygeros and Nancy Lynch. On the formal verification of the TCAS conflict resolution algorithms. In 36th IEEE Conference on Decision and Control, San Diego, CA, December 1997. To appear. Extended abstract.
URl http://theory.lcs.mit.edu/tds/TCAS.html.

Darren Cofer, John Maloney, Rosa Weber, George Pappas, Shankar Sastry, John Lygeros, and Nancy Lynch. Formal specification and analysis of the Center TRACON Automation Systems (CTAS). Final Report SST-C97-002, Honeywell Technology Center, September 30 1997. Prepared for Langley Research Center and NEXTOR: FAA Center of Excellence in Aviation Operations Research.

George Pappas, Shankar Sastry, John Lygeros, and Nancy Lynch. Modeling, specification. and safety analysis of CTAS. Final report, University of California, September 30 1997 Prepared for Langley Research Center and NEXTOR: FAA Center of Excellence in Aviation Operations Research.

Attachments

1. H. B. Weinberg and Nancy Lynch. Correctness of vehicle control systems: A case study. In *17th IEEE Real-Time Systems Symposium*, pages 62–72, Washington, D. C., December 1996.

2. Ekaterina Dolginova and Nancy Lynch. Safety verification for automated platoon maneuvers: A case study. In Oded Maler, editor, *Hybrid and Real-Time Systems* (International Workshop, HART'97, Grenoble, France, March 1997), volume 1201 of *Lecture Notes in Computer Science*, pages 154–170. Springer-Verlag, 1997.

3. Carolos Livadas, Nancy A. Lynch, and H. B. Weinberg. Formal Verification of Safety-Critical Hybrid Systems. In *Hybrid Systems Computation and Control*, Berkeley, California, April 1998. To appear.

4. Nancy Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. In *Proceedings of the Third AMAST Workshop on Real-Time Systems*, pages 1–22, Salt Lake City, Utah, March 1996.

5. John Lygeros and Nancy Lynch. On the formal verification of the TCAS conflict resolution algorithms. In *36th IEEE Conference on Decision and Control*, San Diego, CA, December 1997.

Correctness of Vehicle Control Systems – A Case Study

H. B. Weinberg and Nancy Lynch
MIT
Laboratory for Computer Science
Cambridge, MA 02139, USA

Abstract

Several example vehicle deceleration maneuvers arising in automated transportation systems are specified, and their correctness verified, using the hybrid I/O automaton model of Lynch, Segala, Vaandrager and Weinberg [16]. All system components are formalized using hybrid I/O automata, and their combination described using automaton composition. The proofs use invariant assertions, simulation mappings, and differential calculus.

1 Introduction

A *hybrid system* is one in which digital and analog components interact. Typical examples of hybrid systems are real-time process-control systems such as automated factories or automated transportation systems, in which the digital components monitor and control continuous physical processes in the analog components. The computer science community has developed formal models and methods for reasoning about digital systems, while the control theory community has done the same for analog systems. However, systems that combine both types of activity appear to require new methods. The development and application of such methods is an active area of current research.

One formal tool that has recently been developed is the *hybrid I/O automaton* (HIOA) model [16]. In this case study, we show how the HIOA model can be used to specify and verify part of an automated transportation system — a vehicle deceleration maneuver. The methods we use include computer-science-based techniques such as automaton composition, invariant assertions, and simulation mappings, as well as simple continuous analysis. The purpose of the case study is to investigate the applicability of the HIOA model and various computer-science-based techniques to automated transportation systems in particular, and to hybrid systems in general. We are especially concerned that the methods allow faithful representation of hybrid systems (including all components), and clear and scalable proofs of significant properties of these systems.

The hybrid I/O automaton model is an extension of the *timed I/O automaton* model of [17, 4], inspired by the phase transition system model of [19] and the similar hybrid system model of [1]. A HIOA is a (possibly) infinite state labelled transition system. The *states* of a HIOA are the valuations of a set of *variables*. Certain states are distinguished as *start* states. The *transitions* (*steps*) of a HIOA are of two types: *discrete* and *continuous*. The discrete transitions are labelled with *actions*. Both the variables and the actions are partitioned into three categories: *input*, *output*, and *internal*. A *hybrid execution* of a HIOA is a sequence of transitions that describes a possible behavior of the system over time. A *hybrid trace* is the externally visible part of an execution (i.e., the non-internal part).

We say that one HIOA *implements* a second HIOA if the set of traces of the first is a subset of that of the second. This captures the notion that the implementation HIOA has no external behavior that is not allowed by the specification HIOA. When two HIOAs are *composed* in parallel, they synchronize on shared input/output actions and shared input/output variables. Under certain easily checked conditions, the parallel composition of two HIOAs is itself a HIOA. An important property of HIOAs is *substitutivity*: in a system composed of HIOAs, replacing components by implementations of those components yields an implementation of the entire system.

As has been the case in previous work with timed I/O automata, most of the proofs in this HIOA-based case study use *invariant assertions* and *simulation mappings*. An invariant assertion is a predicate on states that is true in every reachable state. Invariant assertions are usually proved by induction on the length of an execution. A simulation is a mapping between states of two HIOAs that can be used to show that one HIOA implements another. The proof that a given mapping is a simulation is also an induction on the length of an execution of the implementation; the inductive step matches individual transitions in the implementation with corresponding transitions or sequences of transitions in the specification. Even timing properties can be proved us-

ing these techniques: the key idea is to build timing information into the state where it can be tested by assertions.

Our methods have several benefits. First, the HIOA model and its composition operation permit complete representation of hybrid systems, including all components, continuous and discrete, and the interactions among them. Second, the inductive structure and stylized nature of the proofs make them easy to write, check, and understand. In previous work, such proofs have even been checked using automated theorem proving techniques. Third, the implementation relation allows the description of a system at different levels of abstraction. Assertions proved for high level models extend to the lower level models via the simulation mappings. This hierarchy helps manage the complexity of the overall system description, and it helps simplify the proofs because assertions are usually easier to prove on the more abstract models. Fourth and finally, the methods are not completely automatic. They require the user to supply invariants and simulations, which express key insights about the system and serve as useful documentation.

Typical examples of automated transportation systems include the Raytheon Personal Rapid Transit System and the California PATH project [6, 5, 13]. In these hybrid systems, a number of computer-controlled vehicles share a network of tracks or highways. The digital part of the system is the computer vehicle controller and the analog part of the system consists of the vehicle, its engine, the guideway, and so forth. In [6], the control of the transportation system is described hierarchically – the higher levels coordinate and determine strategy while the lowest level performs specific maneuvers.

Our case study focuses on a single maneuver: the task of decelerating a vehicle to a target speed within a given distance. Such a maneuver is invoked, for example, when a vehicle is approaching a region whose maximum allowable velocity is lower than the vehicle's current velocity. We model a vehicle and its controller as two communicating HIOAs. We consider four different sets of assumptions about the communication between vehicle and controller, based on whether or not there is feedback from the vehicle to the controller and whether or not there is communication delay from the controller to the vehicle. For each case, we give a formal specification of what it means for a controller to correctly implement the deceleration maneuver, we give an example implementation of such a controller, and we verify that the implementation is correct. All of our proofs use invariant assertions, including assertions involving timing properties, and some also use simulation mappings. Discrete and continuous methods are combined smoothly, and uncertainty is integrated throughout the presentation.

Our contributions are (a) The complete modelling and proof of the four maneuvers. (b) Many intermediate formal concepts and lemmas that can be reused in formal reasoning about other automated transit systems. (c) A demonstration of the effectiveness of our computer-science-based methods for reasoning about hybrid systems.

This case study is part of a larger project on modelling, verifying, and analyzing problems arising in automated transit systems. A survey of the early results of that project appears in [14]. A preliminary study of the Generalized Railroad Crossing problem appears in [7, 8]; this uses only the timed I/O automaton model, not the HIOA model. In [15], levels of abstraction are used to relate continuous and discrete control of a vehicle maneuver, as well as to relate derivative-based and function-based system descriptions. Safety assurance systems for automated transit are examined in [27]. Current work involves modelling the "platoon join" maneuver from the PATH project [3], as well as continuing the project on safety assurance systems.

The development of models and verification methods for timing-based systems is an active research area within computer science. The timed I/O automaton model is similar, for example, to models of Alur and Dill [2], of Lamport [10] and of Henzinger, Manna and Pnueli [9]. In contrast to those formalisms, the development and use of the timed I/O automaton model has focused on compositional properties [24], implementation relations [17, 23], and semi-automated proof checking [12], with less emphasis on syntactic forms, temporal logics, and fully automatic analysis. Just as timed I/O automata have been extended to hybrid I/O automata to treat hybrid systems, so have other real-time models. For example, the timed transition system model of [9] is extended to the phase transition system model in [19]. Phase transition systems are analogous to hybrid I/O automata: their transitions correspond to our discrete steps and their activities correspond to our trajectories. However, phase transition systems lack good support for composition and abstraction. The hybrid system model of [1] is similar to the phase transition system model except that it includes synchronization labels that correspond to our actions. This allows a notion of parallel composition. The hybrid system model differs from our HIOA model because it has no input/output distinction on either labels (actions) or variables.

The methods of invariant assertions and simulation mappings are widely used in computer science. An overview of these methods, for untimed and timed systems, appears in [18, 17].

Another project involving formal modelling of train control systems, using computer science techniques, was carried out by Schneider and co-workers [20]. Their emphasis was on the use of an extension of Dijkstra's weakest-precondition calculus to *derive* correct solutions. Other case studies in modelling hybrid systems include two analyses of steam boiler controllers — one using timed I/O automaton methods [11] and another using the automated proof checker PVS [25] — and a project using a variety of techniques to

model and verify controllers for aircraft landing gear [22]. This latter reference also includes examples from automated transportation.

The full version of this work appears in [26].

2 Hybrid I/O Automaton Model

The hybrid I/O automaton model [16] is based on the timed I/O automaton model of [17, 4], but it represents continuous behavior more explicitly. We give a brief summary here, and refer the reader to [16] for the details.

A *state* of a HIOA is defined to be a valuation of a set of variables. A *trajectory* w is a function that maps a left-closed interval I of the reals, with left endpoint equal to 0, to states; a trajectory represents the continuous evolution of the state over an interval of time. A trajectory with domain $[0, 0]$ is called a *point* trajectory. Various operations are defined on trajectories, including restriction to a subset of the domain ($\lceil$), projection on a subset of the state variables ($\downarrow$), and concatenation.

A *hybrid I/O automaton (HIOA)* $A = (U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, \mathcal{D}, \mathcal{W})$ consists of:

- Three disjoint sets U, X and Y of variables, called *input*, *internal* and *output* variables, respectively. Variables in $E \triangleq U \cup Y$ are called *external*, and variables in $L \triangleq X \cup Y$ are called *locally controlled*. We write $V \triangleq U \cup L$.

- Three disjoint sets Σ^{in}, Σ^{int}, Σ^{out} of *input*, *internal* and *output actions*, respectively. We assume that Σ^{in} contains a special element e, the *environment action*, which represents the occurrence of a discrete transition outside the system that is unobservable, except (possibly) through its effect on the input variables. Actions in $\Sigma^{ext} \triangleq \Sigma^{in} \cup \Sigma^{out}$ are called *external*, and actions in $\Sigma^{loc} \triangleq \Sigma^{int} \cup \Sigma^{out}$ are called *locally controlled*. We write $\Sigma \triangleq \Sigma^{in} \cup \Sigma^{loc}$.

- A nonempty set Θ of *start states*, a subset of the set of states. This set must be closed under change of values for input variables.

- A set $\mathcal{D}$ of *discrete transitions*, i.e., (state, action, state) triples. This set must satisfy three axioms, saying that input actions are always enabled, that the environment action e only affects inputs, and that any input variable may change when any discrete action occurs. We use $s \xrightarrow{a} s'$ as shorthand for $(s, a, s') \in \mathcal{D}$.

- A set $\mathcal{W}$ of trajectories over the variables of A. This set must satisfy three axioms, asserting existence of point trajectories for all states, and closure of the set of trajectories under subinterval and limit.

When discussing several HIOAs, we often subscript the names of the various components with the name of the HIOA.

We now define executions for HIOAs. A *hybrid execution fragment* of A is a finite or infinite alternating sequence of trajectories and actions, $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$, ending with a trajectory if α is a finite sequence, and with discrete steps connecting consecutive pairs of trajectories, labelled by the intervening actions. An execution fragment records all the discrete changes that occur in an evolution of a system, plus the "continuous" state changes that take place in between. A *hybrid execution* is an execution fragment in which the first state is a start state. A state of A is defined to be *reachable* if it is the last state of some finite hybrid execution of A.

The visible behavior of a HIOA is described in terms of its "hybrid traces". The *hybrid trace* of a hybrid execution is obtained by projecting the trajectories on the external variables, replacing all the internal actions that cause changes in the external state by a special placeholder τ, and removing all the internal actions that cause no such changes. (In this last case, the surrounding trajectories are concatenated.)

HIOAs A and B are *comparable* if they have the same external actions and external variables. If A and B are comparable then we say that $A \leq B$ provided that the set of hybrid traces of A is a subset of that of B. In this case, we say that A *implements* B.

We next define simulation mappings for HIOAs; these are used to describe systems using different levels of abstraction. Let A and B be comparable HIOAs. A *simulation* from A to B is a relation R from states of A to states of B satisfying:

1. If $s_A \in \Theta_A$ then there exists $s_B \in \Theta_B$ such that $s_A \, R \, s_B$.

2. If $s_A \xrightarrow{a}_A s'_A$, $s_A \, R \, s_B$, and both s_A and s_B are reachable, then B has a finite execution fragment starting with s_B, having the same trace as the given step, and ending with a state s'_B with $s'_A \, R \, s'_B$.

3. If w_A is a trajectory of A from s_A to s'_A, $s_A \, R \, s_B$, and both s_A and s_B are reachable, then B has a finite execution fragment starting with s_B, having the same trace as w, and ending with a state s'_B with $s'_A \, R \, s'_B$.

The importance of simulations is given by the following theorem.

Theorem 2.1 *If A and B are comparable HIOAs and there is a simulation from A to B, then $A \leq B$.*

Finally, we define composition and hiding operations for HIOAs. We say that HIOAs A and B are *compatible* if they have no output actions or output variables in common, and

if no internal variable of either is a variable of the other. If A and B are compatible then their *composition* is defined to be the tuple $(U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, \mathcal{D}, \mathcal{W})$ given by

- $U = (U_A \cup U_B) - (Y_A \cup Y_B)$, $X = X_A \cup X_B$, and $Y = Y_A \cup Y_B$.
- $\Sigma^{in} = (\Sigma_A^{in} \cup \Sigma_B^{in}) - (\Sigma_A^{out} \cup \Sigma_B^{out})$, $\Sigma^{int} = \Sigma_A^{int} \cup \Sigma_B^{int}$, and $\Sigma^{out} = \Sigma_A^{out} \cup \Sigma_B^{out}$.
- Θ is the set of states s such that $s\lceil V_A \in \Theta_A \wedge s\lceil V_B \in \Theta_B$.
- $\mathcal{D}$ is the set of triples (s, a, s') such that $s\lceil V_A \xrightarrow{\pi_A(a)}_A s'\lceil V_A \quad \wedge \quad s\lceil V_B \xrightarrow{\pi_B(a)}_B s'\lceil V_B$. (Here, $\pi_A(a)$ is defined to be a if a is an action of A and e otherwise; analogously for B. $\lceil$ denotes restriction to a subset of the variables.)
- $\mathcal{W}$ is the set of trajectories w such that $w \downarrow V_A \in W_A \wedge w \downarrow V_B \in W_B$. (Here, $\downarrow$ denotes projection on a subset of the variables.)

The parallel composition of A and B is itself a HIOA. The following theorem says that a component can be replaced by an implementation in a composition.

Theorem 2.2 *Suppose A_1, A_2 and B are HIOAs with $A_1 \leq A_2$, and each of A_1 and A_2 is compatible with B. Then $A_1 \| B \leq A_2 \| B$.*

Two hiding operations can be defined on any HIOA, one that hides a designated subset of the output actions and one for a designated subset of the output variables. The hiding operators also interact properly with the implementation relation.

3 Case 1: No Delay or Feedback

In the deceleration problem we consider a computer-controlled train moving along a track. The task of the train's controller is to slow the train within a given distance. In this section we consider a very simple model of the train and the controller. The train has two modes, braking and not braking. The controller can effect an instant change in the mode of the train (relaxed in Sections 4 and 6). The controller receives no information from the train (relaxed in Sections 5 and 6). The braking strength of the train varies nondeterministically within known bounds. We model both the train and the controller as hybrid I/O automata.

In the following subsections we describe the parameters of the specification, give a hybrid I/O automaton model for the train, define correctness of a controller for this train, give an example correct controller, and prove that it is correct.

Parameters All the parameters are constants denoted by c with some dots above it and a subscript. Dots above the constant identify the type of the constant: position (no dots), velocity (one dot), or acceleration (two dots). These dots are just a syntactic device – they do not represent differentiation. The subscript identifies the particular constant. Initial values of the train's position, velocity and acceleration are c_s, $\dot{c}_s$, and $\ddot{c}_s$. The goal of the deceleration maneuver is to slow the train to a velocity in the interval $[\dot{c}_{minf}, \dot{c}_{maxf}]$ at position c_f. When the train is not braking its acceleration is exactly 0. When the train is braking, its acceleration varies nondeterministically between $[\ddot{c}_{min}, \ddot{c}_{max}]$, both negative. The range is intended to model inherent uncertainty in brake performance. We impose the following constraints on the parameters:

1. $c_s < c_f$
2. $\dot{c}_s > \dot{c}_{maxf} \geq \dot{c}_{minf} > 0$
3. $\ddot{c}_s = 0$
4. $\ddot{c}_{min} \leq \ddot{c}_{max} < 0$
5. $c_f - c_s \geq \frac{\dot{c}_{maxf}^2 - \dot{c}_s^2}{2\ddot{c}_{max}}$
6. $\frac{\dot{c}_{maxf} - \dot{c}_s}{\ddot{c}_{max}} \leq \frac{\dot{c}_{minf} - \dot{c}_s}{\ddot{c}_{min}}$

The first three constraints just say that the initial position is before the final position, that the initial velocity is higher than the target velocities which are positive, and that the initial acceleration is 0. Since braking is stronger when acceleration is more negative, notice in the fourth constraint that $\ddot{c}_{min}$ is the strongest braking strength, and $\ddot{c}_{max}$ the weakest. The fifth constraint ensures that with the weakest possible braking there is still enough distance to reach the highest allowable speed by position c_f. The right hand side of this equation uses a familiar equation for "change in distance for change in velocity" from constant acceleration Newtonian physics. To understand the sixth constraint consider that since the controller receives no sensory information from the train, it must decide *a priori* how long to brake. The sixth constraint ensures that the least amount of time the controller must brake is less than the greatest amount of time that it can brake.

The TRAIN Automaton We model the train as the HIOA TRAIN represented in Table 1. The train's physical state is modelled using three variables: x, $\dot{x}$, and $\ddot{x}$. As before, the dots are a syntactic device; the fact there there is a differential relationship between the evolution of these variables is a consequence of the definition of the trajectory set for TRAIN. The train accepts commands to turn the brake on or off through discrete actions `brakeOn` and `brakeOff`.

It stores the state of the brake in variable b. While braking, the train applies an acceleration that is nondeterministically chosen at every point but is constrained to be an integrable function with range in the interval $[\ddot{c}_{\min}, \ddot{c}_{\max}]$. While not braking, the train has acceleration exactly 0. The variable *now* represents the current time; when using assertions to reason about the timing behavior of systems, it is convenient to have an explicit state variable that records the current time. At this point in [26], we prove various fundamen-

Actions:
Input: `brakeOn` and `brakeOff`
Vars:
Output: $x \in \mathbb{R}$, initially $x = c_s$
$\dot{x} \in \mathbb{R}$, initially $\dot{x} = \dot{c}_s$
$\ddot{x} \in \mathbb{R}$, initially $\ddot{x} = \ddot{c}_s$
b, a boolean, initially `false`
$now \in \mathbb{R}^{\geq 0}$, initially 0
Discrete Transitions:
`brakeOn`:
Eff: $b :=$ `true`
$\ddot{x} :\in [\ddot{c}_{\min}, \ddot{c}_{\max}]$
`brakeOff`:
Eff: $b :=$ `false`
$\ddot{x} := 0$
Trajectories:
if $w(0).b =$ `true` then
$w.\ddot{x}$ is an integrable function with range $[\ddot{c}_{\min}, \ddot{c}_{\max}]$
else $w.\ddot{x} = 0$
for all $t \in I$ the following hold:
$w(t).b = w(0).b$
$w(t).now = w(0).now + t$
$w(t).\dot{x} = w(0).\dot{x} + \int_0^t w(s).\ddot{x}\, ds$
$w(t).x = w(0).x + \int_0^t w(s).\dot{x}\, ds$

Table 1. The TRAIN automaton.

tal facts about the mechanics of the train. Most of these facts relate the initial state and final states of a trajectory. Here, we give two examples of such lemmas. The first bounds change in velocity and position by change in time. The second bounds change in position by change in velocity. (Notation: If s and s' are states and x is a variable, we often write x for $s.x$ and x' for $s'.x$ when s and s' are understood.)

Lemma 3.1 *Let w be a trajectory of* TRAIN *whose initial and final states are s and s', respectively, and let $\Delta = now' - now$. If $b =$ `true` then:*

1. $\dot{x} + \ddot{c}_{\min}\Delta \leq \dot{x}' \leq \dot{x} + \ddot{c}_{\max}\Delta$
2. $x + \dot{x}\Delta + \frac{1}{2}\ddot{c}_{\min}\Delta^2 \leq x' \leq x + \dot{x}\Delta + \frac{1}{2}\ddot{c}_{\max}\Delta^2$

Lemma 3.2 *Let w, s, s', and Δ be as in the previous lemma. If $b =$ `true` then:*

$$\frac{(\dot{x}')^2 - \dot{x}^2}{2\ddot{c}_{\min}} \leq x' - x \leq \frac{(\dot{x}')^2 - \dot{x}^2}{2\ddot{c}_{\max}}.$$

The train considered here is simple; in a treatment of a system with more complex dynamics, the lemmas of this section would be replaced by more complex lemmas of the same general form. Such lemmas would be derived using methods of continuous mathematics appropriate for the application.

Definition of Controller Correctness We define a *brake-controller* to be a HIOA with no external variables, no input actions, and output actions `brakeOn` and `brakeOff`. A *correct* brake-controller is one that when composed with TRAIN, yields a HIOA whose hybrid traces satisfy:

Safety In all reachable states: If $x = c_f$ then $\dot{c}_{\text{minf}} \leq \dot{x} \leq \dot{c}_{\text{maxf}}$. (That is, if the train ever reaches position c_f then the speed is in the desired range.)

Timeliness There exists $t \in \mathbb{R}^{\geq 0}$ such that: Any execution containing a state with $now = t$ also contains a state in which $x = c_f$. (That is, the train must reach c_f within time t.)

The following lemma says that the safety and timeliness properties are preserved by the implementation relation; in other words, an implementation of a correct brake-controller is itself a correct brake-controller.

Lemma 3.3 *If $A_1 \leq A_2$ and A_2 is a correct brake-controller, then A_1 is a correct brake-controller.*

Proof: Follows from Theorem 2.2 and the definition of correctness. ■

Example Controller: ONE-SHOT There is a broad spectrum of correct controllers one could consider, from fully deterministic to highly nondeterministic, and involving any number of applications of the brake. In this section we consider a correct brake-controller called ONE-SHOT. ONE-SHOT applies the brake exactly once, i.e., it performs exactly one `brakeOn` action followed by exactly one `brakeOff` action. Except for this restriction, ONE-SHOT is highly nondeterministic: it exhibits all the correct braking strategies that involve exactly one application of the brake.

We chose ONE-SHOT as an example because (a) it is simple, (b) its behavior is interesting enough to require some interesting proof techniques, and (c) it can be used to help verify correctness of the more complicated controller given in Section 4, using a simulation proof and Lemma 3.3.

We define some more constants:

$$A = \frac{1}{\dot{c}_s}\left(c_f - c_s - \frac{\dot{c}_{\text{maxf}}^2 - \dot{c}_s^2}{2\ddot{c}_{\max}}\right)$$

$$B = \frac{\dot{c}_{\text{maxf}} - \dot{c}_s}{\ddot{c}_{\max}}$$

$$C = \frac{\dot{c}_{\text{minf}} - \dot{c}_s}{\ddot{c}_{\min}}$$

A represents the longest amount of time a correct controller can wait before applying the brake. B and C are lower and upper bounds, respectively, on the amount of time a correct controller should apply the brake if it only brakes once. These constants are derived using methods of continuous analysis. The formal description of ONE-SHOT appears in Table 2. (Notation: Each "task" is a set of actions that comes equipped with lower and upper bound values on the time required for some action of the task to occur, if any actions of the task are enabled.)

Actions:
  Output: `brakeOn` and `brakeOff`
Vars:
  Internal: *phase* $\in$ {`idle`, `braking`, `done`}, initially `idle`
Discrete Transitions:
  `brakeOn`:
    Pre: *phase* = `idle`
    Eff: *phase* := `braking`
  `brakeOff`:
    Pre: *phase* = `braking`
    Eff: *phase* := `done`
Tasks: ON = {`brakeOn`} : $[0, A]$
  OFF = {`brakeOff`} : $[B, C]$

Table 2. The ONE-SHOT automaton

An execution of ONE-SHOT consists of three phases: `idle`, `braking`, and `done`. ONE-SHOT waits between 0 and A time units (`idle` phase), then applies the brake for at least B and at most C time units (`braking` phase), and then disengages the brake (`done` phase). The *ON* task governs the transitions from `idle` to `braking` and the *OFF* task governs the transitions from `braking` to `done`.

The notation used above is based on [21]. In order to convert this description to a HIOA, the time constraints for the tasks must be built into the automaton's states, transitions and trajectories. We do this by incorporating *deadline variables* $last(ON)$, $first(OFF)$ and $last(OFF)$ into the state, and manipulating them so that the `brakeOn` and `brakeOff` actions occur at allowed times. That is, initially $last(ON) = A$. When `brakeOn` occurs, $first(OFF)$ and $last(OFF)$ are set to times B and C in the future, respectively. ONE-SHOT does not allow time to pass beyond any *last* deadline currently in force, and does not allow a `brakeOff` action to occur if its *first* deadline has not yet been reached. The trajectories are simple – there is no interesting continuous behavior in the controller, so time just passes without changing anything else.

The entire system is modelled formally as the composition of the two HIOAs, TRAIN and ONE-SHOT, which we call ONE-SHOT-SYS.

Correctness of ONE-SHOT At this point in [26], we prove the correctness of the ONE-SHOT controller. In the process of doing this, we prove a variety of properties about ONE-SHOT-SYS, almost all of which take the form of invariant assertions. Some of these assertions involve the deadline variables $last(ON)$, $first(OFF)$ and $last(OFF)$, i.e., they encode claims about timing behavior. These proofs demonstrate the clarity, simplicity and power of the assertional proof style.

Here, we restrict ourselves to two key lemmas that illustrate our use of invariant assertions and deadline variables. The first lemma is used in the proof of the safety property, which says that the following is an invariant of the system:

$$x = c_f \Longrightarrow \dot{c}_{minf} \leq \dot{x} \leq \dot{c}_{maxf}.$$

In particular, we focus on the right hand side of the inequality, $\dot{x} \leq \dot{c}_{maxf}$. In order to prove this invariant, we prove a stronger invariant:

$$x \leq c_f \Longrightarrow c_f - x \geq \frac{\dot{c}_{maxf}^2 - \dot{x}^2}{2\ddot{c}_{max}}.$$

This invariant says that before reaching the final position there must be enough distance left to brake, even at the weakest braking. It has as a special case the upper bound needed in the safety property (note that $\ddot{c}_{max}$ is negative). In [26], we demonstrate this invariant for each phase separately and combine the results into a global invariant. Here we present only the result for the braking phase:

Lemma 3.4 *In all reachable states of* ONE-SHOT-SYS, *if* *phase* = `braking` *then* $c_f - x \geq \frac{\dot{c}_{maxf}^2 - \dot{x}^2}{2\ddot{c}_{max}}$.

Proof: By induction on the length (number of discrete steps and trajectories) in an execution. The inductive steps break down into separate cases for discrete steps and trajectories. The interesting cases are the *ON* steps and those trajectories in which *phase* = `braking`. The *ON* case follows from the invariant for the *idle* phase. In the trajectory case, we substitute from Lemma 3.2 into the inductive hypothesis and simplify:

$c_f - x$	$\geq$	$\frac{\dot{c}_{maxf}^2 - \dot{x}^2}{2\ddot{c}_{max}}$	inductive hypothesis
$x' - x$	$\leq$	$\frac{(\dot{x}')^2 - \dot{x}^2}{2\ddot{c}_{max}}$	Lemma 3.2
$c_f - x'$	$\geq$	$\frac{\dot{c}_{maxf}^2 - (\dot{x}')^2}{2\ddot{c}_{max}}$	subtract

■

The second lemma is used in the proof of the timeliness property. It says that the brake must be disengaged before the velocity has a chance to drop below $\dot{c}_{minf}$, even assuming the strongest deceleration. Symmetrically, the brake cannot be disengaged until after the velocity is guaranteed to reach $\dot{c}_{maxf}$, even assuming the weakest deceleration. Note the use of the deadline variables $first(OFF)$ and $last(OFF)$ in these assertions. For example, the expression $last(OFF) - now$ indicates the greatest amount of time the controller can continue braking.

Lemma 3.5 *In all reachable states of* ONE-SHOT-SYS, *if* $phase = \texttt{braking}$ *the following hold:*

1. $last(OFF) - now \leq \frac{\dot{c}_{minf} - \dot{x}}{\ddot{c}_{min}}$
2. $first(OFF) - now \geq \frac{\dot{c}_{maxf} - \dot{x}}{\ddot{c}_{max}}$

Proof: By induction. ■

Theorem 3.6 ONE-SHOT *is a correct brake-controller.*

This simple example already illustrates several aspects of our model and methods: It shows how vehicles and controllers can be modelled using HIOAs and composition, and in particular, how deadline variables can be used to express timing restrictions. It shows some typical correctness conditions, expressed in terms of the real-world component of the system. It shows how invariants can provide the keys to proofs. Invariants can involve real-valued quantities representing real-world behavior, thus allowing facts about velocities, etc., to be proved using induction; invariants can also involve deadline variables, thus allowing time bounds to be proved by induction.

This proof combines discrete and continuous reasoning within a rigorous framework that helps to ensure that the combination is well-defined and the reasoning sound. The proofs of invariants break down into separate cases involving discrete and continuous reasoning. The example also illustrates careful handling of uncertainty. Finally, the arguments are general – they handle all cases, and are not based on identifying the apparent worst cases.

4 Case 2: Delay and No Feedback

In this section we extend the model of the train by nondeterministically delaying the braking commands. Rather than modify the train automaton itself, we introduce a new automaton called BUFFER that serves as a buffer between the train and a controller. Figure 1 illustrates the components and their communication.

In the following sections we present BUFFER, modify the controller correctness criteria to account for the BUFFER, give an example controller called DEL-ONE-SHOT, and prove that it is correct. The proof uses a simulation mapping to show that the composition of DEL-ONE-SHOT and BUFFER implements ONE-SHOT; the correctness of DEL-ONE-SHOT then follows from Theorem 3.6 and Lemma 3.3.

The BUFFER **Automaton** The buffer stores a single command from the controller. It forwards it to the train after some delay. For each command, the delay is nondeterministically chosen from the interval $[\delta^-, \delta^+]$ (where $0 \leq \delta^- \leq \delta^+$).

Actions:
- Inputs: `bufBrakeOn` and `bufBrakeOff`
- Outputs: `brakeOn` and `brakeOff`

Vars:
- Internal: *request* $\in$ {on, off, none}, initially none
  violation, boolean, initially `false`

Discrete Transitions:

`bufBrakeOn`:
- Eff: Cases of *request*,
  - on : no effect
  - off : *violation* := `true`
  - none : *request* := on

`bufBrakeOff`:
- Eff: Cases of *request*,
  - on : *violation* := `true`
  - off : no effect
  - none: *request* := `off`

`brakeOn`:
- Pre: *request* = on
- Eff: *request* := none

`brakeOff`:
- Pre: *request* = `off`
- Eff: *request* := none

Tasks:
BUFF = {`brakeOn`, `brakeOff`} : $[\delta^-, \delta^+]$

Table 3. The BUFFER **automaton.**

The BUFFER automaton appears in Table 3. The variable *request* stores a command while it is being buffered. The history variable *violation* records when a new command arrives from the controller before the previous one has exited the buffer; this is considered to be an error condition.

Definition of Controller Correctness, Revisited We modify the definition of a correct controller to account for the buffer. We define a *buffered-brake-controller* to be a HIOA with no external variables, no input actions, and output actions `bufBrakeOn` and `bufBrakeOff`. A *correct* buffered-brake-controller is one that, when composed with BUFFER, with the `bufBrakeOn` and `bufBrakeOff` actions hidden, yields a correct brake-controller, as defined in Section 3.

Parameters, Revisited Not only do we need to place restrictions on the value of the new parameters (δ^-, δ^+), but we also need to revise the constraints among the original parameters. Now the controller is subject to more uncertainty and therefore cannot achieve conformance to as tight a target velocity range. The further constraints on the parameters can be viewed as forcing the target velocity range, $[\dot{c}_{minf}, \dot{c}_{maxf}]$ to be wider and hence the controller's task easier. These are the additional constraints:

1. $0 \leq \delta^- \leq \delta^+$
2. $\dot{c}_s \geq \dot{c}_{maxf} + \ddot{c}_{max}\delta^+$

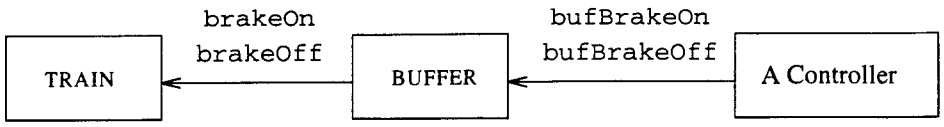


Figure 1. Overview of Delay Deceleration Model

3. $\dot{c}_{\text{maxf}} \geq \dot{c}_{\text{minf}} + \ddot{c}_{\text{min}}\delta^+$
4. $\frac{\dot{c}_{\text{maxf}} - \dot{c}_s}{\ddot{c}_{\text{max}}} + \delta^+ - \delta^- \leq \frac{\dot{c}_{\text{minf}} - \dot{c}_s}{\ddot{c}_{\text{min}}} - \delta^+ + \delta^-$

The first constraint ensures that the delay interval is well-defined. The next two are necessary to ensure that the buffer does not overflow. The last constraint replaces constraint 6 in Section 3; the new version accounts not only for the nondeterminism of the braking strength but also for that of the buffer. The other five original constraints remain as well but are not shown here. Note that these constraints in this section are more restrictive than the constraints from Section 3.

Example Controller: DEL-ONE-SHOT Here we give an example of a valid buffered-brake-controller called DEL-ONE-SHOT. This automaton is identical to ONE-SHOT of Section 3 except in the names of its actions and the duration of its phases. The output actions `brakeOn` and `brakeOff` are replaced by `bufBrakeOn` and `bufBrakeOff`. The time bounds A, B, C are replaced by A', B', C'. These new bounds are:

$$A' = \max(0, A - \delta^+)$$
$$B' = B + \delta^+ - \delta^-$$
$$C' = C - \delta^+ + \delta^-$$

We define DEL-ONE-SHOT-AND-BUF to be the composition of DEL-ONE-SHOT and BUFFER, with the `bufBrakeOn` and `bufBrakeOff` actions hidden, and define DEL-ONE-SHOT-SYS to be the composition of TRAIN and DEL-ONE-SHOT-AND-BUF.

Correctness of DEL-ONE-SHOT In [26] we give a complete proof of correctness of the DEL-ONE-SHOT controller. The proof is based on a simulation mapping from this case to the unbuffered case of Section 3 — specifically, from DEL-ONE-SHOT-AND-BUF to ONE-SHOT. The safety and timeliness properties of the unbuffered case carry over to the buffered case via the simulation.

Since the safety and timeliness properties mention only variables in TRAIN, it may seem surprising that the simulation mapping excludes TRAIN. The simulation mapping implies that the external behavior of DEL-ONE-SHOT-AND-BUF is a subset of the external behavior of ONE-SHOT. Since this external behavior is all the input that TRAIN receives, TRAIN's behavior in the buffered case is a subset of its behavior in the unbuffered case. Therefore, the safety and timeliness properties, which involve only variables of TRAIN, carry over from the unbuffered case to the buffered case.

We present the simulation relation R from DEL-ONE-SHOT-AND-BUF to ONE-SHOT. The key insight is that since external behavior must be preserved, the timing of the external actions, `brakeOn` and `brakeOff`, must coincide in the two systems. Let s denote a state in the implementation (DEL-ONE-SHOT-AND-BUF), and u denote a state in the specification (ONE-SHOT); the states are related via R when the following two conditions hold:

1. $u.now = s.now$
2. By cases of $s.phase$:
   (a) `idle`, then $u.phase =$ `idle`
   (b) `braking`, by cases of $s.request$:
      i. on, then $u.phase =$ `idle`
      ii. none, then $u.phase =$ `braking` and $u.first(OFF) \leq s.first(OFF) + \delta^-$ and $u.last(OFF) \geq s.last(OFF) + \delta^+$
   (c) `done`, by cases of $s.request$:
      i. `off`, then $u.phase =$ `braking` and $u.first(OFF) \leq s.first(BUFF)$ and $u.last(OFF) \geq s.last(BUFF)$
      ii. none, then $u.phase =$ `done`

Roughly speaking, the simulation maps the phases of the implementation DEL-ONE-SHOT-AND-BUF to the phases of ONE-SHOT. The `idle` phase of DEL-ONE-SHOT-AND-BUF, plus the portion of the `braking` phase in which the on request is in the buffer, together map to the `idle` phase of ONE-SHOT. The rest of the `braking` phase of DEL-ONE-SHOT-AND-BUF, after the `brakeOn` action, plus the portion of the `done` phase in which the `off` request is in the buffer, together map to the `braking` phase of ONE-SHOT. The rest of the `done` phase of DEL-ONE-SHOT-AND-BUF, after the `brakeOff` action, maps to the `done` phase of ONE-SHOT. Note that a key part of the simulation mapping is a set of inequalities involving the deadlines in the two automata.

Lemma 4.1 *Relation R is a simulation from* DEL-ONE-SHOT-AND-BUF *to* ONE-SHOT.

Proof: We show the three conditions in the definition of a simulation. As for the proofs of invariants, this breaks down into separate cases for discrete steps and trajectories. (Note that this proof depends on the stronger parameter constraints of this section.) ■

Theorem 4.2 DEL-ONE-SHOT *is a correct buffered-brake-controller.*

Proof: By Lemma 4.1 and Theorem 2.1, DEL-ONE-SHOT-AND-BUF $\leq$ ONE-SHOT. By Lemma 3.3 and Theorem 3.6 DEL-ONE-SHOT-AND-BUF is a correct brake-controller. This implies that DEL-ONE-SHOT is a correct buffered-brake-controller. (Again, this proof depends on the stronger parameter constraints.) ■

This example demonstrates the use of simulation mappings to prove implementation relationships, including implementation relationships involving timing properties. Again, discrete and continuous reasoning are combined.

5 Case 3: Feedback and No Delay

In this section we (briefly) describe a more complex model of the deceleration problem in which the train provides the controller with sensor feedback at regular intervals, allowing the controller to adjust its proposed acceleration. Figure 2 illustrates the components and their communication.

Our new version of the train automaton, SENSOR-TRAIN, reports its acceleration, velocity and position in a `status` message every δ_s time units. (In order to express this in terms of an HIOA, we add a $last(STATUS)$ deadline component and manage it appropriately.) SENSOR-TRAIN has an `accel`(a) input action for each real number a, which causes the actual acceleration to be set to anything in the interval $[a - \ddot{c}_{\text{err}}, a]$. The proposed acceleration a is remembered in a variable acc.

We model a controller ZIG-ZAG that performs an `accel` output immediately after receiving each `status` input. It initially requests an acceleration a such that, if SENSOR-TRAIN followed a exactly, it would reach velocity exactly $\dot{c}_{\text{maxf}}$ when $x = c_f$. Since the train might actually decelerate faster than a, ZIG-ZAG might observe at any sample point that the train is going slower than expected. In this case, ZIG-ZAG does not change a until the velocity actually becomes $\leq \dot{c}_{\text{maxf}}$. Thereafter, at each sample point, ZIG-ZAG requests an acceleration that aims to reach $\dot{c}_{\text{maxf}}$ at exactly the following sample point.

We prove the same two properties for this case as we did for the no-feedback case, but for tighter bounds on the final velocity. The argument again uses invariants. For example, part of our argument involves showing that in all reachable states, $\dot{x} \geq \dot{c}_{\text{minf}}$. Now to prove this by induction, we need auxiliary statements about what is true between sample points, for example:

Lemma 5.1 *In all reachable states between sample points,* $\dot{x} + (acc - \ddot{c}_{\text{err}})(last(STATUS) - now) \geq \dot{c}_{\text{minf}}$.

That is, if the current velocity is modified by allowing the minimum acceleration consistent with the current acc, until the next sample point, then the result will still be $\geq \dot{c}_{\text{minf}}$. Note the use of the $last(STATUS)$ deadline to express the time until the next sample point. This lemma is proved by induction.

This example illustrates how our methods can be used to handle more complicated examples, including periodic sampling and control. It shows how to reason about periodic sampling using intermediate invariants involving the $last(STATUS)$ deadline: The controller issues control requests to the system at sample times, but can "lose control" of the system's behavior between sample points; the invariants are used to bound how badly the system's performance can degrade between sample points.

6 Case 4: Delay and Feedback

This case is more complicated in its details, but does not require any new ideas not present in the previous three cases. We omit the details here.

7 Conclusion

We have demonstrated how hybrid I/O automata and their associated proof techniques can be applied to a non-trivial hybrid system case study. These techniques include HIOA composition, invariants, and simulations, combined with the usual techniques of continuous analysis. The case study proves safety and timeliness properties for four deceleration controllers under different communication models.

We model all system components, both continuous and discrete, and the interactions among them. Deadline variables are used to express timing restrictions. Correctness conditions are formulated in terms of the real-world components of the systems.

The correctness proofs are based predominantly on invariant assertions, including assertions involving real-valued quantities representing real-world behavior, and assertions involving deadline variables representing timing restrictions. The systems are described at different levels of abstraction, with simulation mappings used to connect the levels. Deadline variables are used to reason about periodic sampling. The proofs combine discrete and continuous reasoning within a single framework. Uncertainty is handled carefully throughout. The proofs cover all cases, not just the apparent worst cases. The proofs are clear and scale well from the simplest case to the feedback with delay case.

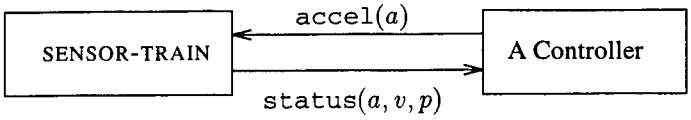


Figure 2. Overview of Feedback Deceleration Model

Our work does not supplant the usual methods of continuous mathematics, but rather incorporates them. We do not provide any new methods for deriving controllers, but rather a framework for understanding their requirements and for verifying that proposed controllers work correctly.

It remains to apply these techniques to additional case studies in automated transportation, especially those with complex discrete activity. We are currently modelling multi-vehicle maneuvers arising in the California PATH project [6, 5, 13]. We are also extending the preliminary treatment of safety systems in [27] to handle additional safety checks. The related discipline of air traffic control should also provide some interesting case studies.

It also remains to integrate into our framework the techniques of relevant disciplines such as control theory. For example, it would be useful to have a catalog of results from control theory that are especially useful for reasoning about transportation systems using HIOAs. Another direction is to develop computer tools to support the representation, specification and verification of such systems using HIOAs. All of this work should lead toward a long-term goal of developing industrial strength formal tools to help in the design of real transportation systems.

References

[1] R. Alur, C. Courcoubetis, T. Henzinger, P. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[2] R. Alur and D. Dill. Automata for modelling real-time systems. In *Proc. 17th ICALP Lecture Notes in Computer Science 443*, pages 322–335. Springer-Verlag, 1990.

[3] J. Frankel, L. Alvarez, R. Horowitz, and P. Li. Robust platoon maneuvers for AVHS. Manuscript, Berkeley, November 10, 1994.

[4] R. Gawlick, R. Segala, J. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. Technical Report MIT/LCS/TR-587, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 02139, December 1993. Condensed version in Serge Abiteloul and Eli Shamir, editors, *Proceedings of the 21st International Colloquim, ICALP94*, volume 820 of *Lecture Notes in Computer Science*, pages 166-177, Jerusalem, Israel, July 1994. Springer-Verlag.

[5] D. Godbole and J. Lygeros. Longitudinal control of the lead car of a platoon. California PATH Technical Memorandum 93-7, Institute of Transportation Studies, University of California, November 1993.

[6] D. N. Godbole, J. Lygeros, and S. Sastry. Hierarchical hybrid control: A case study. Preliminary report for the California PATH program, Institute of Transportation Studies, University of California, August 1994.

[7] C. Heitmeyer and N. Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. In *Proceedings of the Real-Time Systems Symposium*, pages 120–131, San Juan, Puerto Rico, December 1994. IEEE. Full version in Technical Memo MIT/LCS/TM-511, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, November 1994.

[8] C. Heitmeyer and N. Lynch. Formal verification of real-time systems using timed automata. In C. Heitmeyer and D. Mandrioli, editors, *Formal Methods for Real-Time Computing*, Trends in Software, chapter 4, pages 83–106. John Wiley & Sons Ltd, April 1996.

[9] T. Henzinger, Z. Manna, and A. Pnueli. Timed transition systems. In J. W. de Bakker, C. Huizing, and G. Rozenberg, editors, *Proceedings of REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Comupter Science*, pages 226–251. Springer-Verlag, June 1991.

[10] L. Lamport. The temporal logic of actions. Technical Report 79, Digital Systems Research Center, December 25 1991.

[11] G. Leeb and N. Lynch. Proving safety properties of the steam boiler controller: Formal methods for industrial applications, a case study, 1996. To appear in *Lecture Notes in Computer Science*, Springer-Verlag series.

[12] V. Luchangco. Using simulation techniques to prove timing properties. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, June 1995.

[13] J. Lygeros and D. N. Godbole. An interface between continuous and discrete-event controllers for vehicle automation. California PATH Research Report UCB-ITS-PRR-94-12, Institute of Transportation Studies, University of California, April 1994.

[14] N. Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control* (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of *Lecture Notes in Computer Science*, pages 449–463. Springer-Verlag, 1996.

[15] N. Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. In *Proceedings of the Third AMAST Workshop on Real-Time Systems*, pages 1–22, Salt Lake City, Utah, March 1996.

[16] N. Lynch, R. Segala, F. Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control* (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of *Lecture Notes in Computer Science*, pages 496–510. Springer-Verlag, 1996.

[17] N. Lynch and F. Vaandrager. Forward and backward simulations – Part II: Timing-based systems. *Information and Computation*. To appear. Available now as Technical Memo MIT/LCS/TM-487.c, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, April 1995.

[18] N. Lynch and F. Vaandrager. Forward and backward simulations — Part I: Untimed systems. *Information and Computation*, 121(2):214–233, September 1995.

[19] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J. de Bakker, C. Huizing, W. de Roever, and G. Rozenberg, editors, *REX Workshop on Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 447–484, Mook, The Netherlands, June 1991. Springer-Verlag.

[20] K. Marzullo, F. B. Schneider, and N. Budhiraja. Derivation of sequential real-time, process control programs. In A. M. van Tilborg and G. M. Koob, editors, *Foundations of Real-Time Computing*, pages 39–54. Kluwer Academic Publishers, 1991.

[21] M. Merritt, F. Modugno, and M. Tuttle. Time constrained automata. In J. C. M. Baeten and J. F. Goote, editors, *CONCUR'91: 2nd International Conference on Concurrency Theory*, volume 527 of *Lecture Notes in Comupter Science*, pages 408–423, Amsterdam, The Netherlands, Aug. 1991. Springer-Verlag.

[22] S. Nadjm-Tehrani. Modelling and formal analysis of an aircraft landing gear system. In *Second European Workshop on Real-Time and Hybrid Systems*, pages 239–246, Grenoble, France, May 1995.

[23] J. Søgaard-Andersen. *Correctness of Protocols in Distributed Systems*. PhD thesis, Technical University of Denmark, Lyngby, Denmark, December 1993. ID-TR: 1993-131.

[24] F. Vaandrager and N. Lynch. Action transducers and timed automata. In W. R. Cleaveland, editor, *CONCUR '92: 3rd International Conference on Concurrency Theory*, volume 630 of *Lecture Notes in Computer Science*, pages 436–455, Stony Brook, NY, USA, August 1992. Springer Verlag.

[25] J. Vitt and J. Hooman. Specification and verification of a real-time steam boiler system. In *Second European Workshop on Real-Time and Hybrid Systems*, pages 205–208, Grenoble, France, May 1995.

[26] H. Weinberg. Correctness of vehicle control systems: A case study. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, February 1996. Also, MIT/LCS/TR-685 and URL http://theory.lcs.mit.edu/tds/HBW-thesis.html.

[27] H. B. Weinberg, N. Lynch, and N. Delisle. Verification of automated vehicle protection systems. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control* (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of *Lecture Notes in Computer Science*, pages 101–113. Springer-Verlag, 1996.

HART'97

Safety Verification for Automated Platoon Maneuvers: A Case Study

Ekaterina Dolginova and Nancy Lynch

MIT Laboratory for Computer Science
Cambridge, MA 02139, USA
{katya, lynch}@theory.lcs.mit.edu

Abstract. A system consisting of two platoons of vehicles on a single track, plus controllers that operate the vehicles, plus communication channels, is modeled formally, using the hybrid input/output automaton model of Lynch, Segala, Vaandrager and Weinberg [7]. A key safety requirement of such a system is formulated, namely, that the two platoons never collide at a relative velocity greater than a given bound v_{allow}. Conditions on the controller of the second platoon are given, designed to ensure the safety requirement regardless of the behavior of the first platoon. The fact that these conditions suffice to ensure safety is proved. It is also proved that these conditions are "optimal", in that any controller that does not satisfy them can cause the safety requirement to be violated. The model includes handling of communication delays and uncertainty. The proofs use composition, invariants, levels of abstraction, together with methods of mathematical analysis.
This case study is derived from the California PATH intelligent highway project, in particular, from the treatment of the platoon join maneuver in [3].

1 Introduction

Increasing highway congestion has spurred recent interest in the design of intelligent highway systems, in which cars operate under partial or total computer control. An important new effort in this area is the California PATH project (see, for example, [9]), which has developed a design for automating the operation of cars in several lanes of selected California highways. In this design, cars become organized into *platoons* consisting of a leader car and several following cars; the followers do not operate independently, but follow the control instructions of the leader.

An important maneuver for the proposed PATH system is the *platoon join* maneuver, in which two or more adjacent platoons combine to form a single platoon. The design of such a maneuver is described and analyzed in [3]. This maneuver involves both discrete and continuous behavior: discrete behavior appears in the form of synchronization and agreement among the controllers about the join process, plus communication among the various system components,

whereas continuous behavior appears in the motion of the cars. The combination forms a hybrid system of considerable complexity.

A key issue for the platoon join maneuver is its safety, represented by the requirement that cars never collide at too great a relative speed. In [3], a proof of such a safety property is outlined, for the specific platoon join maneuver given in that paper. The key to the proof turns out to be that the given maneuver always ensures that either (a) the platoons are sufficiently far apart that the second platoon can slow down sufficiently before hitting the first platoon, or (b) the relative speeds of the two platoons are already close enough.

Although the outline [3] gives the key ideas, from our point of view, it is incomplete as a safety verification. It does not include a complete model of all system components – in particular, the discrete components are not modeled. It does not seem to cover all cases that could arise: for instance, only some types of communication delay are handled, and uncertainties in the values of some parameters are not considered. The analysis contains informal "jumps" in which certain types of behavior are claimed to be the "worst possible", and then only these cases are analyzed carefully; however, it is not made clear how one can be sure that the claimed worst cases are in fact the worst. Another problem is that the analysis is presented for just the single maneuver, and is intertwined with the proofs of other properties for that maneuver (successful join, optimality of join time). However, it seems that the analysis should be decomposable, for example, proving the safety requirement in a way that allows the proof to apply to other maneuvers besides just the platoon join.

In previous work [7], Lynch, Segala, Vaandrager and Weinberg have developed a formal model, the *hybrid input/output automaton model*, for hybrid systems, together with associated proof techniques. These techniques include methods based on automaton composition, on invariant assertions, on levels of abstraction, and on mathematical analysis for reasoning about continuous behavior. They have developed methods of incorporating standard methods of analysis into automaton-based proofs. So far, these methods have been used to model and verify a variety of simple real-time systems, including several very simple maneuvers arising in automated transportation systems ([11], [10], [6]).

In this case study, we apply the hybrid I/O automaton model and its associated proof methods to the task of describing and verifying safety for the PATH platoon join maneuver. This is a more complex example than those previously considered using hybrid I/O automata. We aim for an accurate, complete model of the system, plus proofs that cover all cases and accommodate all realistic variations, including delays and uncertainties. Our safety proofs should apply as generally as possible, for instance, to other maneuvers besides platoon join. Our model should also be usable for proving other properties, such as successful join and optimality. The system and its proofs should admit decomposition into separate parts, as far as possible, and should be easy to extend.

In the work we have completed so far, we have made certain simplifications. Namely, we consider the case of two platoons only (as in [3]), and we consider uncertainties in only some of the parameter values. Moreover, we pretend that

the controllers control the cars' acceleration rather than their jerk (derivative of the acceleration). We intend to remove these restrictions in later work, and are designing our models and proofs to make such extensions easy.

For this simplified setting, we have succeeded in modeling the complete system, which consists of two platoons of cars on a single track, plus controllers that operate the cars, plus communication channels. We have formulated the safety requirement, namely, that the two platoons never collide at a relative velocity greater than a given bound v_{allow}. We have given conditions on the controller of the second platoon, designed to ensure the safety requirement regardless of the behavior of the first platoon, and we have proved that these conditions suffice to ensure safety. Our proofs cover all cases, and are sufficiently general to apply to other maneuvers besides platoon join. The proofs use discrete systems techniques, such as composition, invariants, and levels of abstraction. Additionally, the methods of mathematical analysis developed for proving invariance of state-space sets in [2] are used for reasoning about the continuous parts of the system.

In addition to proving safety, we also give results showing that the given conditions on the controllers are "optimal", in the sense that any controller that does not satisfy them can cause the safety requirement to be violated. The optimality results are proved using the same techniques (in particular, invariants and composition) that are used for the safety proof. Again, the optimality results apply to other maneuvers besides platoon join.

An alternative approach to proving safety for the platoon join maneuver, based on game theory, is presented in [5], [4]. There has been a large amount of prior work on modelling and verification of hybrid systems, as represented, for example, in the six previous workshops on hybrid systems. Nearly all of this work differs from ours in using either control theory methods, or else algorithmic techniques (e.g., decision procedures based on finite-state analysis). Other formal models for hybrid systems appear in [8], [1]; these differ from ours primarily in placing less emphasis on issues of external behavior, composition and abstraction.

We consider the research contributions of this paper to be: (a) The model and proof of safety for the platoon join (and other maneuvers). (b) The optimality result and its proof. (c) A demonstration of the power of hybrid I/O automata and its associated proof methods for reasoning about interesting hybrid systems. (d) A demonstration of the use of abstraction levels as a means of handling complexity.

2 HIOA Model

The Hybrid I/O Automata model presented in [7] is capable of describing both continuous and discrete behavior. The model allows communication among components using both shared variables and shared actions. Several HIOA techniques make them particularly useful in modeling and reasoning about hybrid systems. These include composition, which allows to form complex automata from simple

building blocks; implementation relations, which make it easy to use levels of abstraction when modeling complex systems; invariant assertions, which describe the non changing properties of the system.

A *state* of a HIOA is defined to be a valuation of a set of variables. A *trajectory* w is a function that maps a left-closed interval I of the reals, with left endpoint equal to 0, to states; a trajectory represents the continuous evolution of the state over an interval of time. An HIOA A consists of:

- Three disjoint sets of *input, output and internal* variables. Input and output variables together are called *external* variables.
- Three disjoint sets of *input, output and internal* actions.
- A nonempty set of *start states.*
- A set of *discrete transition*, i.e. (state, action, state) triples.
- A set of trajectories over the variables of A.

We now define executions of HIOAs. A *hybrid execution fragment* of A is a finite or infinite alternating sequence of trajectories and actions, ending with a trajectory if it is finite. An execution fragment records all the discrete changes that occur in an evolution of a system, plus the continuous state changes that occur in between. Hybrid execution fragments are called *admissible* if they are infinite. A *hybrid execution* is an execution fragment in which the first state is a start state. A state of A is defined to be *reachable* if it is the last state of some finite hybrid execution of A. A *hybrid trace* of a hybrid execution records only the changes to the external variables. Hybrid traces of an HIOA A (hybrid trace that arise from all the finite and admissible hybrid executions of A) describe its visible behavior.

HIOA A *implements* HIOA B if every behavior of A is allowed by B. A is typically more deterministic than B in both the discrete and the continuous level. Formally, if A implements B, then 1) A and B are *comparable* HIOA, meaning that they have the same external actions and external variables; 2) all the hybrid traces of A are included in those of B. To prove the second part, we need to show that there exists a *simulation relation* from A to B. A simulation relation from A to B is a relation R from states of A to states of B satisfiying:

- If s_A is a start state of A, then there exists s_B, $s_A R s_B$, such that s_B is a start state of B.
- If a is an action of A, (s_A, a, s'_A) is a discrete transition of A, $s_A R s_B$, and both s_A and s_B are reachable, then B has a finite execution fragment starting with s_B, having the same trace as the given step, and ending with a state s'_B with $s'_A R s'_B$.
- If w_A is a trajectory of A from s_A to s'_A, $s_A R s_B$, and both s_A and s_B are reachable, then B has a finite execution fragment starting with s_B, having the same trace w, and ending with a state s'_B with $s'_A R s'_B$.

Another technique for reducing complexity is HIOA *composition.* HIOAs A and B can be composed if they have no output actions or output variables in common, and if no internal variable of either is a variable of the other. The

composed HIOA C's input variables/actions are the union of A and B's input variables/actions minus the union of A and B's output variables/actions; all the other components (output and internal variables/actions, start states, discrete actions, trajectories) are the unions of the corresponding components of A and B. The crucial result is that the composition operator respects the implementation relation: if A_1 implements A_2 then A_1 composed with B implements A_2 composed with B. Finally, *invariant assertions* state system properties that are true in any reachable state of the system.

3 System Model

We consider two platoons of vehicles, moving along a single track. While the behavior of the leading platoon is arbitrary, the second platoon's controller must make sure that no "bad" collisions occur. "Bad" collisions are collision at a high relative speed. This is called the *Safety* requirement for the second controller. This *Safety* requirement is general for all platoon maneuvers, and is independent of the particular algorithm used. We devise the most nondeterministic safe controller, so that later we can use this controller as a correctness check: a controller implementing any platoon maneuver must implement our safe controller in order to be correct. This should be very useful in formally proving correctness of complicated algorithms.

3.1 *Controlled-Platoons*

We compose our system of a piece modeling the real world (the physical platoons) and two pieces modeling the controllers of each platoon (which are described in the next subsection). Each piece is modeled by a hybrid automaton. The real world piece is called *Controlled-Platoons*, shown in Figure 1. It consists of two platoons, named 1 and 2, where platoon 1 precedes platoon 2 on a single track. Positions on the track are labeled with nonnegative reals, starting with 0 as a designated beginning point. We pretend for simplicity here that the platoons have size 0. In the full version of the paper this restriction is relaxed. Note that the velocities of the platoons are always nonnegative – the vehicles will never go backwards, and the platoons are not allowed to bypass each other.

Only single collisions are modeled here. A special *collided* variable keeps track of the first occurrence of a collision. Before a collision, the platoons obey their respective controllers by setting the given acceleration. After a collision occurs, the platoons are uncoupled from the controllers and their velocities are set arbitrarily.

We use the constants $v_{allow} \in \mathsf{R}^{\geq 0}$ to represent the largest allowable velocity when a collision occurs, and $a_{min} \in \mathsf{R}^{\geq 0}$ to represent the absolute value of the maximum emergency deceleration. The platoons' position, velocity, and acceleration data is modeled by x_i, $\dot{x}_i$, and $\ddot{x}_i$, respectively. The dots are used as a syntactic device only. The differential relationships between these variables is a consequence of the trajectory definitions; however, this differential relationship

Actions:
Internal: *collide*
Variables:
Input: $\ddot{x}_i \in \mathsf{R}$, $i \in \{1,2\}$, initially arbitrary
Output: $\dot{x}_i \in \mathsf{R}^{\geq 0}$, $i \in \{1,2\}$, initially arbitrary
$x_i \in \mathsf{R}^{\geq 0}$, $i \in \{1,2\}$; initially $x_2 = 0$ and x_1 is arbitrary
collided, Boolean, initially *false*
now, initially 0
Discrete Transitions:
collide
Pre: $x_1 = x_2$
collided = *false*
Effect: $\dot{x}_i :=$ arbitrary value, $i \in \{1,2\}$
collided = *true*
Trajectories:
an I-trajectory w is included among the set of nontrivial trajectories exactly if
collided is unchanged in w
for all $t \in I$ the following hold:
if *collided* = *false* in w then
$w(t).\dot{x}_i = w(0).\dot{x}_i + \int_0^t w(u).\ddot{x}_i du$, $i \in \{1,2\}$.
$w(t).now = w(0).now + t$
$w(t).x_2 \leq w(t).x_1$
$w(t).x_i = w(0).x_i + \int_0^t w(u).\dot{x}_i du$
if $w(t).x_1 = w(t).x_2$ and t is not the right endpoint of I then
collided = *true*.

Fig. 1. The *Controlled-Platoons* Hybrid I/O Automaton

is partly lost after a collision occurs. The acceleration data is received from the controllers which are defined below. This will be used in our statement of the correctness property, below — we only want to assert what happens the first time a collision occurs. The second conditions on the trajectories of *Controlled-Platoons* guarantees that the platoon only executes the controller's decisions until the first collision occurs.

3.2 Controllers

Controller$_1$ is described in Figure 2. It is an arbitrary hybrid automaton with the given interface, restricted only by physical limitations. Note that the controller does not have any actions. The last restriction on continuous trajectories, for example, guarantees that the controller does not make the platoon to have negative velocity. The internal velocity and position variables ($\dot{x}_{int1}$ and x_{int1}) are used to keep track of the platoon's own data. This data is obtained by integrating their acceleration settings. Since there are no delays or uncertainties, these variables should correspond exactly to the actual position and velocity of the platoon.

The *Controller*$_2$ hybrid automaton is the same as *Controller*$_1$, except that

Variables:

Input: $\dot{x}_2 \in \mathsf{R}^{\geq 0}$

$x_2 \in \mathsf{R}^{\geq 0}$

Output: $\ddot{x}_1$

Internal: $\dot{x}_{int1} \in \mathsf{R}^{\geq 0}$, initially $\dot{x}_{int1} = \dot{x}_1$

$x_{int1} \in \mathsf{R}^{\geq 0}$, initially $x_{int1} = x_1$

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if

$\ddot{x}_1$ is an integrable function

for all $t \in I$, at $w(t)$

$\dot{x}_{int1} = w(0).\dot{x}_{int1} + \int_0^t w(u).\ddot{x}_1 du$

$x_{int1} = w(0).x_{int1} + \int_0^t w(u).\dot{x}_{int1} du$

$\ddot{x}_1 \geq -a_{min}$

Fig. 2. *Controller*$_1$ Hybrid I/O Automaton

it inputs x_1 and $\dot{x}_1$, and outputs $\ddot{x}_2$.

Compose *Controlled-Platoons*, *Controller*$_1$ and *Controller*$_2$ using hybrid I/O automata composition rules to obtain an automaton that models our platoon system with each platoon having its own controller.

3.3 Safety Condition

We place a safety condition on states of *Controlled-Platoons*. The safety condition guarantees that if the platoons ever collide, then the first time they do so, their relative velocity is no more than v_{allow}. We formulate this condition formally as an invariant assertion:

Safety : If $x_1 = x_2$ and *collided* = *false*, then $\dot{x}_2 \leq \dot{x}_1 + v_{allow}$.

We define a new automaton, *Safe-Platoons*, to serve as a correctness specification. *Safe-Platoons* is exactly the same as *Controlled-Platoons* except that all the states are restricted to satisfy the safety condition.

We are supposed to design *Controller*$_2$ so that when it is composed in this way with arbitrary *Controller*$_1$, the resulting system satisfies the safety condition. Then we can say that it implements the *Safe-Platoons* automaton, using a notion of implements based on preserving hybrid traces. Here, the hybrid trace includes the output variables, which are the positions, velocities and accelerations of both platoons plus the *collided* flag. That is enough to ensure that the *Safety* condition of the spec carries over to the implementation.

4 The Ideal Case

4.1 The Model

We start with a treatment of the safety property in the ideal setting. This allows us to prove some important properties of the simpler model first, and then extend

them to the more complicated models via simulation mappings. By ideal setting we mean that there are no delays and/or uncertainties in either the sensor's data or the controller's directives. In the next few sections we will make the model more realistic by relaxing these restrictions. Also, in this abstract, we make the simplifying assumption that the platoons have size 0. In the full paper we show how to relax this restriction easily.

We define and prove correctness of a specific $Controller_2$, called C_2, which implements our safety condition, in Figure 3. This controller is very nondeterministic.

Definition:

$safe\text{-}measure = \max(x_1 - x_{int2} - \frac{(\dot{x}_{int2})^2 - (\dot{x}_1)^2 - (v_{allow})^2}{2a_{min}}, \dot{x}_1 + v_{allow} - \dot{x}_{int2})$

Variables:

Input: $\dot{x}_1 \in \mathsf{R}^{\geq 0}$

$x_1 \in \mathsf{R}^{\geq 0}$

Output: $\ddot{x}_2$, initially if $safe\text{-}measure \leq 0$, then $\ddot{x}_2 = 0$

Internal: $\dot{x}_{int2} \in \mathsf{R}^{\geq 0}$, initially $\dot{x}_{int2} = \dot{x}_2$

$x_{int2} \in \mathsf{R}^{\geq 0}$, initially $x_{int2} = x_2$

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if

w is a trajectory of $Controller_2$

if $collided = false$ in $w(0)$ then $\forall t \in I$

if $safe\text{-}measure \leq 0$ then $\ddot{x}_2 = -a_{min}$

Fig. 3. C_2 Hybrid I/O Automaton

C_2 ensures that if the position and velocity parameters are on the boundary defined by *safe-measure*, then platoon 2 is guaranteed to be decelerating as fast as possible. This is guaranteed by the second condition on the trajectories of C_2.

4.2 Correctness of C_2

We will now prove correctness of our controller. This means that any controller that implements C_2, will be correct (safe).

We define a predicate S on states of *Platoons*, as follows:

Predicate S: If $collided = false$ then $safe\text{-}measure \geq 0$, where

$$safe\text{-}measure = \max(x_1 - x_{int2} - \frac{(\dot{x}_{int2})^2 - (\dot{x}_1)^2 - (v_{allow})^2}{2a_{min}}, \dot{x}_1 + v_{allow} - \dot{x}_{int2})$$

This says (from the definition of *safe-measure*, see Figure 3) that if the platoons have not collided yet, then either (a) the distance between the two platoons is great enough to allow platoon 2 to slow down sufficiently before hitting platoon 1, even if platoon 1 decelerates at its fastest possible rate, or (b) the relative velocities of the two platoons are already close enough.

We define a new automaton *C-Platoons*, which is exactly like *Controlled-Platoons*, with the additional restriction that in all the initial states *safe-measure* ≥ 0 (thus all the initially states satisfy Predicate S, since initially *collided* = *false*). The system *Implemented-Platoons* that we are considering is the composition of *C-Platoons*, an arbitrary $Controller_1$, and C_2. C_2 is designed to guarantee explicitly that if S is ever violated, or even if it in danger of being violated (because equality holds), platoon 2 is decelerating as fast as possible. We claim that this strategy is sufficient to guarantee that S is always true:

Lemma 1. *S is true in every reachable state of the* Implemented-Platoons.

As a simple consequence of Lemma 1, we obtain the safety condition:

Lemma 2. *In any reachable state of Implemented-Platoons, if $x_1 = x_2$ and collided = false, then $\dot{x}_2 \leq \dot{x}_1 + v_{allow}$.*

Now we use Lemma 2 to prove that the system is in fact safe, i.e., that it implements *Safe-Platoons*. We prove this using a simulation relation f. This simulation is trivial – the identity on all state components of *Safe-Platoons* (velocities, positions, and the *collided* flag).

Lemma 3. *f is a forward simulation from the composed system* Implemented-Platoons *to Safe-Platoons.*

Proof: By induction on the number of steps in the hybrid execution. Lemma 2 deals with trajectories; the proofs for the start states and discrete steps are relatively simple.

Theorem 4. *The Implemented-Platoons system implements Safe-Platoons, in the sense that for every hybrid execution α of Implemented-Platoons, there is a hybrid execution α' of Safe-Platoons that has the same hybrid trace – here, means same positions, velocity and collided flag values.*

Proof. Implemented-Platoons and *Safe-Platoons* are comparable and by Lemma 3, there is a simulation relation f from *Implemented-Platoons* to *Safe-Platoons*. Therefore, this composed system implements *Safe-Platoons*.

4.3 Optimality

We will now prove optimality of *safe-measure* using the analysis theorem about non-increasing functions. Informally, we want to prove that any $Controller_2$ that does not implements C_2 is unsafe. The formal definition of this optimality property appears in Theorem 7. Combined with the correctness result of the previous subsection, this will allows to decide whether any given controller is safe, since it is safe *if and only if* it implements C_2.

Define *Controller*$_2$ to be bad (and call it *Bad-Controller*$_2$), if there exists some *Controller*$_1$, such that in any admissible hybrid execution α of an automaton composed of *Controlled-Platoons*, *Controller*$_2$ and *Controller*$_1$, $\exists s \in \alpha$, which does not satisfy Predicate S.

Define *Bad-Controller*$_1$, given *Bad-Controller*$_2$, so that in the system composed of *Bad-Controller*$_1$, *Bad-Controller*$_2$ and *Controlled-Platoons* (the system called *Bad-Platoons*), for any admissible hybrid execution β, the following hold:

- $\exists s \in \beta$, s does not satisfy Predicate S;
- strictly after the occurrence of s, $\ddot{x}_1 = -a_{min}$.

The first lemma shows that once Predicate S is violated, it will remain violated, given some "bad" *Controller*$_1$. Formally,

Lemma 5. *If a given Controller*$_2$ *is bad, then in any Bad-Platoons system with this Controller*$_2$*, Predicate S is violated in all the states* $\in \beta$ *that occur strictly after s, in which collided = false. (Bad-Platoons,* β*, s are as defined above.)*

The next lemma shows that if Predicate S is violated in some state, then *safety* will also be violated eventually. Formally,

Lemma 6. *If a given Controller*$_2$ *is bad, then in any Bad-Platoons system with this Controller*$_2$*, in any admissible execution* γ*,* $\exists s' \in \gamma$*, which does not satisfy* safety*).*

Theorem 7. *For any* Bad-Controller$_2$*, there always exists such* Controller$_1$ *(*C_1'*, which is not necessarily the same as* C_1*), that a Bad-Platoons system composed with these controllers has in its hybrid trace a state* s'*, in which safety is violated.*

The last theorem shows that our controller is optimal, i.e., any *Controller*$_2$ that does not implement it, might lead to an unsafe state, given some "bad" *Controller*$_1$.

5 Delayed Response

Now we consider the case where there is a delay between the receipt of information by the controller for platoon 2 and its resulting action. There appear to be two distinct types of delay to consider — the inbound and the outbound delay; we model them separately. The inbound delay is due to delays in communicating sensor information to the controllers. The outbound delay comes from the fact the controller's decision are implemented by the platoons after some delay.

We use levels of abstractions to deal with the complexity of the delayed case. The use of simulation relations enables us to build correctness and optimality proofs based on the previous ideal case results. This makes all the proofs significantly easier.

5.1 The System with Inbound and Outbound Delays

We model both the inbound and the outbound delays by special delay buffers. To obtain the delayed system, we then compose our new controller with the delay buffers. First, we introduce the inbound delay buffer B_i (lag time in communicating sensor information) in Figure 4.

Variables:

Input: $\dot{x}_1 \in \mathsf{R}^{\geq 0}$, $x_1 \in \mathsf{R}^{\geq 0}$

Output: $\dot{x}_{i1} \in \mathsf{R}^{\geq 0}$, $x_{i1} \in \mathsf{R}^{\geq 0}$

Internal: *saved* - maps from an interval $(0, d_i)$ to $(\dot{x}_1, x_1)$

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if for all $t \in I$, $t > 0$ the following hold:

$$w(t).(\dot{x}_{i1}, x_{i1}) = \begin{cases} w(0).saved(t) & \text{if } t < d_i \\ (w(t-d_i).\dot{x}_1, w(t-d_i).x_1) & \text{otherwise} \end{cases}$$

$\forall t' \in (0, d_i)$,

$$w(t).saved(t') = \begin{cases} w(0).saved(t'-t) & \text{if } t' > t \\ (w(t-t').\dot{x}_1, w(t-t').x_1) & \text{otherwise} \end{cases}$$

Fig. 4. B_i Hybrid I/O Automaton

B_i acts in such a way that the output variables have exactly the values of the input variables, exactly time d_i earlier, where d_i is the maximum "information delay" – the longest time that it can take for a controller to receive the velocity and position sensor data. This delay buffer actually implements the more realistic version, in which the length of the delay varies nondeterministically within known bounds. Initially, the buffer (*saved*) is "prefed" with information that could have happened in that initial time period (so that the last position and velocity values in the buffer match up the initial position and velocity values of the platoons). Setting the maximum deceleration for that "imaginary" time period lets the controller be the most flexible (and thus optimal, as will be proven later), in the initial d_i time period.

Formally, the initial value of the *saved* variable is determined as follows. For any start state s of the system, construct a trajectory w of length d_i of *Controlled-Platoons* so that $w(0).\dot{x}_1 = s.\dot{x}_1 + d_i a_{min}$, $w(0).x_1 = s.x_1 + s.\dot{x}_1 d_i + \frac{a_{min} {d_i}^2}{2}$, and $\forall t \in (0, d_i)$, $w(t).\ddot{x}_1 = -a_{min}$. The second platoon's state components can be arbitrary, as long as no collisions occur. Now, $\forall t \in (0, d_i)$, let $saved(t) = w(t).(\dot{x}_1, x_1)$.

Next, we add an outbound delay buffer B_o (lag time in communicating control information). An outbound delay buffer B_o, is almost like the inbound buffer B_i, but with input variable $\ddot{x}_{o2}$, and output variable $\ddot{x}_2$. The *saved* variable is the same as in B_i, except that it now "saves" $\ddot{x}_{o2}$ and the length of the interval is d_o, where $d_o \in \mathsf{R}^{\geq 0}$ is the maximum "action delay" - the longest time that it can take

for a platoon to react to the controllers directives. Again, the delay time-length is exact. Initially, $\forall t \in (0, d_o)$, $saved(t) = -a_{min}$. This makes the platoons safe in the initial d_o time interval even if the first platoon starts decelerating.

Definition:

$$safe\text{-}measure_d = \max(x_{i1} + \dot{x}_{i1}t' - \frac{a_{min}t'^2}{2} - x_{new2} - \frac{\dot{x}_{new2}^2 - (\dot{x}_{i1} - a_{min}t_1)^2 - (v_{allow})^2}{2a_{min}},$$
$$\dot{x}_{i1} - a_{min}t' - \dot{x}_{new2} + v_{allow}),$$
where $t' = \min(d_i + d_o, \frac{\dot{x}_{i1}}{a_{min}})$

Variables:

Input: $\dot{x}_{i1} \in \mathsf{R}^{\geq 0}$
$x_{i1} \in \mathsf{R}^{\geq 0}$

Output: $\ddot{x}_{o2}$, initially if $safe\text{-}measure \leq 0$, then $\ddot{x}_2 = 0$

Internal: internal variables of $Controller_2$ ($\dot{x}_{int2}$ and x_{int2})
a_2 - maps from an interval $(0, d_o)$ to $\ddot{x}_{o2}$,
initially, $\forall t \in (0, d_o), a_2(t) = -a_{min}$, otherwise - arbitrary
x_{new2}, $\dot{x}_{new2}$ - the position and velocity of the second platoon after time d_o passes, provided *collided* still equals *false*

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if
w is a trajectory of $Controller_2$
if $collided = false$ in $w(0)$ then for all $t \in I$, $t > 0$:
if $safe\text{-}measure_d \leq 0$ then
$\ddot{x}_2 = -a_{min}$
$\forall t' \in (0, d_o)$,

$$w(t).a_2(t') = \begin{cases} w(0).a_2(t' - t) & \text{if } t' > t \\ w(t - t').\ddot{x}_{o2} & \text{otherwise} \end{cases}$$

$w(t).\dot{x}_{new2} = w(t).\dot{x}_{int2} + \int_0^{d_o} w(t).a_2(u)du$
$w(t).x_{new2} = w(t).x_{int2} + \int_0^{d_o}(\int_0^u w(t).a_2(u')du' + w(t).\dot{x}_{int2})du$

Fig. 5. D_2 Hybrid I/O Automaton

Finally, we modify the controller so that it handles the delays correctly. The controller D_2 (see Figure 5) implements $Controller_2$. It is similar to C_2 in that it also tries to keep the second platoon within the bounds set by $safe\text{-}measure_d$, which is *safe-measure* redefined for the delayed case. The new controller gets its inputs from the inbound delay buffer B_i and its output variable goes into the outbound delay buffer B_o. Additional internal variables (x_{new2} and $\dot{x}_{new2}$) are added to store the "future" position and velocity data, as calculated from the acceleration settings. A buffer for storing acceleration settings that the controller has output, but that has not been executed yet (a_2) is used for this purpose. Also, $safe\text{-}measure_d$ is defined instead of *safe-measure*; the only changes from *safe-measure* are that the 1st platoon's parameters are exchanged by their "worst-case" values after $d_i + d_o$ time units pass; and the 2nd platoon's parameters are exchanged by their projected values after d_o time units pass.

5.2 Correctness of D_2

We will now compose B_i, D_2, B_o using the hybrid I/O automata composition rules to obtain the delayed controller, which we call *Buffered-Controller*. A straightforward simulation relation shows that this composed system implements C_2. This simulation relation f is the identity on all the external state components of C_2. The use of simulation relations will allows us to prove correctness of our more complicated delayed controller relatively easily, since we have already proven correctness in the simple (ideal) case.

First we prove that if the old *safe-measure* (the one used in the ideal case) is non-positive in some state of a trajectory of *Buffered-Controller*, then the new controller D_2 (the one that has both the inbound and the outbound delays), will also output maximum deceleration, just as the old (ideal) controller would. Formally,

Lemma 8. *If collided = false in $w(0)$ of a trajectory w of Buffered-Controller, then $\forall t \in I$, such that $w(t).safe\text{-}measure \leq 0$, $\ddot{x}_2 = -a_{min}$.*

Lemma 9. *f (an identity relation on all the external components of C_2) is a forward simulation from the composed system Buffered-Controller to C_2.*

Proof. By induction on the number of steps in the hybrid execution. Start states and discrete steps are proven trivially; Lemma 8 is used to prove the simulation relation on continuous trajectories.

This lemma proves that *Buffered-Controller* implements C_2, since the two automata are comparable and there is a simulation relation from the first one to the second one. Therefore, we are now able to prove correctness of our delayed controller:

Theorem 10. *The doubly-delayed hybrid automaton composed of C-Platoons, Buffered-Controller implements Safe-Platoons.*

Proof. We have proved that the system *Buffered-Controller* implements C_2 in Lemma 9. But by Theorem 4, C_2 composed with *C-Platoons* (the *Implemented-Platoons* system) implements *Safe-Platoons*. Thus, the doubly-delayed hybrid automaton composed of *C-Platoons*, and *Buffered-Controller* also implements *Safe-Platoons* by the hybrid I/O automata composition rules!

5.3 Optimality

We will now prove optimality of D_2. Again, we will be basing our proofs on the optimality property of the controller C_2, which was proven in section 4.3. We want to prove that any controller with both inbound and outbound delays that does not implement controller D_2, is unsafe given some "bad" controller C_1. However, knowing that C_2 is optimal makes the proof much easier: we only need to show that a controller that would let $safe\text{-}measure_d$ get negative, will

eventually lead to a state in which *safe-measure* itself is negative. Then we can use optimality of C_2 to show that any such controller would not be correct.

Define *Buffered-Controller*$_2$ to be bad (*Bad-Buffered-Controller*$_2$), if there exists some *Controller*$_1$ (C_{d1}), such that in any admissible hybrid execution α of an automaton composed of *Controlled-Platoons*, C_{d1} and *Bad-Buffered-Controller*$_2$, $\exists s_d \in \alpha$, which does not satisfy Predicate S_d.

Lemma 11. *Any Bad-Buffered-Controller*$_2$ *implements Bad-Controller*$_2$.

Since we have just shown that the delayed automaton implements the non-delayed one, we can use the optimality property of the ideal case controller, to prove the optimality of the delayed controller easily:

Theorem 12. *Given any* Bad-Buffered-Controller$_2$, *we can always construct* Controller$_1$ *(*C_{d1}*) such that a system composed of* Controlled-Platoons *and these controllers has in any admissible hybrid execution a state* s'*, in which* safety *is violated.*

Proof. Take any *Bad-Buffered-Controller*$_2$. By Lemma 11, it implements *Bad-Controller*$_2$. Then by Theorem 7, there exists such *Controller*$_1$ (C_1'), that a system composed of *Controlled-Platoons*, C_1' and this *Bad-Buffered-Controller*$_2$ has in its hybrid trace a state s' that violates *safety.*

Therefore, the delayed *Controller*$_2$ is also optimal.

6 Uncertainty

Our model already includes both the inbound and the outbound delays in sending and receiving information between the controller and *Controlled-Platoons*. Now we will introduce an extra complexity which will make the model even more realistic: the uncertainty in information that the controller receives. This inbound uncertainty arises from inexact sensors that communicate the position and velocity data to the controllers. We will use similar methods to the ones used in the delay case. A special "uncertainty buffer" automaton will be defined, similar to the previous delay buffers. Then, the uncertainty will be implemented by adding this new automaton to the model and modifying the controller slightly. We will then prove correctness using the simulation relation to the delayed case which we have already worked out. This use of levels of abstraction makes the proofs for the complicated case, which involves both the delays and the uncertainties, relatively easy to both write and understand.

6.1 The System

We implement the delayed controller D_2 with a composition of two hybrid automata: another controller U_2, and an inbound uncertainty buffer U_i. We call

Variables:

Input: $\dot{x}_{i1}$, x_{i1}

Output: $\dot{x}_{iu1}$, x_{iu1}

Trajectories:

an I-trajectory w is included among the set of nontrivial trajectories exactly if for all $t \in I$, $t > 0$ the following hold:

$\dot{x}_{iu1} \in [\dot{x}_{i1} - \dot{\delta}, \dot{x}_{i1} + \dot{\delta}]$

$x_{iu1} \in [x_{i1} - \delta, x_{i1} + \delta]$

Fig. 6. U_i Hybrid I/O Automaton

this composed system *Uncertain-Controller*. The uncertainty buffer U_i nondeterministically garbles the position and velocity data within the given bounds (see Figure 6). The bounds are predefined constants $\delta \in \mathsf{R}^{\geq 0}$ — the maximum absolute value of uncertainty in position sensor data, and $\dot{\delta} \in \mathsf{R}^{\geq 0}$ — the maximum absolute value of uncertainty in velocity sensor data. The controller U_2 is the same as D_2 except that it now takes its inputs from the inbound uncertainty buffer U_i and that *safe-measure*$_u$ (see below) is defined to account for the uncertainties. Same as in the delay case, the only changes from *safe-measure*$_d$ are that the first platoon's data is adjusted to the "worst case" behavior of the first platoon.

$$\text{safe-measure}_u = \max((x_{iu1} - \delta) + (\dot{x}_{iu1} - \dot{\delta})t'' - \frac{a_{min}t''^2}{2} - x_{new2} - \frac{(\dot{x}_{new2})^2 - ((\dot{x}_{iu1} - \dot{\delta}) - a_{min}t'')^2 - (v_{allow})^2}{2a_{min}}, (\dot{x}_{iu1} - \dot{\delta}) - a_{min}t'' - \dot{x}_{new2} + v_{allow}),$$

where $t'' = \min(d_i + d_o, \frac{\dot{x}_{iu1} + \dot{\delta}}{a_{min}})$.

6.2 Correctness of U_2

A straightforward simulation relation shows that the *Uncertain-Controller* system implements D_2. This simulation relation f is the identity on all state components of D_2.

First we show that if the old *safe-measure*$_d$ (the one used in the delayed case) is non-positive in some state of a trajectory of *Uncertain-Controller*, then the new controller (the one that has an inbound uncertainty), will also output maximum deceleration, same as the old (delayed only) controller would. Formally,

Lemma 13. *Let w be an I-trajectory of* Uncertain-Controller. *If collided = false in $w(0)$, then $\forall t \in I$, such that safe-measure$_d \leq 0$, $\ddot{x}_{o2} = -a_{min}$ and safe-measure$_u \leq 0$.*

Then we are able to prove that f is the simulation relation from the composed system *Uncertain-Controller* to D_2.

Theorem 14. *f (an identity relation on all the external state components of D_2) is a forward simulation from the Uncertain-Controller system to D_2.*

We have already proven correctness of the delayed controller D_2, and we have also shown that *Uncertain-Controller* implements D_2; thus, our new uncertain system is also correct in a sense that it implements our correctness specification, *Safe-Platoons*.

7 Conclusion

The system consisting of two platoons moving on a single track has been modeled using hybrid I/O automata, including all the components (physical platoons, controllers, delay and uncertainty buffers), and the interactions between them. Safety conditions were formulated using invariant assertions. Correctness and optimality of controllers were proved using composition, simulation mappings and invariants, and the methods of mathematical analysis. Complexity (delays and uncertainty) was introduced gradually, using the levels of abstraction, which significantly simplified the proofs.

The case study describes formally a general controller that would guarantee the safety requirement regardless of the behavior of the leading platoon. Such a controller can be later reused to prove correctness of complicated maneuvers, such as merging and splitting, where the setup is similar.

In future work, we will extend the model to handle outbound uncertainty; use jerk instead of acceleration; motion in 2D planes. Also, we will consider cases with several platoons operating independently. Additional properties of the join maneuver, such as successful join, time optimality, and passenger comfort, will be studied; other maneuvers arising in this setting will be investigated using the same models.

References

1. R. Alur, C. Courcoubetis, T.A. Henzinger, P.H. Ho, X. Nicollin, A. Olivero, J Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.
2. Michael S. Branicky, Ekaterina Dolginova, and Nancy Lynch. A toolbox for proving and maintaining hybrid specifications. Submitted for publication. To be presented at *HS'96: Hybrid Systems*, October 12-16, 1996, Cornell University, Ithacs, NY.
3. Jonathan Frankel, Luis Alvarez, Roberto Horowitz, and Perry Li. Robust platoon maneuvers for AVHS. Manuscript, Berkeley, November 10, 1994.
4. John Lygeros. *Hierarchical Hybrid Control of Large Scale Systems.* PhD thesis, University of California, Department of Electrical Engineering, Berkeley, California, 1996.
5. John Lygeros, Datta N. Godbole, and Shankar Sastry. A game theoretic approach to hybrid system design. Technical Report UCB/ERL-M95/77, Electronic Research Laboratory, University of California Berkeley, October 1995.

6. Nancy Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. In *Proceedings of the Third AMAST Workshop on Real-Time Systems*, pages 1–22, Salt Lake City, Utah, March 1996.
7. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O automata. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control* (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of *Lecture Notes in Computer Science*, pages 496–510. Springer-Verlag, 1996.
8. O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *REX Workshop on Real-Time: Theory in Practice*, volume 600 of *Lecture Notes in Computer Science*, pages 447–484, Mook, The Netherlands, June 1991. Springer-Verlag.
9. Pravin Varaiya. Smart cars on smart roads: Problems of control. *IEEE Transactions on Automatic Control*, AC-38(2):195–207, 1993.
10. H. B. Weinberg and Nancy Lynch. Correctness of vehicle control systems: A case study. In *17th IEEE Real-Time Systems Symposium*, pages 62–72, Washington, D. C., December 1996. Complete version in Technical Report MIT/LCS/TR-685, Laboratory for Computer Science, Massachusetts Institute of Technology, February 1996. Masters Thesis.
11. H. B. Weinberg, Nancy Lynch, and Norman Delisle. Verification of automated vehicle protection systems. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control* (DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, October 1995), volume 1066 of *Lecture Notes in Computer Science*, pages 101–113. Springer-Verlag, 1996.

This article was processed using the LaTeX macro package with LLNCS style

Preliminary Version of Paper Accepted to:
Hybrid Systems: Computation and Control
Berkeley, California, April 13–15, 1998

Formal Verification of Safety-Critical Hybrid Systems*

Carolos Livadas and Nancy A. Lynch

Laboratory for Computer Science
Massachusetts Institute of Technology
{clivadas,lynch}@theory.lcs.mit.edu

Abstract. This paper investigates how formal techniques can be used for the analysis and verification of hybrid systems [1,5,7,16] — systems involving both discrete and continuous behavior. The motivation behind such research lies in the inherent similarity of the hierarchical and decentralized control strategies of hybrid systems and the communication and operation protocols used for distributed systems in computer science. This paper focuses on the use of hybrid I/O automata [11,12] to model, analyze, and verify safety-critical hybrid systems that use emergency control subsystems to prevent the violation of their safety requirements. The paper is split into two parts. First, we develop an abstract model of a *protector* — an emergency control component that guarantees that the physical plant at hand adheres to a particular safety requirement. The abstract protector model specialized to a particular physical plant and a particular safety requirement constitutes the specification of a protector that enforces the particular safety property for the particular physical plant. The correctness proof of the abstract protector model leads to simple correctness proofs of the implementations of particular protectors. In addition, the composition of independent protectors, and even dependent protectors under mild conditions, guarantees the conjunction of the safety properties guaranteed by the individual protectors being composed. Second, as a case study, we specialize the aforementioned abstract protector model to simplified versions of the personal rapid transit system (PRT 2000™) under development at Raytheon Corporation and verify the correctness of overspeed and collision avoidance protectors. Such correctness proofs are repeated for track topologies ranging from a single track to a directed graph of tracks involving Y-shaped merges and diverges.

* This research was performed at the Theory of Distributed Systems Group of the Department of Electrical and Engineering and Computer Science of the Massachusetts Institute of Technology. The research was supported by NSF Grant 9225124-CCR, U.S. Department of Transportation Contract DTRS95G-0001-YR.8, AFOSR-ONR Contracts F49620-94-1-0199 and F49620-97-1-0337, and ARPA Contracts N00014-92-J-4033 and F19628-95-C-0118.

1 Introduction

The recent trend of system integration and automation has encouraged the study of hybrid systems — systems that combine continuous and discrete behavior. Although the individual problems of continuous and discrete behavior have been extensively analyzed by control theory and formal analysis, respectively, their combination has recently been aggressively studied. In particular, the automation in various safety-critical systems, such as automated transportation systems, has indicated the need for formal approaches to system analysis, design, and verification. Automated highway systems [2,8], personal rapid transit systems [6,17], and air traffic control systems [9,15] have served as benchmark problems for the development of techniques to analyze, design, and verify hybrid systems.

Many of the safety-critical systems in use today abide by the engineering paradigm of using an emergency control, or protection, subsystem to prevent the violation of the system's safety requirements. In this paper we present a formal framework for the analysis of systems that adhere to this engineering paradigm. The framework is used to prove the correctness of such protection subsystems in an effort to provide indisputable system safety guarantees. The formal approach to the analysis of such systems has several advantages. Formal analysis yields a precise specification of the system and its safety requirements, provides insight as to the location of possible design errors, and minimizes the duplication of verification effort when such errors are corrected. The technique of system validation through exhaustive testing lacks the insightful feedback and requires full-fledged regression testing when design errors are detected.

In this paper, we use *hybrid I/O automata* [11,12] — an extension of *timed I/O automata* [4,14] — to define an abstract model of a *protector* — a subsystem that guarantees that the physical plant adheres to a particular safety requirement. The abstract protector model is parameterized in terms of the physical plant, the safety requirement, and several other quantities. The instantiation of the abstract protector, obtained by specifying the abstract protector's parameters, constitutes the specification of a protector that guarantees a particular safety property for a particular physical plant model. The proof of correctness of the abstract protector model minimizes the effort in verifying the correct operation of a particular protector implementation. In fact, such correctness proofs get reduced to simple simulations from the protector implementations to the particular instantiation of the abstract protector model. As a case study, we apply the formal framework developed towards the verification of overspeed and collision protection subsystems for simplified models of the personal rapid transit system (PRT 2000™) under development at Raytheon Corporation. The case studies presented in this paper extend the work of Weinberg, Lynch, and Delisle [17] by introducing a powerful formal framework that allows more complete system models to be used. The actual PRT 2000™ system is comprised of 4-passenger vehicles that travel on an elevated guideway of tracks involving Y-shaped merges and diverges and provide point-to-point passenger transportation. In this treatment, we verify the correct operation of overspeed and collision avoidance protectors for track topologies ranging from a single track to a directed

graph of tracks involving Y-shaped merges and diverges. A detailed treatment of the work presented in this paper can be found in Ref. 6.

2 Hybrid I/O Automata

A hybrid I/O automaton A is a (possibly) infinite state model of a system involving both discrete and continuous behavior. The automaton $A = (U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, \mathcal{D}, \mathcal{W})$ consists of three disjoint sets U, X, and Y of variables (*input*, *internal*, and *output* variables, respectively), three disjoint sets Σ^{in}, Σ^{int}, and Σ^{out} of *actions* (*input*, *internal*, and *output* actions, respectively), a nonempty set Θ of *initial states*, a set $\mathcal{D}$ of *discrete transitions* and a set $\mathcal{W}$ of trajectories over V, where $\Sigma = \Sigma^{in} \cup \Sigma^{int} \cup \Sigma^{out}$ and $V = U \cup X \cup Y$. The initial states, the discrete transitions, and the trajectories of any HIOA A must however satisfy several technical conditions which are omitted here. For a detailed presentation of the HIOA model, the reader is referred to Refs. 11 and 12.

Variables in the set V are typed; that is, each variable $v \in V$ ranges over the set of values *type*(v). A *valuation* of V, also referred to as a *state* of A, is a function that associates to each variable v of V a value in *type*(v). The set of all valuations of V, or equivalently the set of all states of A, is denoted by $\boldsymbol{V}$, or equivalently *states*(A). Letting $v \in V$ and $S_v \subseteq$ *type*(v), we use the notation $v :\in S_v$ to denote the assignment of an arbitrary element of the set S_v to the variable v. Similarly, letting $S_V \subseteq \boldsymbol{V}$, we use the notation $V :\in S_V$ to denote the assignment of an element of the set *type*(v) to the variable v, for each v in V, such that the resulting valuation of V is an arbitrary element of the set S_V. Letting s be a state of A, *i.e.*, $s \in \boldsymbol{V}$, and $V' \subseteq V$, we define the *restriction* of s to V', denoted by $s\lceil V'$, to be the valuation s' of the variables of V' in s. Letting $\boldsymbol{X} \subseteq \boldsymbol{V}$, we say that $\boldsymbol{X}$ *is* V'*-determinable* if for all $x \in \boldsymbol{X}$ and $s \in \boldsymbol{V}$, such that $x\lceil V' = s\lceil V'$, it is the case that $s \in \boldsymbol{X}$. The continuous time evolution of the valuations of the variables in V is described by a *trajectory* w over V; that is, a function $T_I \to \boldsymbol{V}$, where T_I is a left-closed interval of $\mathbb{R}^{\geq 0}$ with left endpoint equal to 0. The *limit time* of w, denoted by $w.ltime$, is defined to be the supremum of the domain of w, *dom*(w). We define the *first state* of a trajectory w, denoted by $w.fstate$, to be the state $w(0)$. Moreover, if the domain of a trajectory w is right-closed, then we define the *last state* of w, denoted by $w.lstate$, to be the state $w(w.ltime)$.

A *hybrid execution fragment* α of A is a finite or infinite alternating sequence $w_0 a_1 w_1 a_2 w_2 \cdots$, where $w_i \in \mathcal{W}$, $a_i \in \Sigma$, and if w_i is not the last trajectory of α then w_i is right-closed and the discrete transition $(w_i.lstate, a_{i+1}, w_{i+1}.fstate)$ is in $\mathcal{D}$, or equivalently $w_i.lstate \xrightarrow{a_{i+1}}_A w_{i+1}.fstate$. If $w_0.fstate \in \Theta$ then α is a *hybrid execution* of A. If $R \subseteq$ *states*(A) and $s, s' \in R$, then s' is *R-reachable from* s provided that there is a hybrid execution fragment of A that starts in s, ends in s', and all of whose states are in the set R.

A *superdense time* in an execution fragment α of A is a pair (i, t), where $t \leq w_i.ltime$. We totally order superdense times in the execution fragment α

lexicographically. An *occurrence of* a state s *in* an execution fragment α of A is a triple (i,t,s) such that (i,t) is a superdense time in α and $s = w_i(t)$. State occurrences in α are ordered according to their superdense times. If S is a set of states of A and α is an execution fragment of A, then $past(S,\alpha)$ is the set of state occurrences (i,t,s) in α such that either $s \in S$ or there is a previous state occurrence (i',t',s') in α with $s' \in S$.

Two HIOA A_1 and A_2 are *compatible* if $X_i \cap V_j = Y_i \cap Y_j = \Sigma_i^{int} \cap \Sigma_j = \Sigma_i^{out} \cap \Sigma_j^{out} = \emptyset$, for $i,j \in \{1,2\}, i \neq j$. If A_1 and A_2 are compatible then their *composition* $A_1 \times A_2$ is defined to be the tuple $A = (U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, \mathcal{D}, \mathcal{W})$ given by $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$, $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$, $\Sigma^{in} = (\Sigma_1^{in} \cup \Sigma_2^{in}) - (\Sigma_1^{out} \cup \Sigma_2^{out})$, $\Sigma^{int} = \Sigma_1^{int} \cup \Sigma_2^{int}$, $\Sigma^{out} = \Sigma_1^{out} \cup \Sigma_2^{out}$, $\Theta = \{s \in \boldsymbol{V} \mid s\lceil V_1 \in \Theta_1 \ \wedge \ s\lceil V_2 \in \Theta_2\}$ and sets of discrete transitions $\mathcal{D}$ and trajectories $\mathcal{W}$ each of whose elements projects to discrete transitions and trajectories, respectively, of A_1 and A_2.

Two HIOA A_1 and A_2 are *comparable* if they have the same external interface, *i.e.*, $U_1 = U_2$, $Y_1 = Y_2$, $\Sigma_1^{in} = \Sigma_2^{in}$, and $\Sigma_1^{out} = \Sigma_2^{out}$. If A_1 and A_2 are comparable, then $A_1 \leq A_2$ is defined to denote that the hybrid traces of A_1 are included in those of A_2; that is, $A_1 \leq A_2 \stackrel{\Delta}{=} h\text{-}traces(A_1) \subseteq h\text{-}traces(A_2)$. If $A_1 \leq A_2$, then we say that A_1 *implements* A_2.

3 Protected Plant Systems

A *protected plant system* is modeled abstractly as a *physical plant* interacting with a *protection system.* The protection system is modeled as the composition of a set of *protectors* each of which is supposed to enforce a particular safety requirement of the physical plant. Our model is abstract in the sense that it does not specify any of the details or safety requirements of the physical plant.

The physical plant and each of the protectors are modeled as HIOA. The *physical plant PP* is an automaton that is assumed to be interacting with the protectors through the set J of communication channels, which are referred to as *ports.* The input action set Σ_{PP}^{in}, the output action set Σ_{PP}^{out}, and the input variable set U_{PP} are partitioned into subsets $\Sigma_{PP_j}^{in}$, $\Sigma_{PP_j}^{out}$, and U_{PP_j}, respectively, one for each port j. We use the letter p to denote a state of PP and P to denote a set of states of PP. A *protector* A for the physical plant PP and the port set $K \subseteq J$ is an automaton that is compatible with PP and whose output actions are exactly the input actions of PP on ports in K, whose output variables are exactly the input variables of PP on ports in K, and all of whose input actions and input variables are outputs of PP. It can easily be shown that the composition of two distinct protectors is itself a protector.

Letting S, R, and G be particular sets of states of PP, a protector automaton A for PP and ports K *guarantees* G *in* PP *from* S *given* R provided that every finite execution of the composition $PP \times A$ starting in a state in S that only involves states in R ends in a state in G regardless of the inputs that arrive at PP on ports other that those in K. Two protectors are *dependent*, if the

correct operation of one relies on the correct operation of the other, and *independent*, otherwise. The following theorems express the *substitutivity* condition — the condition under which the implementation of a protector is *correct* with respect to its specification — and the *compositional* conditions — conditions under which the composition of independent or dependent protectors guarantees the conjunction of the safety properties guaranteed by the protectors being composed.

Theorem 1 (Substitutivity). *Let A_1 and A_2 be two protector automata for the same port set K of a physical plant automaton PP, and suppose that $A_1 \leq A_2$. If A_2 guarantees G in PP from S given R, then A_1 guarantees G in PP from S given R.*

Theorem 2 (Independent Protector Composition). *Suppose that $A_1, A_2, \ldots, A_k$ are protector automata for a physical plant automaton PP, with respective port sets $K_1, K_2, \ldots, K_k$, where $K_i \cap K_{i'} = \emptyset$, for all $i, i' \in \{1, \ldots, k\}, i \neq i'$. Suppose that each of the protectors A_i, for all $i \in \{1, \ldots, k\}$, guarantees G_i from S_i given R_i. If the protectors $A_1, A_2, \ldots, A_k$ are compatible, then their composition $\prod_{i \in \{1,\ldots,k\}} A_i$ is a protector for PP that guarantees $\bigcap_{i \in \{1,\ldots,k\}} G_i$ from $\bigcap_{i \in \{1,\ldots,k\}} S_i$ given $\bigcap_{i \in \{1,\ldots,k\}} R_i$.*

Theorem 3 (Dependent Protector Composition). *Suppose that $A_1, A_2, \ldots, A_k$ are protector automata for a physical plant automaton PP, with respective port sets $K_1, K_2, \ldots, K_k$, where $K_i \cap K_{i'} = \emptyset$, for all $i, i' \in \{1, \ldots, k\}, i \neq i'$. Suppose that each of the protector automata A_i, for all $i \in \{1, \ldots, k\}$, guarantees G_i from S_i given $R_i \bigcap \left(\bigcap_{i' \in \{1,\ldots,k\}, i' \neq i} G_{i'} \right)$.*

Assume that α is any finite execution of the system $PP \times \prod_{i \in \{1,\ldots,k\}} A_i$ starting from a state in $\bigcap_{i \in \{1,\ldots,k\}} S_i$ and all of whose states are in the set $\bigcap_{i \in \{1,\ldots,k\}} R_i$. Then, one of the following holds:

1. *Every state in α is in $\bigcap_{i \in \{1,\ldots,k\}} G_i$.*
2. *The finite execution α can be written as $\alpha_1 \frown \alpha_2$, where*
   (a) *all state occurrences in α_1, except possibly the last, are in the set of states $\bigcap_{i \in \{1,\ldots,k\}} G_i$,*
   (b) *if the last state occurrence in α_1 is in $\overline{G_i}$, for some $i \in \{1, \ldots, k\}$, then there exists $i' \in \{1, \ldots, k\}, i' \neq i$, such that the last state occurrence in α_1 is in $\overline{G_{i'}}$, and*
   (c) *all state occurrences in α_2, except possibly the first, are in the set of states $\bigcap_{i \in I} past(\overline{G_i}, \alpha)$, for some $I \subseteq \{1, \ldots, k\}$, where $|I| \geq 2$.*

In loose terms, Theorem 3 states that the composition of dependent protectors guarantees the conjunction of the safety properties guaranteed by the protectors being composed provided a single action or trajectory of the composed system can cause the violation of *at most* one of the safety properties guaranteed by the protectors being composed.

4 An Abstract Protector

The abstract protector automaton is parameterized in terms of the automaton PP, the subsets R, G, and S of the states of PP, the port index j, and the positive real-valued sampling period d. The PP automaton represents the physical plant being modeled. The set R, also referred to as the set of *reliance*, is the set of states to which we restrict the states of the PP automaton while considering a particular protector. This set is usually comprised of states satisfying a particular property of the physical plant that is required by the protector under consideration. The set G, also referred to as the set of *guarantee*, is the set of states to which the protector is designed to constrain the PP automaton. The set S is a set of states from which the protector under consideration is said to guarantee G given R; that is, given that the states of the PP automaton are restricted to the set R, the protector guarantees that every finite execution starting from an initial state in S ends in a state in G. The port index j and the sampling period d denote the port and the sampling period with which the abstract protector interacts with the PP automaton. Thus, an instantiation of the abstract protector automaton $Abs(PP, S, R, G, j, d)$ is obtained by specifying the parameters PP, *etc.*

To begin, we define several functions and sets that are useful in the definition of the abstract protector $Abs(PP, S, R, G, j, d)$. Although, formal definitions of these functions and sets are presented in Table 1, their informal interpretations follow. First, we define a function, $future_{PP,R,j}$, that yields the set of states of PP that are R-reachable from the given subset of R within an amount of time in the given subset of $\mathbb{R}^{\geq 0}$, under the constraint that no input actions arrive on port j of the PP automaton. We define a function, $no\text{-}op_{PP,R,j}$, which yields, for a given state in R, the set of input actions on port j of the PP automaton that do not affect the state of the PP automaton, provided they are executed prior to either time-passage, or other input actions on port j. For any state p in R, the input actions in the set $no\text{-}op_{PP,R,j}(p)$ are referred to as no-op input actions on port j of PP for the state p. We define a set, $very\text{-}safe_{PP,R,G,j}$, which is comprised of the states of PP that satisfy R and from which all R-reachable states of PP with no input actions on port j are in G. The set $very\text{-}safe_{PP,R,G,j}$ may be interpreted as the set consisting of the states from which the PP automaton is bound to remain within the set G provided that it remains within the set R and the protector on port j does not retract or issue additional protective actions. We define a set, $safe_{PP,R,G,j}$, which is comprised of the states of PP that satisfy R and from which the protector on port j has a "winning protective strategy"; that is, for any state p in $safe_{PP,R,G,j}$ there exists an input action on port j of the PP automaton whose immediate execution — its execution prior to any time-passage with the possibility that its execution follows an arbitrary number of discrete actions other than input actions on port j — guarantees that all subsequent R-reachable states of PP with no input actions on port j are in G; that is, the state following the execution of the particular input action of PP on port j is in the set $very\text{-}safe_{PP,R,G,j}$. We overload the notation $safe_{PP,R,G,j}$ by defining a function, $safe_{PP,R,G,j}$, which yields the states of PP that satisfy R and for which the immediate execution of the given input action on port j — its

Table 1 Terminology for the abstract protector $Abs(PP, S, R, G, j, d)$.

$future_{PP,R,j} : \mathcal{P}(R) \times \mathcal{P}(\mathbb{R}^{\geq 0}) \to \mathcal{P}(R)$, defined by:

$p \in future_{PP,R,j}(P, T)$, where $P \subseteq R$ and $T \subseteq \mathbb{R}^{\geq 0}$, if and only if p is R-reachable from some $p' \in P$ via a finite execution fragment α of PP with no input actions on port j and with $\alpha.ltime \in T$.

$no\text{-}op_{PP,R,j} : R \to \mathcal{P}(\Sigma^{in}_{PP_j})$, defined by:

$\pi \in no\text{-}op_{PP,R,j}(p)$ if and only if π is an input action on port j of PP such that for all $p', p'' \in R$ satisfying $p' \in future_{PP,R,j}(p, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' = p'$.

$very\text{-}safe_{PP,R,G,j} \subseteq R$, defined by:

$p \in very\text{-}safe_{PP,R,G,j}$ if and only if $future_{PP,R,j}(p, \mathbb{R}^{\geq 0}) \subseteq G$.

$safe_{PP,R,G,j} \subseteq R$, defined by:

$p \in safe_{PP,R,G,j}$ if and only if both of the following hold:

1. $future_{PP,R,j}(p, 0) \subseteq G$.
2. There exists an input action π on port j, such that for every $p', p'' \in R$ satisfying $p' \in future_{PP,R,j}(p, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' \in very\text{-}safe_{PP,R,G,j}$.

$safe_{PP,R,G,j} : \Sigma^{in}_{PP_j} \to \mathcal{P}(R)$, defined by:

$p \in safe_{PP,R,G,j}(\pi)$ if and only if both of the following hold:

1. $future_{PP,R,j}(p, 0) \subseteq G$.
2. For every $p', p'' \in R$ such that $p' \in future_{PP,R,j}(p, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' \in very\text{-}safe_{PP,R,G,j}$.

$delay\text{-}safe_{PP,R,G,j} : \mathbb{R}^{\geq 0} \to \mathcal{P}(R)$, defined by:

$p \in delay\text{-}safe_{PP,R,G,j}(t)$ if and only if both of the following hold:

1. $future_{PP,R,j}(p, [0, t]) \subseteq G$.
2. $future_{PP,R,j}(p, t) \subseteq safe_{PP,R,G,j}$.

execution prior to any time-passage with the possibility that its execution follows an arbitrary number of discrete actions other than input actions on port j — guarantees that all subsequent R-reachable states of PP with no input actions on port j are in G; that is, the state following the execution of the given input action on port j is in the set $very\text{-}safe_{PP,R,G,j}$. Finally, we define a function, $delay\text{-}safe_{PP,R,G,j}$, which yields the set of states of PP that satisfy R and for which all states R-reachable within the given amount of time and with no input actions on port j are in G, and all states R-reachable in exactly the given amount of time and with no input actions on port j are in $safe_{PP,R,G,j}$.

We proceed by stating the various assumptions made about the physical plant PP and the abstract protector $Abs(PP, S, R, G, j, d)$. We assume that the PP automaton has no input variables on port j, for all $j \in J$; that is, the protectors control the state of the physical plant only through input actions. A consequence of this assumption is that the environment action of the PP automaton is stuttering. Moreover, we assume that the PP automaton has no output actions on port j, for all $j \in J$. The physical plant is modeled as a passive system in the sense that the protectors observe the state of the plant only through output variables. We assume that there exist no-op input actions on port j for

Fig. 1 $Sensor_j$ automaton definition.

Actions: Input: e, the environment action
Output: $\texttt{snapshot}(y)_j$, for each valuation y of Y_{PP}

Variables: Input: $u \in type(u)$, for all $u \in Y_{PP}$,
initially $u \in type(u)$, for each $u \in Y_{PP}$
Internal: $now_j \in \mathbb{R}^{\geq 0}$, initially 0
$next\text{-}snap_j \in \mathbb{R}^{\geq 0}$, initially 0

Discrete Transitions:

e
Eff: $Y_{PP} :\in \boldsymbol{Y_{PP}}$

$\texttt{snapshot}(y)_j$
Pre: $next\text{-}snap_j = now_j$
y is current valuation of Y_{PP}
Eff: $Y_{PP} :\in \boldsymbol{Y_{PP}}$
$next\text{-}snap_j := now_j + d$

Trajectories:

for all $u \in Y_{PP}$
u assumes arbitrary values in $type(u)$ throughout w
$next\text{-}snap_j$ is constant throughout w
for all $t \in T_I$
$w(t).now_j = w(0).now_j + t$
$w(t).now_j \leq w(t).next\text{-}snap_j$

every state of the PP automaton in the set R. We assume that membership of a state of the PP automaton in the set $safe_{PP,R,G,j}$ is determinable from the output variables of the PP automaton, *i.e.*, the set $safe_{PP,R,G,j}$ is Y_{PP}-determinable. Moreover, we assume that for any state in the set $safe_{PP,R,G,j}$, an appropriate action to guarantee safety can be determined from the output variables of the PP automaton, *i.e.*, the variables in Y_{PP}. For any valuation y of the output variables Y_{PP} of the PP automaton, we use the notation $y \in safe_{PP,R,G,j}$ to denote the existence of a state $p \in safe_{PP,R,G,j}$ such that $p\lceil Y_{PP} = y$. We assume that the state information provided by the output variables of the PP automaton is sufficient to determine membership of any state of the PP automaton in the sets R and G, *i.e.*, the sets R and G are Y_{PP}-determinable. Moreover, we assume that the set of start states S is a subset of the set $safe_{PP,R,G,j}$.

The protector is defined as the composition of a *sensor automaton* (Figure 1) and a *discrete controller automaton* (Figure 2). Both the sensor and the discrete controller are described abstractly in terms of PP, S, R, G, j, and d and are respectively denoted $Sensor(PP, S, R, G, j, d)$ and $DC(PP, S, R, G, j, d)$. At intervals of d time units, the sensor automaton samples the output variables of the PP automaton. The discrete controller automaton is rather nondeterministic. Based on the output state information of the PP automaton sampled by the sensor automaton, the discrete controller automaton issues protective actions so as to guarantee that (i) the PP automaton remains within the set G up to the next sampling point, and (ii) the state of the PP automaton at the next sampling point is in the set $safe_{PP,R,G,j}$. The nondeterminism in the description of the DC_j automaton allows the freedom to choose any response that satisfies the given conditions — however, in a discrete controller automaton implementation,

Fig. 2 DC_j automaton definition.

Actions: Input: e, the environment action (stuttering)
$\texttt{snapshot}(y)_j$, for each valuation y of Y_{PP}
Output: π, for all $\pi \in \Sigma^{in}_{PP_j}$
Variables: Internal: $send_j \in \Sigma^{in}_{PP_j} \cup \{null\}$, initially *null*
Discrete Transitions:

e
Eff: None

π
Pre: $send_j = \pi$
Eff: $send_j := null$

$\texttt{snapshot}(y)_j$
Eff: if $y \in safe_{PP,R,G,j}$ then
$send_j :\in \{\phi \in \Sigma^{in}_{PP_j} \mid$
$\forall p, p', p'' \in R$ such that
$p\lceil Y_{PP} = y, p' \in future_{PP,R,j}(p, 0)$,
and $p' \xrightarrow{\phi}_{PP} p''$,
it is the case that
$p'' \in delay\text{-}safe_{PP,R,G,j}(d)\}$
else
$send_j :\in \Sigma^{in}_{PP_j}$

Trajectories:
$w.send_j \equiv null$

a response that least restricts the future states of the physical plant automaton PP would be preferred because it would represent a weaker protective action.

Theorem 4. *$Abs(PP, S, R, G, j, d)$ guarantees G in PP from S given R.*

The correctness proof of a particular protector implementation involves defining the particular protector's specification as the instantiation of the abstract protector for particular definitions of PP, *etc.* and showing that the particular protector implementation is correct with respect to the particular instantiation of the abstract protector. The first step simply involves specifying the parameters PP, *etc.* The second step is simplified by choosing the protector implementation to be the composition of the sensor automaton $Sensor(PP, S, R, G, j, d)$ and a discrete automaton that is chosen so as to guarantee the effect clause of the $\texttt{snapshot}(y)_j$ action in $DC(PP, S, R, G, j, d)$. Thus, the correctness proof of the implementation is reduced to a simulation from the implementation of the discrete controller automaton to its specification.

5 Modeling the PRT 2000™

In this section, we present a model for a simplified version of the PRT 2000™ whose track topology involves a single track. The model, VEHICLES, which is presented in Figure 3, is a HIOA that conforms to the restrictions and assumptions made about the PP automaton in Sections 3 and 4. It involves n vehicles of identical dimensions and acceleration/deceleration capabilities traveling on a single track. Its state variables include the position x_i, the velocity $\dot{x}_i$, and the acceleration $\ddot{x}_i$ of each vehicle i in the set of vehicles I and several other variables that record whether each vehicle has collided *into* each other vehicle

(*collided*(i, i'), for $i' \in I, i' \neq i$), whether each vehicle is braking (*brake*(i), for $i \in I$), and whether each protector j in the set of protectors J is requesting each particular vehicle to brake (*brake-req*(i, j), for $i \in I$ and $j \in J$). Several properties of the physical plant are enforced by restricting the states of the VEHICLES automaton to the set *VALID* (Appendix A). In particular, we assume that the vehicles occupy non-overlapping sections of the track, the vehicles are only allowed to move forward on the track, the non-malfunctioning vehicle acceleration/deceleration capabilities to be within the interval $[\ddot{c}_{min}, \ddot{c}_{max}]$, and the non-malfunctioning braking deceleration to be given by $\ddot{c}_{brake}$, if the vehicle is moving forward, and 0, otherwise.

The formal definitions of the derived variables and sets of the VEHICLES automaton are shown in Appendix A. For brevity, we only give informal definitions of the key derived variables. Each of the variables E_i, for $i \in I$, denotes the *extent* of the vehicle i; that is, the section of the track *occupied* by the vehicle i. It is defined as the section of track ranging from the position of the rear of the vehicle i to the point on the track that is a distance of c_{len} downstream of the rear of the vehicle i — a distance that specifies the minimum allowable separation between vehicles, *i.e.*, $E_i = [x_i, x_i + c_{len}]$, for $i \in I$. Each of the variables O_i, for $i \in I$, denotes the section of the track that the vehicle i *owns*; that is, the range extending from the current position of the rear of the vehicle i to the point on the track that the vehicle can reach even if it is braked immediately. Each of the variables $C_i(t)$, for $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, denotes the section of the track that the vehicle i *claims* within t time units; that is, the range extending from the current position of the rear of the vehicle i to the point on the track that the vehicle i can reach if it is braked after t time units and assuming worst-case vehicle behavior up to the point in time when it is braked. Moreover, each of the variables *collided*$(*, i, *)$, for $i \in I$, denotes whether the vehicle i has ever been involved in a collision. Some auxiliary sets for the vehicles automaton that will be used in the following sections are defined in Appendix B.

The input actions of the VEHICLES automaton are the environment action e and the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, for $i \in I$ and $j \in J$. Since the VEHICLES automaton has no input variables, the environment action e is stuttering. Each of the actions $\texttt{brake}(i)_j$ and $\texttt{unbrake}(i)_j$, for $i \in I$ and $j \in J$, correspond to actions performed by the protector j instructing the vehicle i to apply or release its "emergency" brake, respectively. Each $\texttt{brick-wall}(i)$ action, for $i \in I$, models the instantaneous stopping of the vehicle i — as if it hit a brick wall. Thereafter however, the vehicle i is allowed to reinitiate forward motion. Each $\texttt{colliding-pair}(i, i')$ action, for $i, i' \in I, i \neq i'$, records the fact that the vehicle i has collided into the vehicle i'. Since the trailing vehicle is the only vehicle that can prevent the collision through braking, a collision is recorded only by the trailing vehicle as if the trailing vehicle were the only vehicle liable for the particular collision. Each $\texttt{collision-effects}(i)$ action, for $i \in I$, models the adverse effects of a collision involving the vehicle i and may be executed, even repeatedly, at any instant of time following the first collision involving the vehicle i. Thus, the malfunctioning apparatus of any vehicle i, for $i \in I$, is modeled

Fig. 3 The VEHICLES automaton.

Actions:

Input:

e, the environment action (stuttering)

$\texttt{brake}(i)_j$, for all $i \in I, j \in J$

$\texttt{unbrake}(i)_j$, for all $i \in I, j \in J$

Internal:

$\texttt{colliding-pair}(i, i')$, for all $i, i' \in I, i' \neq i$

$\texttt{collision-effects}(i)$, for all $i \in I$

$\texttt{brick-wall}(i)$, for all $i \in I$

Variables

Internal:

$\ddot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\ddot{x}_i \in \mathbb{R}$

$brake(i) \in \texttt{Bool}$, for all $i \in I$, initially $\texttt{False}$

$brake\text{-}req(i, j) \in \texttt{Bool}$, for all $i \in I, j \in J$, initially $\texttt{False}$

Output:

$x_i \in \mathbb{R}$, for all $i \in I$, initially $x_i \in \mathbb{R}$

$\dot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\dot{x}_i \in \mathbb{R}$

$collided(i, i') \in \texttt{Bool}$, for all $i, i' \in I, i' \neq i$, initially $\texttt{False}$

subject to *VALID*

Discrete Transitions:

e

Eff: None

$\texttt{brake}(i)_j$

Eff: $brake\text{-}req(i, j) := \texttt{True}$
if $\neg brake(i)$ then
$brake(i) := \texttt{True}$
if $\dot{x}_i = 0$ then $\ddot{x}_i := 0$
else $\ddot{x}_i := \ddot{c}_{brake}$

$\texttt{unbrake}(i)_j$

Eff: $brake\text{-}req(i, j) := \texttt{False}$
if $brake(i)$
$\wedge(\neg \vee_{k \in J}\ brake\text{-}req(i, k))$
then
$brake(i) := \texttt{False}$
$\ddot{x}_i :\in [\ddot{c}_{min}, \ddot{c}_{max}]$

$\texttt{colliding-pair}(i, i')$

Pre: $\neg collided(i, i')$
$\wedge(E_i \cap E_{i'} \neq \emptyset)$
$\wedge(x_i < \min(E_i \cap E_{i'}))$

Eff: $collided(i, i') := \texttt{True}$

$\texttt{collision-effects}(i)$

Pre: $collided(*, i, *)$

Eff: $\dot{x}_i :\in \mathbb{R}^{\geq 0}$
$\ddot{x}_i :\in \mathbb{R}$

$\texttt{brick-wall}(i)$

Pre: $\texttt{True}$

Eff: $\dot{x}_i := 0$
if $brake(i)$ then
$\ddot{x}_i := 0$
else
$\ddot{x}_i :\in [0, \ddot{c}_{max}]$

Trajectories:

for all $i, i' \in I, i \neq i'$, $collided(i, i')$ is constant throughout w

for all $i \in I$ and $j \in J$, $brake(i)$ and $brake\text{-}req(i, j)$ are constant throughout w

for all $i, i' \in I, i \neq i'$

the function $w.\ddot{x}_i$ is integrable

for all $t \in T_I$

$w(t).\dot{x}_i = w(0).\dot{x}_i + \int_0^t w(s).\ddot{x}_i\ ds$

$w(t).x_i = w(0).x_i + \int_0^t w(s).\dot{x}_i\ ds$

if $\neg w.collided(i, i')$
$\wedge(w(t).E_i \cap w(t).E_{i'} \neq \emptyset)$
$\wedge(w(t).x_i < \min(w(t).E_i \cap w(t).E_{i'}))$
then
$t = w.ltime$

subject to *VALID*

by succeeding each of the discrete actions with a collision-effects(i) action for the malfunctioning vehicle.

The trajectories of the VEHICLES automaton model the continuous evolution of the state of the VEHICLES automaton. If during a trajectory a vehicle i collides into a vehicle i' for the first time, the trajectory is stopped so that the collision can be recorded.

6 Example Overspeed and Collision Avoidance Protectors

6.1 Example 1: Overspeed Protection System

In this section, we present a protector, called OS-PROT, that prevents the vehicles of the VEHICLES automaton from exceeding a prespecified global speed limit $\dot{c}_{max}$, provided that they do not collide among themselves. The protector OS-PROT is defined to be the composition of n separate copies of another protector called OS-PROT-SOLO$_i$, one copy for each vehicle $i \in I$. Each of the OS-PROT-SOLO$_i$ protectors, for $i \in I$, guarantees that the vehicle i, does not exceed the speed limit $\dot{c}_{max}$, provided that no collisions among any of the vehicles occur. The braking strategy of the OS-PROT-SOLO$_i$ protector is to instruct the vehicle i to brake if it is capable of exceeding the speed limit $\dot{c}_{max}$ within the time until the next sampling point.

Let PP_i be the VEHICLES automaton of Figure 3, the port j_i and the sampling period d_i be the port and sampling period with which the protector OS-PROT-SOLO$_i$ communicates with the VEHICLES automaton, the set R_i be the set of states in which none of the vehicles have ever collided, *i.e.*, $R_i = P_{not\text{-}collided}$ (Appendix B), the set G_i be the set of states in which the vehicle i is at or below the speed limit, *i.e.*, $G_i = VALID - P_{overspeed(i)}$ (Appendix B), and the set S_i be the set $safe_{PP_i,R_i,G_i,j_i}$. We define the OS-PROT-SOLO$_i$ automaton to be the composition of $Sensor(PP_i, S_i, R_i, G_i, j_i, d_i)$ and the discrete controller automaton of Appendix C.

Lemma 1. *The protector* OS-PROT-SOLO$_i$ *guarantees* G_i *in* VEHICLES *starting from* S_i *given* R_i.

Corollary 1. *The protector* OS-PROT $= \prod_{i \in I}$ OS-PROT-SOLO$_i$ *for the* VEHICLES *automaton guarantees* $\bigcap_{i \in I} G_i$ *in* VEHICLES *starting from* $\bigcap_{i \in I} S_i$ *given* $P_{not\text{-}collided}$.

Corollary 1 follows directly from Lemma 1 and Theorem 2.

6.2 Example 2: Collision Avoidance on a Single Track

In this section, we present a protector, called CL-PROT, that prevents the vehicles of the VEHICLES automaton from colliding among themselves, provided that they are all abiding by the speed limit $\dot{c}_{max}$. The protector CL-PROT is defined to be the composition of n separate copies of another protector called CL-PROT-SOLO$_i$,

one copy for each vehicle $i \in I$. Each of the OS-PROT-SOLO$_i$ protectors, for $i \in I$, guarantees that the vehicle i does not collide into any of the vehicles it trails, provided that all the vehicles in the VEHICLES automaton are abiding by the speed limit and that all other vehicles $i' \in I, i' \neq i$, do not collide into any of the vehicles they respectively trail. The braking strategy of the CL-PROT-SOLO$_i$ protector is to instruct the vehicle i to brake if it has a d_i time unit claim overlap with any of the vehicles it trails. The rationale behind this braking strategy is that a collision between two vehicles in the VEHICLES automaton can only be prevented by instructing the trailing vehicle to brake.

Let PP_i be the VEHICLES automaton of Figure 3, the port j_i and the sampling period d_i be the port and sampling period with which the protector CL-PROT-SOLO$_i$ communicates with the VEHICLES automaton, and the set G_i be the set of states in which the vehicle i has not collided into any of the other vehicles, *i.e.*, $G = VALID - P_{collided(i)}$ (Appendix B). Moreover, let the set R_i be the set of states in which all of the vehicles are abiding by the speed limit and in which each of the other vehicles has never collided into any other vehicle, *i.e.*, $R_i = P_{not\text{-}overspeed} \bigcap \left(\bigcap_{i' \in I, i' \neq i} G_{i'} \right)$ (Appendix B), and the set S_i be the set $safe_{PP_i,R_i,G_i,j_i}$. We define the CL-PROT-SOLO$_i$ automaton to be the composition of $Sensor(PP_i, S_i, R_i, G_i, j_i, d_i)$ and the discrete controller automaton of Appendix D.

Lemma 2. *The protector* CL-PROT-SOLO$_i$ *guarantees* G_i *in* VEHICLES *starting from* S_i *given* R_i.

Lemma 3. *The protector* CL-PROT $= \prod_{i \in I}$ CL-PROT-SOLO$_i$ *for the* VEHICLES *automaton guarantees* $\bigcap_{i \in I} G_i$ *in* VEHICLES *starting from* $\bigcap_{i \in I} S_i$ *given* $P_{not\text{-}overspeed}$.

Lemma 3 is shown by combining Lemma 2 and Theorem 3 and realizing that the second condition of Theorem 3 does not hold.

6.3 Example 3: Collision Avoidance on Merging Tracks

In this section, we present a protector, called MERGE-PROT, that guarantees that none of the n vehicles that are traveling on a track involving a Y-shaped merge collide, provided that they are all abiding by the speed limit $\dot{c}_{max}$. The MERGE-PROT protector is defined as the composition of $n(n-1)/2$ separate copies of another protector called MERGE-PROT-PAIR$_{\{i,i'\}}$, one copy for each unordered pair of vehicles $\{i, i'\}$, where $i, i' \in I, i \neq i'$. Each of these MERGE-PROT-PAIR$_{\{i,i'\}}$ protectors, for $i, i' \in I, i \neq i'$, guarantees that the vehicles i and i' do not collide into each other, provided that all the vehicles are abiding by the speed limit and the vehicles of all other vehicle pairs do not collide between themselves.

We augment the VEHICLES automaton to involve a track topology consisting of a Y-shaped merge. This is done by replacing the position component of a vehicle's state with a location component — a component that specifies the track on which the vehicle is traveling and the vehicle's position with respect to the

merge point — and update the definitions of the discrete steps and the trajectories of the VEHICLES automaton to handle the location variables. Furthermore, we replace the `brake` and `unbrake` input actions of the VEHICLES automaton with `protect` input actions which allow single protectors to instruct sets of vehicles to apply their "emergency" brakes. Finally, we augment the definitions of the discrete actions pertaining to vehicle collisions such that the blame for a particular collision is assigned to either only the trailing vehicle, if one vehicle collides into the other vehicle from behind, or both vehicles, if the vehicles collide sideways while merging. The resulting physical plant automaton is henceforth referred to as MERGE-VEHICLES (Appendix E).

Let $PP_{\{i,i'\}}$ be the MERGE-VEHICLES automaton. Let the port $j_{\{i,i'\}}$ and the sampling period $d_{\{i,i'\}}$ be the port and sampling period with which the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ communicates with the MERGE-VEHICLES automaton. Let $G_{\{i,i'\}}$ be the set of states in which the vehicles i and i' have not collided into each other, *i.e.*, $G_{\{i,i'\}} = VALID - P_{collided(i,i')} - P_{collided(i',i)}$ (Appendix B). Let $R_{\{i,i'\}}$ be the set of states of the MERGE-VEHICLES automaton in which all the vehicles are abiding by the speed limit and in which the vehicles of all other vehicle pairs have not collided into each other, *i.e.*, $R_{\{i,i'\}} = P_{not\text{-}overspeed} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$ (Appendix B). Finally, let $S_{\{i,i'\}}$ be the set $safe_{PP_{\{i,i'\}},R_{\{i,i'\}},G_{\{i,i'\}},j_{\{i,i'\}}}$. We define the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ to be the composition of $Sensor(PP_{\{i,i'\}}, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j_{\{i,i'\}}, d_{\{i,i'\}})$ and the discrete controller automaton of Appendix G.

The braking strategy of the MERGE-PROT-PAIR$_{\{i,i'\}}$ protector is as follows. The protector is allowed to brake the vehicles i and i' only if the sections of the track they claim in time $d_{\{i,i'\}}$ overlap. Given that the vehicles i and i' are indeed involved in such a claim overlap, there are two possible scenarios depending on whether the locations of the vehicles i and i' are comparable, or not. If their locations are comparable, then the vehicle i is instructed to brake if it trails the vehicle i'; otherwise, the vehicle i' is instructed to brake. On the other hand, if the vehicle locations are not comparable, the vehicle i is instructed to brake either if *only* the vehicle i' owns the merge point, or if both or neither vehicles own the merge point and the vehicle i is traveling on the `left` branch of the merge; otherwise, the vehicle i' is instructed to brake. In the latter case, we choose to brake the vehicle traveling on the `left` branch for no particular reason. In fact, it is plausible to brake either or both of the vehicles involved in the claim overlap.

Lemma 4. *The protector* MERGE-PROT-PAIR$_{\{i,i'\}}$ *guarantees that the* MERGE-VEHICLES *automaton remains within* $G_{\{i,i'\}}$ *starting from* $S_{\{i,i'\}}$ *given* $R_{\{i,i'\}}$.

Lemma 5. *The protector* MERGE-PROT $= \prod_{i,i' \in I, i \neq i'}$ MERGE-PROT-PAIR$_{\{i,i'\}}$ *for the* MERGE-VEHICLES *automaton guarantees* $\bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$ *in* MERGE-VEHICLES *starting from* $\bigcap_{i,i' \in I, i \neq i'} S_{\{i,i'\}}$ *given* $P_{not\text{-}overspeed}$.

Lemma 5 is shown by combining Lemma 4 and Theorem 3 and realizing that the second condition of Theorem 3 does not hold.

6.4 Example 4: Collision Avoidance on a General Graph of Tracks

In this section, we present a protector, called GRAPH-PROT, that guarantees that none of the n vehicles traveling on a directed graph of tracks comprised of Y-shaped merges and diverges collide, provided that they are all abiding by the speed limit $\dot{c}_{max}$. As in Section 6.3, the GRAPH-PROT protector is defined as the composition of $n(n-1)/2$ separate copies of another protector called GRAPH-PROT-PAIR$_{\{i,i'\}}$, one copy for each unordered pair of vehicles $\{i, i'\}$, where $i, i' \in I, i \neq i'$. Each of the GRAPH-PROT-PAIR$_{\{i,i'\}}$ protectors, for $i, i' \in I, i \neq i'$, guarantees that the vehicles i and i' do not collide into each other, provided that all the vehicles are abiding by the speed limit and the vehicles of all other vehicle pairs do not collide between themselves.

We augment the MERGE-VEHICLES automaton to involve a general track topology consisting of a directed graph G of Y-shaped merges and diverges. All the edges of the graph G are assumed to be of sufficient length to rule out collisions among vehicles that are neither on identical, nor on contiguous edges and all cycles of the graph G are assumed to have at least three edges. Moreover, in order to brake the topological symmetry in merge situations, we associate with each edge of the track topology a unique and totally ordered priority index. The resulting physical plant automaton is henceforth referred to as GRAPH-VEHICLES (Appendix H).

Letting $PP_{\{i,i'\}}$, $S_{\{i,i'\}}$, $R_{\{i,i'\}}$, $G_{\{i,i'\}}$, $j_{\{i,i'\}}$, and $d_{\{i,i'\}}$ be as defined in Section 6.3, we define the GRAPH-PROT-PAIR$_{\{i,i'\}}$ automaton to be the composition of $Sensor(PP_{\{i,i'\}}, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j_{\{i,i'\}}, d_{\{i,i'\}})$ and the discrete controller automaton of Appendix J.

The braking strategy of the GRAPH-PROT-PAIR$_{\{i,i'\}}$ protector is as follows. The protector is allowed to brake the vehicles i and i' only if the sections of the track they claim in $d_{\{i,i'\}}$ time units overlap. Given that the vehicles i and i' are indeed involved in such a claim overlap, there are two possible scenarios depending on whether the vehicles i and i' are traveling in succession, or on adjacent tracks. If the vehicles are traveling in succession, then the vehicle i is instructed to brake if it trails the vehicle i'; otherwise, the vehicle i' is instructed to brake. On the other hand, if the vehicles i and i' are traveling on adjacent edges, the vehicle i is instructed to brake either if *only* the vehicle i' owns the merge point, or if both or neither vehicles own the merge point and the vehicle i' is traveling on the edge of greater priority; otherwise, the vehicle i' is instructed to brake.

Lemma 6. *The protector* GRAPH-PROT-PAIR$_{\{i,i'\}}$ *guarantees that the* GRAPH-VEHICLES *automaton remains within* $G_{\{i,i'\}}$ *starting from* $S_{\{i,i'\}}$ *given* $R_{\{i,i'\}}$.

Lemma 7. *The protector* GRAPH-PROT $= \prod_{i,i' \in I, i \neq i'}$ GRAPH-PROT-PAIR$_{\{i,i'\}}$ *for the* GRAPH-VEHICLES *automaton guarantees* $\bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$ *in* GRAPH-VEHICLES *starting from* $\bigcap_{i,i' \in I, i \neq i'} S_{\{i,i'\}}$ *given* $P_{not\text{-}overspeed}$.

Lemma 7 is shown by combining Lemma 6 and Theorem 3 and realizing that the second condition of Theorem 3 does not hold.

6.5 Composing the Overspeed and Collision Protectors

In the previous sections, we presented example protectors whose correct operation required that the physical plant automaton at hand satisfied particular properties. For example, in the case of the VEHICLES automaton of Section 5, the overspeed protector OS-PROT of Section 6.1 assumes that none of the vehicles collide among themselves and the collision protector CL-PROT of Section 6.2 assumes that none of the vehicles exceed the speed limit. Using Theorem 3 it can be shown that the composition OS-PROT $\times$ CL-PROT is a protector for the VEHICLES automaton that guarantees that the vehicles in the VEHICLES automaton neither exceed the speed limit, nor collide among themselves. In fact, realizing that the OS-PROT protector extends, virtually unchanged, to the MERGE-VEHICLES and GRAPH-VEHICLES automata, such composition results extend to the MERGE-VEHICLES and GRAPH-VEHICLES automata by composing the OS-PROT protector with the MERGE-PROT and GRAPH-PROT protectors, respectively.

7 Conclusions

In this paper, we demonstrate how formal analysis techniques using the hybrid I/O automaton model can be applied to the specification and verification of hybrid systems whose structure adheres to the protection subsystem paradigm. We propose a parameterized abstract protector model which allows simple specification of an abstract protector for any hybrid system of this form. Such specification is obtained by defining the physical system, the start states, the sets of guarantee and reliance, and the port and sampling period with which the protector communicates with the physical plant. The proof of correctness of the abstract model leads to simple correctness proofs of the protector implementations for particular instantiations of the abstract model. Finally, the composition of independent, and even dependent protectors under mild conditions, guarantees the conjunction of the safety properties guaranteed by the individual protectors. The examples presented in this paper show that the proposed formal framework provides a precise and succinct protector specification, involves simple and straight forward proof methodology, and scales to complex hybrid systems through abstraction and modular decomposition.

Acknowledgements

We would hereby like to thank Dr. Steven L. Spielman of Raytheon Corporation and Norman M. Delisle formerly of Raytheon Corporation for helpful discussions regarding the PRT 2000™. We are grateful for having the opportunity to develop our formal modeling framework on the basis of a real application. We would also like to thank the submission's reviewers for their helpful suggestions and constructive comments.

References

1. Michael S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control.* Doctor of Science Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, June 1995.
2. Ekaterina Dolginova and Nancy A. Lynch. Safety Verification for Automated Platoon Maneuvers: A Case Study. In Oded Maler, editor, *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 154–170. Springer-Verlag, 1997. The International Workshop on Hybrid and Real-Time Systems took place in Grenoble, France, in March 1997.
3. Rainer Gawlick, Roberto Segala, Jørgen Søgaard-Andersen, and Nancy A. Lynch. Liveness in Timed and Untimed Systems. Technical Report MIT/LCS/TR-587, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, December 1993.
4. Rainer Gawlick, Roberto Segala, Jørgen Søgaard-Andersen, and Nancy A. Lynch. Liveness in Timed and Untimed Systems. In Serge Abiteboul and Eli Shamir, editors, *Proc. 21st International Colloquium on Automata, Languages and Programming (ICALP'94)*, volume 820 of *Lecture Notes in Computer Science*, pages 166–177. Springer-Verlag, 1994. The 21st International Colloquium on Automata, Languages and Programming (ICALP'94) took place in Jerusalem, Israel, in July 1994. Full version appeared as Ref. 3.
5. Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science.* Springer-Verlag, 1993. This volume of LNCS was inspired by a workshop on the Theory of Hybrid Systems, held on Oct. 19–21, 1992 at the Technical University, Lyngby, Denmark, and by a prior Hybrid Systems Workshop, held on June 10–12, 1991 at the Mathematical Sciences Institute, Cornell University.
6. Carolos Livadas. Formal Verification of Safety-Critical Hybrid Systems. Master of Engineering Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 1997.
7. John Lygeros. *Hierarchical, Hybrid Control of Large Scale Systems.* Doctor of Philosophy Thesis, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, May 1996.
8. John Lygeros, Datta N. Godbole, and Shankar Sastry. A Verified Hybrid Controller for Automated Vehicles. In *35th IEEE Conference on Decision and Control (CDC'96)*, pages 2289–2294, Kobe, Japan, December 1996.
9. John Lygeros and Nancy Lynch. On the Formal Verification of the TCAS Conflict Resolution Algorithm. In *36th IEEE Conference on Decision and Control (CDC'97)*, San Diego, CA, December 1997. To appear.
10. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O Automata. Technical Memo MIT/LCS/TM-544, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, December 1995.
11. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O Automata. In R. Alur, T. Henzinger, and E. Sontag, editors, *Proc. DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, Hybrid Systems III: Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 496–510. Springer-Verlag, 1996. The DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems took place in New Brunswick, New Jersey, in October 1995.

12. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O Automata. Preprint/Work in Progress. Preliminary versions appeared as Refs. 10 and 11, June 1997.
13. Nancy Lynch and Frits Vaandrager. Forward and Backward Simulations — Part II: Timing-Based Systems. Technical Memo MIT/LCS/TM-487.c, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, April 1995.
14. Nancy Lynch and Frits Vaandrager. Forward and Backward Simulations — Part II: Timing-Based Systems. *Information and Computation*, 128(1):1–25, July 1996. Preliminary version appeared as Ref. 13.
15. George J. Pappas, Claire Tomlin, and Shankar Sastry. Conflict Resolution in Multi-Agent Hybrid Systems. In *35th IEEE Conference on Decision and Control (CDC'96)*, Kobe, Japan, December 1996.
16. Amir Pnueli and Joseph Sifakis, editors. *Special Issue on Hybrid Systems*, volume 138, part 1 of *Theoretical Computer Science*. Elsevier Science Publishers, February 1995.
17. H. B. Weinberg, Nancy Lynch, and Norman Delisle. Verification of Automated Vehicle Protection Systems. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 101–113. Springer-Verlag, 1996.

A Derived Variables and Sets of the VEHICLES Automaton

$E_i \in \mathcal{P}(\mathbb{R})$, defined by

$$E_i = [x_i, x_i + c_{len}]$$

$collided(i, *) \in \texttt{Bool}$, for $i \in I$, defined by

$$collided(i, *) = \bigvee_{i' \in I, i' \neq i} collided(i, i')$$

$collided(*, i) \in \texttt{Bool}$, for $i \in I$, defined by

$$collided(*, i) = \bigvee_{i' \in I, i' \neq i} collided(i', i)$$

$collided(*, i, *) \in \texttt{Bool}$, for $i \in I$, defined by

$$collided(*, i, *) = collided(*, i) \vee collided(i, *)$$

$VALID \subseteq states(\text{VEHICLES})$, defined by

$VALID = \{p \in states(\text{VEHICLES}) \mid$
1. $\nexists\, i, i' \in I, i \neq i'$ such that the set $p.E_i \cap p.E_{i'}$ is a positive length closed interval of $\mathbb{R}$.
2. $p.\dot{x}_i \geq 0$, for all $i \in I$.
3. If $\neg p.collided(*, i, *)$ then $p.\ddot{x}_i \in [\ddot{c}_{min}, \ddot{c}_{max}]$, for all $i \in I$.
4. If $\neg p.collided(*, i, *) \wedge p.brake(i)$ then if $p.\dot{x}_i = 0$ then $p.\ddot{x}_i = 0$ else $p.\ddot{x}_i = \ddot{c}_{brake}$, for all $i \in I$. $\}$

$stop\text{-}dist_i \in \mathbb{R}^{\geq 0}$, for all $i \in I$, defined by

$$stop\text{-}dist_i = -\frac{\dot{x}_i^2}{2\ddot{c}_{brake}}$$

$max\text{-}range_i(t) \in \mathbb{R}^{\geq 0}$, for all $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, defined by

$$max\text{-}range_i(t) = \begin{cases} \dot{x}_i \Delta t + \frac{1}{2}\ddot{c}_{max}\Delta t^2 + \dot{c}_{max}(t - \Delta t), \\ \quad \text{where } \Delta t = \min\left(t, \frac{\dot{c}_{max} - \dot{x}_i}{\ddot{c}_{max}}\right) & \text{if } \dot{x}_i \leq \dot{c}_{max}, \text{ and} \\ \dot{x}_i \Delta t + \frac{1}{2}\ddot{c}_{brake}\Delta t^2 + \dot{c}_{max}(t - \Delta t), \\ \quad \text{where } \Delta t = \min\left(t, \frac{\dot{c}_{max} - \dot{x}_i}{\ddot{c}_{brake}}\right) & \text{otherwise.} \end{cases}$$

$max\text{-}vel_i(t) \in \mathbb{R}^{\geq 0}$, for all $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, defined by

$$max\text{-}vel_i(t) = \begin{cases} \min(\dot{c}_{max}, \dot{x}_i + t\ddot{c}_{max}) & \text{if } \dot{x}_i \leq \dot{c}_{max}, \text{ and} \\ \max(\dot{c}_{max}, \dot{x}_i + t\ddot{c}_{brake}) & \text{otherwise.} \end{cases}$$

$O_i \subseteq \mathbb{R}$, for all $i \in I$, defined by

$$O_i = [x_i, x_i + stop\text{-}dist_i + c_{len}]$$

$C_i(t) \subseteq \mathbb{R}$, for all $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, defined by

$$C_i(t) = [x_i, x_i + max\text{-}range_i(t) - max\text{-}vel_i(t)^2/(2\ddot{c}_{brake}) + c_{len}]$$

B Auxiliary Sets for the VEHICLES Automaton

$P_{overspeed(i)} \subseteq VALID$, for $i \in I$, defined by

$$P_{overspeed(i)} = \{p \in VALID \mid p.\dot{x}_i > \dot{c}_{max}\}$$

$P_{overspeed} \subseteq VALID$, defined by

$$P_{overspeed} = \bigcup_{i \in I} P_{overspeed(i)}$$

$P_{not\text{-}overspeed} \subseteq VALID$, defined by

$$P_{not\text{-}overspeed} = VALID - P_{overspeed}$$

$P_{collided(i,i')} \subseteq VALID$, for $i, i' \in I$, $i \neq i'$, defined by

$$P_{collided(i,i')} = \{p \in VALID \mid p.collided(i,i') = \texttt{True}\}$$

$P_{collided(i)} \subseteq VALID$, defined by

$$P_{collided(i)} = \bigcup_{i' \in I, i' \neq i} P_{collided(i,i')}$$

$P_{collided} \subseteq VALID$, defined by

$$P_{collided} = \bigcup_{i \in I} P_{collided(i)} = \bigcup_{i,i' \in I, i \neq i'} P_{collided(i,i')}$$

$P_{not\text{-}collided} \subseteq VALID$, defined by

$$P_{not\text{-}collided} = VALID - P_{collided}$$

$disjoint\text{-}extents(i,i') \subseteq VALID$, for $i, i' \in I, i \neq i'$, defined by

$$disjoint\text{-}extents(i,i') = \{p \in VALID \mid p.E_i \cap p.E_{i'} = \emptyset\}$$

$P_E \subseteq VALID$, defined by

$$P_E = \bigcap_{i,i' \in I, i \neq i'} disjoint\text{-}extents(i,i')$$

$disjoint\text{-}owned\text{-}tracks(i,i') \subseteq VALID$, for $i, i' \in I, i \neq i'$, defined by

$$disjoint\text{-}owned\text{-}tracks(i,i') = \{p \in VALID \mid p.O_i \cap p.O_{i'} = \emptyset\}$$

$P_O \subseteq VALID$, defined by

$$P_O = \bigcap_{i,i' \in I, i \neq i'} disjoint\text{-}owned\text{-}tracks(i,i')$$

$disjoint\text{-}claimed\text{-}tracks(i,i',t) \subseteq VALID$, for $i, i' \in I, i \neq i'$, and $t \in \mathbb{R}^{\geq 0}$, defined by

$$disjoint\text{-}claimed\text{-}tracks(i,i',t) = \{p \in VALID \mid p.C_i(t) \cap p.C_{i'}(t) = \emptyset\}$$

$P_{C(t)} \subseteq VALID$, for $t \in \mathbb{R}^{\geq 0}$, defined by

$$P_{C(t)} = \bigcap_{i,i' \in I, i \neq i'} disjoint\text{-}claimed\text{-}tracks(i,i',t)$$

$P_{B_{ij}} \subseteq VALID$, defined by

$$P_{B_{ij}} = \{p \in VALID \mid p.brake\text{-}req(i,j) = \texttt{True}\}$$

C Discrete Controller Automaton for the Protector OS-PROT-SOLO$_i$

Actions: Input: e, the environment action (stuttering)
$\texttt{snapshot}(y)_j$, for each valuation y of Y_{VEHICLES}
Output: $\texttt{brake}(i)_j$
$\texttt{unbrake}(i)_j$

Variables: Internal: $send_j \in \{\texttt{brake}, \texttt{unbrake}, null\}$, initially $null$

Discrete Transitions:

e
Eff: None

$\texttt{snapshot}(y)_j$
Eff: if $(y.\dot{x}_i \leq \dot{c}_{max} - d\ddot{c}_{max})$ then
$send_j := \texttt{unbrake}$
else
$send_j := \texttt{brake}$

$\texttt{brake}(i)_j$
Pre: $send_j = \texttt{brake}$
Eff: $send_j := null$

$\texttt{unbrake}(i)_j$
Pre: $send_j = \texttt{unbrake}$
Eff: $send_j := null$

Trajectories:
$w.send_j \equiv null$

D Discrete Controller Automaton for the Protector CL-PROT-SOLO$_i$

Actions: Input: e, the environment action (stuttering)
$\texttt{snapshot}(y)_j$, for each valuation y of Y_{VEHICLES}
Output: $\texttt{brake}(i)_j$
$\texttt{unbrake}(i)_j$

Variables: Internal: $send_j \in \{\texttt{brake}, \texttt{unbrake}, null\}$, initially $null$

Discrete Transitions:

e
Eff: None

$\texttt{snapshot}(y)_j$
Eff: if $\exists i' \in I, i' \neq i$ such that
$y \notin$ *disjoint-claimed-tracks*(i, i', d)
$\wedge (y.x_i < y.x_{i'})$
then
$send_j := \texttt{brake}$
else
$send_j := \texttt{unbrake}$

$\texttt{brake}(i)_j$
Pre: $send_j = \texttt{brake}$
Eff: $send_j := null$

$\texttt{unbrake}(i)_j$
Pre: $send_j = \texttt{unbrake}$
Eff: $send_j := null$

Trajectories:
$w.send_j \equiv null$

E The MERGE-VEHICLES Automaton

Actions:

Input:

e, the environment action (stuttering)

$\texttt{protect}(C)_j$, for all $C \in \mathcal{P}(I), j \in J$

Internal:

$\texttt{colliding-pair}(i, i')$, for all $i, i' \in I, i' \neq i$

$\texttt{collision-effects}(i)$, for all $i \in I$

$\texttt{brick-wall}(i)$, for all $i \in I$

Variables

Internal:

$\ddot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\ddot{x}_i \in \mathbb{R}$

$brake(i) \in \texttt{Bool}$, for all $i \in I$, initially $\texttt{False}$

$brake\text{-}req(i, j) \in \texttt{Bool}$, for all $i \in I, j \in J$, initially $\texttt{False}$

Output:

$l_i \in L$, for all $i \in I$, initially $l_i \in L$

$\dot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\dot{x}_i \in \mathbb{R}$

$collided(i, i') \in \texttt{Bool}$, for all $i, i' \in I, i' \neq i$, initially $\texttt{False}$

subject to *VALID*

Discrete Transitions:

e

Eff: None

$\texttt{protect}(C)_j$

Eff: for all $i \in C$

$brake\text{-}req(i, j) := \texttt{True}$

if $\neg brake(i)$ then

$brake(i) := \texttt{True}$

if $\dot{x}_i = 0$ then

$\ddot{x}_i := 0$

else

$\ddot{x}_i := \ddot{c}_{brake}$

for all $i \in I - C$

$brake\text{-}req(i, j) := \texttt{False}$

if $brake(i)$

$\wedge(\neg \vee_{k \in J}\ brake\text{-}req(i, k))$

then

$brake(i) := \texttt{False}$

$\ddot{x}_i :\in [\ddot{c}_{min}, \ddot{c}_{max}]$

$\texttt{colliding-pair}(i, i')$

Pre: $\neg collided(i, i')$

$\wedge(E_i \cap E_{i'} \neq \emptyset)$

$\wedge(l_i < \min(E_i \cap E_{i'}))$

Eff: $collided(i, i') := \texttt{True}$

if $(l_i.b \neq l_{i'}.b)$

$\wedge(l_i.b \neq \texttt{out}) \wedge (l_{i'}.b \neq \texttt{out})$

then

$collided(i', i) := \texttt{True}$

$\texttt{collision-effects}(i)$

Pre: $collided(*, i, *)$

Eff: $\dot{x}_i :\in \mathbb{R}^{\geq 0}$

$\ddot{x}_i :\in \mathbb{R}$

$\texttt{brick-wall}(i)$

Pre: $\texttt{True}$

Eff: $\dot{x}_i := 0$

if $brake(i)$ then

$\ddot{x}_i := 0$

else

$\ddot{x}_i :\in [0, \ddot{c}_{max}]$

Trajectories:

for all $i, i' \in I, i \neq i'$, *collided*(i, i') is constant throughout w
for all $i \in I$ and $j \in J$, *brake*(i) and *brake-req*(i, j) are constant throughout w
for all $i, i' \in I, i \neq i'$
the function $w.\ddot{x}_i$ is integrable
for all $t \in T_I$
$w(t).\dot{x}_i = w(0).\dot{x}_i + \int_0^t w(s).\ddot{x}_i \, ds$
$w(t).l_i.x = w(0).l_i.x + \int_0^t w(s).\dot{x}_i \, ds$
if $\neg w.\mathit{collided}(i, i')$
$\wedge (w(t).E_i \cap w(t).E_{i'} \neq \emptyset)$
$\wedge (w(t).l_i < \min(w(t).E_i \cap w(t).E_{i'}))$
then
$t = w.\mathit{ltime}$
subject to *VALID*

F Auxiliary Sets for the MERGE-VEHICLES Automaton

comparable$(i, i') \subseteq$ *VALID*, for $i, i' \in I, i \neq i'$, defined by

$$\mathit{comparable}(i, i') = \{p \in \mathit{VALID} \mid (p.l_i.b = p.l_{i'}.b) \vee (p.l_i.b = \mathtt{out}) \vee (p.l_{i'}.b = \mathtt{out})\}$$

incomparable$(i, i') \subseteq$ *VALID*, for $i, i' \in I, i \neq i'$, defined by

$$\mathit{incomparable}(i, i') = \mathit{VALID} - \mathit{comparable}(i, i')$$

yield-comparable$(i, i') \subseteq$ *comparable*(i, i'), for $i, i' \in I, i \neq i'$, defined by

$$\mathit{yield\text{-}comparable}(i, i') = \{p \in \mathit{comparable}(i, i') \mid p.l_i < p.l_{i'}\}$$

yield-incomparable$(i, i') \subseteq$ *incomparable*(i, i'), for $i, i' \in I, i \neq i'$, defined by

$$\begin{aligned}\mathit{yield\text{-}incomparable}(i, i') = \{p \in \mathit{incomparable}(i, i') \mid \\ (\langle \mathtt{out}, 0\rangle \notin p.O_i \wedge \langle \mathtt{out}, 0\rangle \in p.O_{i'}) \\ \vee (\langle \mathtt{out}, 0\rangle \in p.O_i \wedge \langle \mathtt{out}, 0\rangle \in p.O_{i'} \\ \wedge p.l_i.b = \mathtt{left}) \\ \vee (\langle \mathtt{out}, 0\rangle \notin p.O_i \wedge \langle \mathtt{out}, 0\rangle \notin p.O_{i'} \\ \wedge p.l_i.b = \mathtt{left})\}\end{aligned}$$

yield$(i, i') \subseteq$ *VALID*, for $i, i' \in I, i \neq i'$, defined by

$$\mathit{yield}(i, i') = \mathit{yield\text{-}comparable}(i, i') \cup \mathit{yield\text{-}incomparable}(i, i')$$

G Discrete Controller Automaton for the Protector MERGE-PROT-PAIR$_{\{i,i'\}}$

Actions: Input: e, the environment action (stuttering)
$\texttt{snapshot}(y)_j$, for each valuation y of $Y_{\text{MERGE-VEHICLES}}$
Output: $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(\{i, i'\})$

Variables: Internal: $send_j \in \mathcal{P}(\{i, i'\}) \cup null$, initially $null$

Discrete Transitions:

e
Eff: None

$\texttt{protect}(C)_j$
Pre: $send_j = C$
Eff: $send_j := null$

$\texttt{snapshot}(y)_j$
Eff: if $y \notin$ *disjoint-claimed-tracks*(i, i', d) then
if $y \in$ *yield*(i, i') then
$send_j := \{i\}$
else
$send_j := \{i'\}$
else
$send_j := \emptyset$

Trajectories:
$w.send_j \equiv null$

H The GRAPH-VEHICLES Automaton

Actions:

Input:

e, the environment action (stuttering)

$\texttt{protect}(C)_j$, for all $C \in \mathcal{P}(I), j \in J$

Internal:

$\texttt{colliding-pair}(i, i')$, for all $i, i' \in I, i' \neq i$

$\texttt{collision-effects}(i)$, for all $i \in I$

$\texttt{brick-wall}(i)$, for all $i \in I$

$\texttt{reset-location}(i)$, for all $i \in I$

Variables

Internal:

$\ddot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\ddot{x}_i \in \mathbb{R}$

$brake(i) \in \texttt{Bool}$, for all $i \in I$, initially $\texttt{False}$

$brake\text{-}req(i, j) \in \texttt{Bool}$, for all $i \in I, j \in J$, initially $\texttt{False}$

Output:

$l_i \in L$, for all $i \in I$, initially $l_i \in L$

$\dot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\dot{x}_i \in \mathbb{R}$

$collided(i, i') \in \texttt{Bool}$, for all $i, i' \in I, i' \neq i$, initially $\texttt{False}$

subject to *VALID*

Discrete Transitions:

e

Eff: None

$\texttt{protect}(C)_j$

Eff: for all $i \in C$

$brake\text{-}req(i, j) := \texttt{True}$

if $\neg brake(i)$ then

$brake(i) := \texttt{True}$

if $\dot{x}_i = 0$ then

$\ddot{x}_i := 0$

else

$\ddot{x}_i := \ddot{c}_{brake}$

for all $i \in I - C$

$brake\text{-}req(i, j) := \texttt{False}$

if $brake(i)$

$\wedge(\neg \vee_{k \in J}\ brake\text{-}req(i, k))$

then

$brake(i) := \texttt{False}$

$\ddot{x}_i :\in [\ddot{c}_{min}, \ddot{c}_{max}]$

$\texttt{reset-location}(i)$

Pre: $l_i.x = length(l_i.e)$

Eff: $l_i.e :\in out(l_i.e)$

$l_i.x := 0$

$\texttt{colliding-pair}(i, i')$

Pre: $\neg collided(i, i')$

$\wedge(E_i \cap E_{i'} \neq \emptyset)$

$\wedge(l_i < \min(E_i \cap E_{i'}))$

Eff: $collided(i, i') := \texttt{True}$

if $(l_i.e \neq l_{i'}.e)$

$\wedge(l_i.e.v_{final} = l_{i'}.e.v_{final})$

then

$collided(i', i) := \texttt{True}$

$\texttt{collision-effects}(i)$

Pre: $collided(*, i, *)$

Eff: $\dot{x}_i :\in \mathbb{R}^{\geq 0}$

$\ddot{x}_i :\in \mathbb{R}$

$\texttt{brick-wall}(i)$

Pre: $\texttt{True}$

Eff: $\dot{x}_i := 0$

if $brake(i)$ then

$\ddot{x}_i := 0$

else

$\ddot{x}_i :\in [0, \ddot{c}_{max}]$

Trajectories:

for all $i, i' \in I, i \neq i'$, *collided*(i, i') is constant throughout w
for all $i \in I$ and $j \in J$, *brake*(i) and *brake-req*(i, j) are constant throughout w
for all $i, i' \in I, i \neq i'$
the function $w.\ddot{x}_i$ is integrable
for all $t \in T_I$
$w(t).\dot{x}_i = w(0).\dot{x}_i + \int_0^t w(s).\ddot{x}_i \, ds$
$w(t).l_i.x = w(0).l_i.x + \int_0^t w(s).\dot{x}_i \, ds$
if $\neg w.collided(i, i')$
$\wedge (w(t).E_i \cap w(t).E_{i'} \neq \emptyset)$
$\wedge (w(t).l_i < \min(w(t).E_i \cap w(t).E_{i'}))$
then
$t = w.ltime$
if $w(t).l_i.x = length(w(t).l_i.e)$ then
$t = w.ltime$
subject to *VALID*

I Auxiliary Sets for the GRAPH-VEHICLES Automaton

successive$(i, i') \subseteq$ *VALID*, for $i, i' \in I, i \neq i'$, defined by

$$successive(i,i') = \{p \in VALID \mid (p.l_i.e = p.l_{i'}.e) \vee (p.l_i.e.v_{final} = p.l_{i'}.e.v_{init}) \vee (p.l_{i'}.e.v_{final} = p.l_i.e.v_{init})\}$$

adjacent$(i, i') \subseteq$ *VALID*, for $i, i' \in I, i \neq i'$, defined by

$$adjacent(i,i') = \{p \in VALID \mid (p.l_i.e \neq p.l_{i'}.e) \wedge (p.l_i.e.v_{final} = p.l_{i'}.e.v_{final})\}$$

yield-successive$(i, i') \subseteq$ *VALID*, for $i, i' \in I, i \neq i'$, defined by

$$yield\text{-}successive(i,i') = \{p \in successive(i,i') \mid p.l_i < p.l_{i'}\}$$

yield-adjacent$(i, i') \subseteq$ *VALID*, for $i, i' \in I, i \neq i'$, defined by

$$\begin{aligned} yield\text{-}adjacent(i,i') = \{p \in adjacent(i,i') \mid\ & (\langle p.l_i.e, length(p.l_i.e)\rangle \notin p.O_i \\ & \wedge \langle p.l_{i'}.e, length(p.l_{i'}.e)\rangle \in p.O_{i'}) \\ & \vee (\langle p.l_i.e, length(p.l_i.e)\rangle \in p.O_i \\ & \quad \wedge \langle p.l_{i'}.e, length(p.l_{i'}.e)\rangle \in p.O_{i'} \\ & \quad \wedge priority(p.l_i.e) < priority(p.l_{i'}.e)) \\ & \vee (\langle p.l_i.e, length(p.l_i.e)\rangle \notin p.O_i \\ & \quad \wedge \langle p.l_{i'}.e, length(p.l_{i'}.e)\rangle \notin p.O_{i'} \\ & \quad \wedge priority(p.l_i.e) < priority(p.l_{i'}.e))\} \end{aligned}$$

yield$(i, i') \subseteq$ *VALID*, for $i, i' \in I, i \neq i'$, defined by

$$yield(i,i') = yield\text{-}successive(i,i') \cup yield\text{-}adjacent(i,i')$$

J Discrete controller automaton for the protector GRAPH-PROT-PAIR$_{\{i,i'\}}$

Actions: Input: e, the environment action (stuttering)
$\texttt{snapshot}(y)_j$, for each valuation y of $Y_{\text{GRAPH-VEHICLES}}$
Output: $\texttt{protect}(C)_j$, for $C \in \mathcal{P}(\{i, i'\})$
Variables: Internal: $send_j \in \mathcal{P}(\{i, i'\}) \cup null$, initially $null$

Discrete Transitions:

e
Eff: None

$\texttt{snapshot}(y)_j$
Eff: if $y \notin$ *disjoint-claimed-tracks*(i, i', d) then
if $y \in$ *yield*(i, i') then
$send_j := \{i\}$
else
$send_j := \{i'\}$
else
$send_j := \emptyset$

$\texttt{protect}(C)_j$
Pre: $send_j = C$
Eff: $send_j := null$

Trajectories:
$w.send_j \equiv null$

AMAST '96

A Three-Level Analysis of a Simple Acceleration Maneuver, with Uncertainties (preliminary version)*

Nancy Lynch

February 15, 1996

1 Introduction

In this note, we give a three-level analysis of a toy vehicle acceleration maneuver. The goal of the maneuver is to cause a vehicle, starting at velocity 0 at time 0, to attain a velocity of b (or as close to b as possible) at a later time a. The vehicle is assumed to provide accurate sampled data every d time units. The vehicle is assumed to be capable of receiving control signals, one immediately after each vehicle data output. Each control signal can set an "acceleration variable", acc, to an arbitrary real number. However, the actual acceleration exhibited by the vehicle need not be exactly equal to acc – instead, we assume that it is defined by an integrable function whose values are always in the range $[acc - \epsilon, acc]$.[1] We can think of this uncertainty as representing, say, uncertainty in the performance of the vehicle's propulsion system.

The vehicle interacts with a controller, presumably a computer. In this note, we describe a particular controller and analyze the behavior of the combination of the vehicle and controller. One conclusion we draw is that the velocity of the vehicle at time a is in the range $[b - \epsilon d, b]$. That is, the uncertainty in setting acc combines multiplicatively with the sampling period to yield the uncertainty in the final velocity of the vehicle. More strongly, we obtain a range for the velocity of the vehicle at each time in the interval $[0, a]$.

We prove this fact using invariants and levels of abstraction (in particular, simulation methods), based on a new hybrid I/O automaton model of Lynch, Segala, Vaandrager and Weinberg [1]. Invariants and levels of abstraction are standard methods used in computer science for reasoning about discrete systems. Many of the pieces of the proofs use standard continuous methods, such as solving algebraic and differential equations. The entire proof represents a smooth combination of discrete and continuous methods.

The point of this exercise is to demonstrate some simple uses of levels of abstraction in reasoning about hybrid control problems. We use levels of abstraction here for two purposes: (a) to express the relationship between a derivative-based description of a system and an explicit description, and (b) to express the relationship between a system in which corrections are made at discrete sampling

*This work was supported by ARPA contracts N00014-92-J-4033 and F19628-95-C-0118, AFOSR-ONR contract F49620-94-1-0199, NSF grant 9225124-CCR and DOT contract DTRS95G-0001.

[1] We could also have included some uncertainty in the upper bound, but that would not add any interesting features to the example.

points and a system in which corrections are made continuously. The uncertainty in the acceleration is treated at all three levels of our example, and is integrated throughout the presentation.

We do not contribute anything new in the way of techniques for continuous mathematics; for example, we use standard methods of solving differential equations. Our contributions lie, rather, in the smooth combination of discrete and continuous methods within a single mathematical framework, and in the application of standard methods of discrete analysis (in particular, invariants and levels of abstraction) to hybrid systems. Our methods are particularly good at handling uncertainties and other forms of system nondeterminism.

2 Hybrid Input/Output Automata

We use the Lynch-Segala-Vaandrager-Weinberg hybrid input/output automaton (HIOA) model [1], and refer the reader to [1] for the details. We give a rough summary here.

A *hybrid I/O automaton* (HIOA) is a state machine having a (not necessarily finite) set of *states* with a subset distinguished as the *start states*, a set of *discrete actions* partitioned into *input*, *output* and *internal actions*, and a set of *variables*, similarly partitioned into *input*, *output* and *internal variables*. The states are simply combinations of values for the variables. An HIOA also has a set of *discrete steps*, which are state transitions labelled by discrete actions, plus a set of *trajectories*, which are mappings from a left-closed interval of $\mathsf{R}^{\geq 0}$ with left endpoint 0 to states. A trajectory shows how the state evolves during an interval of time. An HIOA must satisfy a collection of axioms describing restrictions on the behavior of input actions and variables, closure properties of trajectories, etc.

The operation of an HIOA is described by *hybrid execution fragments*, each of which is a finite or infinite alternating sequence, $\alpha = w_0\pi_1 w_1\pi_2 w_2 \cdots$, of trajectories and discrete actions, where successive states match up properly. A *hybrid execution* is a hybrid execution fragment that begins with a start state. A state is defined to be *reachable* if it is the final state of some finite hybrid execution.

The externally-visible behavior of an HIOA is defined using the notion of a *hybrid trace*. The *hybrid trace* of any hybrid execution fragment α is obtained from α by projecting all trajectories onto external (input and output) variables, removing all internal actions, concatenating all consecutive trajectories for which states match up properly, and inserting a special placeholder symbol τ between consecutive trajectories for which states do not match up.

The levels of abstraction that we referred to in the introduction are captured by means of mappings called *simulations*. A *simulation* from HIOA A to HIOA B with the same external actions and the same external variables is a relation R from $states(A)$ to $states(B)$ satisfying the following conditions:

1. For every start state of A, there is an R-related start state of B.

2. For every discrete step (s_A, π, s'_A) of A with s_A a reachable state, and every reachable state s_B of B that is R-related to s_A, there is a finite hybrid execution fragment of B that starts with s_B, ends with some s'_B that is R-related to s'_A, and has the same hybrid trace as the given step.

3. For every right-closed trajectory w_A of A starting with a reachable state, and every reachable state s_B that is R-related to the first state of w_A, there is a finite hybrid execution fragment of B that starts with s_B, ends with some s'_B that is R-related to the last state of w_A, and has the same hybrid trace as w_A.

The important fact about a simulation is:

Theorem 2.1 *If there is a simulation from A to B, and if α_A is any hybrid execution of A, then there is a hybrid execution α_B of B having the same hybrid trace.*

There is a notion of *composition* of HIOA's, based on identifying actions with the same name and variables with the same name in different automata. There are also *hiding* operations on HIOA's, which simply reclassify some output actions or output variables as internal. All definitions and results are given in [1].

3 Mathematical Preliminaries

3.1 Assumptions About the Constants

In the informal description in the introduction, we mentioned several constants: a, b, d and ϵ. All are assumed to be positive real-valued. We assume only that d divides evenly into a.

3.2 Some Useful Functions

3.2.1 Function f

The following function $f : [0, a] \rightarrow \mathsf{R}$ will be used in the analysis:

$$f(t) = \begin{cases} \frac{bt}{a} + \epsilon(a-t)\log(\frac{a-t}{a}) & \text{if } t \in [0, a), \\ b & \text{if } t = a. \end{cases}$$

In particular, $f(0) = 0$ and $f(a) = b$. Function f is continuous over $[0, a]$, since $\lim_{t \rightarrow a} f(t) = f(a)$. Function f satisfies:

$$\dot{f}(t) = \frac{b}{a} - \epsilon\log(\frac{a-t}{a}) - \epsilon = \frac{b - f(t)}{a-t} - \epsilon,$$

for all $t \in (0, a)$. Moreover, f has a right derivative of $\frac{b}{a} - \epsilon$ at 0, while at a, f's left derivative is undefined. (It approaches $+\infty$.)

Function f describes the behavior of a continuous process that starts at time 0 at value 0, always "tries to" set its derivative so as to point to the graph point (a, b), but consistently "misses low" by exactly ϵ. That is, f is a solution to the differential equation

$$\dot{f}(t) = \frac{b - f(t)}{a-t} - \epsilon,$$

where $t \in [0, a)$, with the boundary condition $f(0) = 0$. Function f is depicted in Figure 1.

3.2.2 Function g

We also define the function $g : [0, a] \rightarrow [0, b]$, by

$$g(t) = \frac{bt}{a}.$$

Then $g(0) = 0$ and $g(a) = b$, and g is continuous over $[0, a]$. Function g satisfies:

$$\dot{g}(t) = \frac{b}{a} = \frac{b - g(t)}{a-t}$$

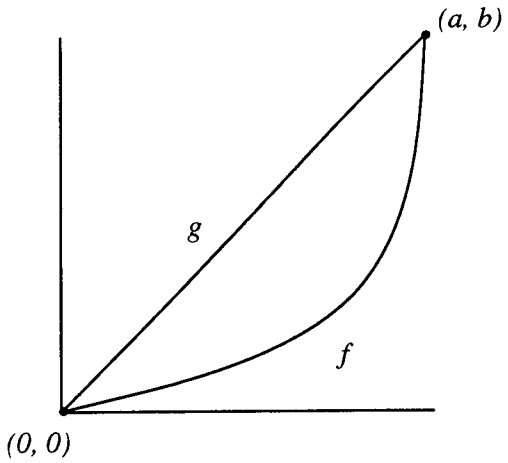


Figure 1: Functions f and g.

for all $t \in (0, a)$. Moreover, g has a right derivative of $\frac{b}{a}$ at 0 and a left derivative, also of $\frac{b}{a}$ at a. Function g is a solution to the differential equation

$$\dot{g}(t) = \frac{b - g(t)}{a - t},$$

where $t \in [0, a)$, with the boundary condition $g(0) = 0$. Function g is also depicted in Figure 1.

3.2.3 Function f_1

The following function $f_1 : [0, a] \to \mathsf{R}$ is like f, but it uses the goal of $(a, b - \epsilon d)$ instead of (a, b).

$$f_1(t) = \begin{cases} \frac{(b-\epsilon d)t}{a} + \epsilon(a - t)\log(\frac{a-t}{a}) & \text{if } t \in [0, a), \\ b - \epsilon d & \text{if } t = a. \end{cases}$$

In particular, $f_1(0) = 0$ and $f_1(a) = b - \epsilon d$. Function f_1 is continuous over $[0, a]$. Function f_1 satisfies:

$$\dot{f}_1(t) = \frac{b - \epsilon d}{a} - \epsilon \log(\frac{a - t}{a}) - \epsilon = \frac{b - \epsilon d - f_1(t)}{a - t} - \epsilon,$$

for all $t \in (0, a)$. Moreover, f_1 has a right derivative of $\frac{b-\epsilon d}{a} - \epsilon$ at 0, while at a, f_1's left derivative is undefined. (It approaches $+\infty$.) Function f_1 is a solution to the differential equation

$$\dot{f}(t) = \frac{b - \epsilon d - f(t)}{a - t} - \epsilon,$$

where $t \in [0, a)$, with the boundary condition $f(0) = 0$. The function f_1 is depicted in Figure 2.

3.2.4 Function h

Finally, we consider the function $h : [0, a] \to [0, b - \epsilon a]$, where

$$h(t) = \frac{bt}{a} - \epsilon t.$$

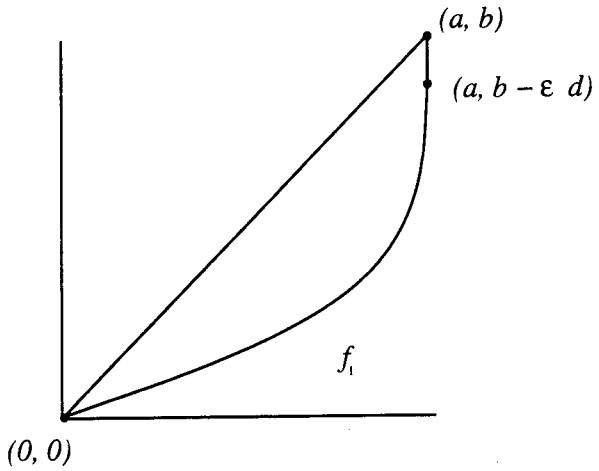


Figure 2: Function f_1.

In particular, $h(0) = 0$ and $h(a) = b - \epsilon a$. Also,

$$\dot{h}(t) = \frac{b}{a} - \epsilon$$

for all $t \in (0, a)$, and the half derivatives at the endpoints are also equal to $\frac{b}{a} - \epsilon$. Function h satisfies:

$$\dot{h}(t) \leq \frac{b - h(t)}{a - t} - \epsilon$$

for all $t \in [0, a)$.

4 High Level Specification V

We begin with a high-level system specification. This will not be our final version of the high-level specification – this preliminary version includes only the effects of the uncertainty in the acceleration, but not the effects of sampling delays. We add those later, in V_1, in Section 6.1.

4.1 Overview

Our highest-level system description consists of constraints on the vehicle velocity, embodied in an HIOA V. V simply constrains the vehicle velocity v to be anywhere within a given region bounded by the continuous functions f and g. This region is represented by the area under the line and over the curve in Figure 1 above. Note that this region is determined by the parameters a, b and ϵ; in particular, it depends on the uncertainty of acceleration.

We imagine that this region delineates the "acceptable" vehicle velocities at various times. These limitations on velocities might be used to prove some properties of a system containing the vehicle. This description places no limitations on, say, vehicle acceleration; for example, it permits the vehicle to accelerate arbitrarily quickly, as long as the velocity remains within the given region.[2]

[2]Of course, in a practical context, there might also be limitations on acceleration, imposed, for example, by passenger comfort requirements or physical laws. In such a case, the high-level specification would be different from what we give here, including restrictions on acceleration as well as velocity.

We think that it is reasonable to use such region descriptions to express system requirements. It might not matter *how* a system ensures that the controlled entity remains within the required region – just the region restriction itself might be enough to ensure that the system behaves as required. For example, an air traffic control system might operate by allocating regions in space-time to airplanes. As long as the allocated regions are disjoint, planes can fly without danger of collision. It should not matter how the system ensures that the planes remain within their regions.

In Section 5, we will give a lower-level description of the system, in terms of $\dot{v}$, the derivative of the velocity. We think of the derivative-based description as a way of implementing the region description.

4.2 Formal Description

We define a single HIOA V. Automaton V has no discrete actions (except for a dummy *environment action* e required as a technicality by the formal model). It has the following variables:

Input:
none
Internal:
none
Output:
$now \in [0, a]$, initially 0
$v \in \mathsf{R}$, initially 0

The only discrete steps are dummy e-steps that cause no state change. The trajectories of V are all the mappings w from left-closed subintervals I of $[0, a]$ to states of V such that:

1. For all $t \in I$, the following conditions hold in state $w(t)$.
   (a) $now = w(0).now + t$.
   (b) $v \in [f(now), g(now)]$.

Condition (a) says that the value of *now* just increases along with the real time – the difference is that t is a relative time measure, which starts at 0 in each trajectory, while *now* is an absolute time measure, which starts at 0 at the beginning of an entire hybrid execution. Condition (b) describes the envelope for v. The *now* component allows us to express the second condition just in terms of the automaton state, a useful style for invariant and simulation proofs.

We do not require any other assumptions. For instance, continuity of v is not required at this level, although it will be guaranteed by any real implementation. Our assumption is that the continuity condition for v will not affect uses of this specification, but is only relevant in reasoning about implementations.

Note that our description at this level does not involve any controller. At the highest level, it is probably appropriate to consider just the behavior of the controlled system, regarding the controller as a part of the implementation.

In general, we follow the philosophy of using the maximum possible nondeterminism in our specifications – in particular, we do not include assumptions such as continuity, bounds on acceleration, etc., until we need them in the proof of some result.

We give a trivial invariant of V:

Lemma 4.1 *In every reachable state of V, the following is true.*

1. $v \in [f(now), g(now)]$.

Proof: The proof (as usual for invariants of HIOA's) is by induction on the length, that is, the number of trajectories and discrete steps, in a finite hybrid execution that leads to the state in question. Here (as usual for such proofs), we must show three things: that the property is true

in every initial state, that it is preserved by every discrete step, and that it is preserved by every right-closed trajectory. (Note that we need only consider right-closed trajectories, and that we need only show that the property holds in the last state of the trajectory, given that it holds in the first state. We do not need to show anything about the intermediate states in the trajectory.)

In this case, all of these are easy to see. In the unique start state of V, we have $v = 0$, $now = 0$, and $f(0) = g(0) = 0$, so that $v \in [0, 0]$, which is what is needed. The only discrete steps are the dummy e-steps, which obviously preserve this property. And trajectories are defined explicitly so as to preserve this property. ■

Note that Lemma 4.1 implies that, in every reachable state of V in which $now = a$, it must be that $v = b$.

5 Derivative Automaton D

In Section 4, we gave a high-level specification HIOA V, describing a region that contains the allowed values of the velocity v at all times. Now we give a lower-level description in terms of constraints on the derivative of the velocity, $\dot{v}$; this description is given as another HIOA D. Again, there is no controller.

After defining D, we prove some basic properties of its behavior, and then show that D implements V, in the sense of hybrid trace inclusion. Finally, we give an example to show how similar results could be proved for cases where the differential equations do not have known solutions.

5.1 Formal Description

HIOA D includes a variable acc, which is assumed to always "point to" to the goal point (a, b). For this section, we include no uncertainty in the value of acc – we assume that it is set completely accurately. However, there is uncertainty in the actual acceleration $\dot{v}$ – we assume that the value of $\dot{v}$ is in the interval $[acc - \epsilon, acc]$. The actual velocity v is derived from the actual acceleration $\dot{v}$ using integration.

Formally, HIOA D has a single discrete action (besides the dummy environment action e) – an internal *reset* action that simple resets $\dot{v}$ arbitrarily, as long as it preserves the required relationship between $\dot{v}$ and acc.

The discrete actions of D are:

Input:
none
Internal:
reset
Output:
none

D has the following variables:

Input:
none
Output:
$now \in [0, a]$, initially 0
$v \in \mathsf{R}$, initially 0
Internal:
$acc \in \mathsf{R}$, initially $\frac{b}{a}$
$\dot{v} \in \mathsf{R}$, initially any value in $[\frac{b}{a} - \epsilon, \frac{b}{a}]$

The e steps cause no state change, while the non-e discrete steps are all of the form:

reset
Precondition:
true
Effect:
$\dot{v} :=$ any value in $[acc - \epsilon, acc]$

The trajectories of D are all the mappings w from left-closed subintervals I of $[0, a]$ to states of D such that:

1. $\dot{v}$ is an integrable function in w.[3]
2. For all $t \in I$, the following conditions hold in state $w(t)$.
   (a) $now = w(0).now + t$.
   (b) If $now \neq a$ then $acc = \frac{b-v}{a-now}$. (Otherwise, acc is arbitrary).
   (c) If $now \neq a$ then $\dot{v} \in [acc - \epsilon, acc]$.
   (d) $v = w(0).v + \int_0^t w(x).\dot{v}dx$.

In D, acc points directly at the "goal" (a, b), but $\dot{v}$ reflects an uncertainty of ϵ. The quantity v is simply derived from $\dot{v}$, using integration.

5.2 Some Properties of D

Lemma 5.1 *Let w be any trajectory of D. Then v is a continuous function in w.*[4]

There are some obvious invariants.

Lemma 5.2 *In every reachable state of D, the following are true.*

1. *If $now \neq a$ then $acc = \frac{b-v}{a-now}$.*
2. *If $now \neq a$ then $\dot{v} \in [acc - \epsilon, acc]$.*

Proof: These follow easily from the definition of D. ■

The following invariant is a little less obvious, but is an easy consequence of Lemma 5.2. The functions f and g used in this lemma are as defined in Section 3.2.

Lemma 5.3 *In every reachable state of D in which $now \neq a$, the following are true.*

1. $\frac{v-f(now)}{a-now} \geq \dot{f}(now) - \dot{v}$.
2. $\frac{g(now)-v}{a-now} \geq \dot{v} - \dot{g}(now)$.

Proof:

1. By definition of f, we have that:

$$\dot{f}(now) = \frac{b - f(now)}{a - now} - \epsilon.$$

[3] More precisely, this means that $w(t).\dot{v}$ is an integrable function of t, where t ranges over the interval I.
[4] This means continuous in the time argument of w.

By Lemma 5.2, we have that:

$$\dot{v} \geq acc - \epsilon = \frac{b - v}{a - now} - \epsilon.$$

Therefore,

$$\dot{f}(now) - \dot{v} \leq \frac{b - f(now)}{a - now} - \epsilon - (\frac{b - v}{a - now} - \epsilon) = \frac{v - f(now)}{a - now}.$$

This is as needed.

2. By definition of g, we have that:

$$\dot{g}(now) = \frac{b - g(now)}{a - now}.$$

By Lemma 5.2, we have that:

$$\dot{v} \leq acc = \frac{b - v}{a - now}.$$

Therefore,

$$\dot{v} - \dot{g}(now) \leq \frac{b - v}{a - now} - \frac{b - g(now)}{a - now} = \frac{g(now) - v}{a - now}.$$

This is as needed.

■

Also, there are limitations on the rate of change of the velocity in D (for contrast, recall there were no such limitations in V).

Lemma 5.4 *Let w be any (right-closed or right-open) trajectory of D whose now values do not include a, and that starts from a reachable state of D. Then:*

1. *The ratio $\frac{v - f(now)}{a - now}$ is monotone nondecreasing in w.*[5]

2. *The ratio $\frac{g(now) - v}{a - now}$ is monotone nondecreasing in w.*

This says that v cannot increase too slowly – its distance from f, weighted by the time remaining, cannot decrease. Likewise, v cannot increase too fast – its distance from g, weighted by the time remaining, cannot decrease.

Proof: In each case, it suffices to show that the first derivative of the ratio is always nonnegative.

1. The first derivative of the ratio is:

$$\frac{(a - now)(\dot{v} - \dot{f}(now)) - (v - f(now))(-1)}{(a - now)^2}$$

$$= \frac{(a - now)(\dot{v} - \dot{f}(now)) + (v - f(now))}{(a - now)^2}.$$

(Here we are using the fact that $\dot{v}$ is the derivative of v – this is justified formally by the integral definition of the variable v.)

[5]This means monotone nondecreasing in the time argument of w.

Since the denominator is always positive, it suffices to show that:

$$= (a - now)(\dot{v} - \dot{f}(now)) + (v - f(now)) \geq 0$$

in all states of w. This is equivalent to saying that:

$$\frac{v - f(now)}{a - now} \geq \dot{f}(now) - \dot{v},$$

in all states of w. But this follows immediately from Lemma 5.3 (using the fact that w starts in a reachable state of D, so all its states are reachable).

2. The derivative of the ratio is:

$$\frac{(a - now)(\dot{g}(now) - \dot{v}) - (g(now) - v)(-1)}{(a - now)^2}$$

$$= \frac{(a - now)(\dot{g}(now) - \dot{v}) + (g(now) - v)}{(a - now)^2}.$$

Since the denominator is always positive, it suffices to show that:

$$(a - now)(\dot{g}(now) - \dot{v}) + (g(now) - v) \geq 0$$

in all states of w. But this is equivalent to saying that:

$$\frac{g(now) - v}{a - now} \geq \dot{v} - \dot{g}(now)$$

in all states of w. This follows from Lemma 5.3.

■

5.3 D Implements V

The main result that we want to show about D is the following:

Theorem 5.5 *If α_D is a hybrid execution of D, then there is a hybrid execution α_V of V having the same hybrid trace.*

Note that the hybrid trace of each of V and D includes just the *now* and v values. Theorem 5.5 implies that the changes in *now* and v that are exhibited by D are allowed, according to the constraints expressed by V. The correspondence does not mention the implementation variables *acc* and $\dot{v}$

We prove Theorem 5.5 using a simulation, as defined informally in Section 2. We define a relation *fsim* from states of D to states of V as follows. If s_D is a state of D and s_V is a state of V, then we say that $(s_D, s_V) \in$ *fsim* provided that the following hold.

1. $s_D.now = s_V.now$.
2. $s_D.v = s_V.v$.

We show:

Lemma 5.6 *fsim is a simulation from D to V.*

Proof: We show the three conditions in the definition of a simulation. The start condition is straightforward: If s_D is any start state of D and s_V is the unique start state of V, then both states have $now = 0$ and $v = 0$. It follows that $(s_D, s_V) \in fsim$.

Next, we consider discrete steps. Suppose that (s_D, π, s'_D) is any discrete step of D, and that $(s_D, s_V) \in fsim$. Then let the hybrid execution fragment corresponding to this step consist of the trivial trajectory containing exactly one state and no steps. Then both the discrete step and the corresponding fragment have the same hybrid trace, consisting of the values of now and v that appear in s_D. It suffices to show that $(s'_D, s_V) \in fsim$. But this is immediate, because π (a *reset* or e action) does not modify either now or v.

Now we consider trajectories. Suppose that w_D is an I-trajectory w_D of D, where I is right-closed, and suppose that the first state, s_D, of w_D is reachable in D. Suppose that s_V is a reachable state of V such that $(s_D, s_V) \in fsim$. Then let the corresponding hybrid execution fragment of V consists of a single trajectory w_V, where $w_V(t).now = w_D(t).now$ and $w_V(t).v = w_D(t).v$ for all t in the domain of I. It is obvious that the two trajectories have the same hybrid trace. The only interesting thing to show is that w_V is in fact a trajectory of V. By the definition of a trajectory of V, what we must show is that

1. For all $t \in I$, the following conditions hold in state $w(t)$.
   (a) $now = w(0).now + t$.
   (b) $v \in [f(now), g(now)]$.

(We must verify these conditions throughout the trajectory, not just at the beginning and end.) The first condition follows immediately from the same condition for w_D and the definition of w_V in terms of w_D. The second condition has two parts, a lower bound and an upper bound.

For the lower bound, since s_V is a reachable state of V, Lemma 4.1 implies that, in s_V, $v \geq f(now)$. By Lemma 5.4 and the definition of w_V in terms of w_D, we know that the ratio $\frac{v - f(now)}{a - now}$ is monotone nondecreasing in w_V, except possibly at the right endpoint of w_V if $now = a$ there. It follows that $v \geq f(now)$ throughout w_V, except possibly at the right endpoint in case $now = a$ there. But since $f(now)$ and v are continuous functions of the time argument of w_V, this inequality must hold at the right endpoint as well.

The upper bound argument is analogous. Since s_V is reachable, Lemma 4.1 implies that, in s_V, $v \leq g(now)$. By Lemma 5.4 and the definition of w_V in terms of w_D, we know that the ratio $\frac{g(now) - v}{a - now}$ is monotone nondecreasing in w_V, except possibly at the right endpoint of w_V if $now = a$ there. It follows that $v \leq g(now)$ throughout w_V, exept possibly at the right endpoint in case $now = a$ there. But since $g(now)$ and v are continuous functions of the time argument of w_V, this inequality must hold at the right endpoint as well. ■

Proof: (of Theorem 5.5)

By Lemma 5.6 and Theorem 2.1. ■

Note that the correspondence between D and V is only one-way. It says, roughly speaking, that everything that D does is allowed by V. It does not say that D has to exhibit *all* the possibilities that are allowed by V. For example, extremely fast increases in v that cannot be achieved by accelerations in the allowed ranges, but that keep v within the allowed envelope, are permitted by V, but do not actually occur in D. Also, note that D performs some activities – here, changes to acc and $\dot{v}$ – that are not explicitly represented in V.

Although Theorem 5.5 is very simple, it does demonstrate, at least in a small way, how one can carry out a correctness proof using invariants and simulations, integrating discrete and continuous reasoning, and coping with some uncertainty.

5.4 An Approximate Result

The lower bound function f is essentially defined as the solution of a differential equation that is extracted from the definition of the trajectories of D. In this case, the differential equation is easy to solve. But what if it were not so easy? In this case, the same methods could still be used, but now the lower bound produced might be a loose bound rather than an exact bound.

For example, suppose that instead of trying to prove a lower bound of f, we only tried to prove a lower bound of h, where h is the function defined in Section 3.2.4. Showing that h is a lower bound essentially requires redefining V to use h instead of f. Proving the simulation now rests on the fact, stated in Section 3.2.4, that

$$\dot{h}(t) \leq \frac{b - h(t)}{a - t} - \epsilon$$

for all $t \in [0, a)$. Using this fact, it is easy to obtain the analog to part 1 of Lemma 5.3 for h: that in every reachable state of D,

$$\frac{v - h(now)}{a - now} \geq \dot{h}(now) - \dot{v}.$$

This fact follows as in the proof of part 1 of Lemma 5.3 (but using the inequality above at one step instead of an equality as before). Next, we can prove the analog to part 1 of Lemma 5.4 for h: that the ratio $\frac{v-h(now)}{a-now}$ is monotone nondecreasing in w. This is what is needed to complete the analog to the proof of Lemma 5.6.

6 Modifications to V and D to Incorporate Periodic Feedback

The discussions and results in Sections 4 and 5 have dealt with hypothetical systems with continuous control. But recall from the introduction that in the actual implementation in which we are interested, the sampling outputs and control signals are not continuous but periodic, at intervals of d. It turns out that the abstract automata D and V do not quite provide accurate models of the actual implementation. However, they can be modified easily so that they do.

We believe that providing accurate models for the handling of uncertainties is important. It is not sufficient to give a careful analysis of a situation without uncertainty, then argue informally about the variations in behavior that are introduced by uncertainties. Handling uncertainties correctly requires considering them appropriately at all levels of abstraction.

6.1 Modified High Level Specification V_1

First, we modify V only a tiny bit to get V_1, by changing the lower bound f to the function f_1 defined in Section 3.2.3. The upper bound g remains the same as before. (Of course, we could have written the original V with parameters, so that the modifications in this section would just amount to different parameter settings.)

This modification makes the region of allowable values for v bigger by making the lower bound function smaller. The particular way that we make it smaller amount to simply replacing the "goal" of (a, b) in V with the goal of $(a, b - \epsilon d)$ in V_1, for the lower bound function only. Thus, the value of v at time a will be in the range $[b - \epsilon d, b]$, instead of always being exactly b.

It was not obvious to us at first that the high-level effect of the sampling delays is just this simple change of goal point; we discovered this only through detailed analysis of the behavior of the discrete-sampling system. We do not expect to use any general rule for determining the high-level effect of uncertainties; indeed, we expect that this will usually require some serious work,

using results of robust control theory. It is important that the high-level effect of uncertainties be described completely accurately, though the bounds need not be as tight as possible.

6.2 Modified Derivative Automaton D_1

Now we modify D to get D_1, again by modifying the lower bound requirement. Here we do this by introducing uncertainty into *acc*, allowing it to "point" anywhere from (a, b) (where it points in D) to $(a, b - \epsilon d)$. We still have the same uncertainty as before in $\dot{v}$. Thus, D_1 expresses two different types of uncertainty. We can think of the uncertainty in $\dot{v}$ as representing propulsion system uncertainty and the uncertainty in *acc* as encompassing the sampling delays.

The modifications are as follows. The states and start states of D_1 are the same as those of D, except for the following changes: The initial value of *acc* is any value in the interval $[\frac{b-\epsilon d}{a}, \frac{b}{a}]$, and the initial value of $\dot{v}$ is any value in the interval $[acc - \epsilon, acc]$. The *reset* action now changes slightly, to allow changes in *acc* as well as $\dot{v}$. These changes keep *acc* and $\dot{v}$ within the desired ranges.

reset
  Precondition:
    true
  Effect:
    $acc :=$ any value $\in [\frac{b-\epsilon d-v}{a-now}, \frac{b-v}{a-now}]$
    $\dot{v} :=$ any value $\in [acc - \epsilon, acc]$

The trajectories of D_1 are all the mappings w from left-closed subintervals I of $[0, a]$ to states of D_1 such that:

1. $\dot{v}$ is an integrable function in w.
2. For all $t \in I$, the following conditions hold in state $w(t)$.
   (a) $now = w(0).now + t$.
   (b) If $now \neq a$ then $acc \in [\frac{b-\epsilon d-v}{a-now}, \frac{b-v}{a-now}]$.
   (c) If $now \neq a$ then $\dot{v} \in [acc - \epsilon, acc]$.
   (d) $v = w(0).v + \int_0^t w(x).\dot{v}dx$.

(Again, we could have written the original D with parameters, so that the modifications in this section would amount to different parameter setting.)

6.3 Modified Correctness Proof

Our claim now is that the arguments that worked to show that D implements V can be modified slightly (and systematically) to show that D_1 implements V_1. We give the modified result statements.

Lemma 6.1 *Let w be any trajectory of D_1. Then v is a continuous function in w.*

Lemma 6.2 *In every reachable state of D_1, the following are true.*

1. *If $now \neq a$ then $acc \in [\frac{b-\epsilon d-v}{a-now}, \frac{b-v}{a-now}]$.*
2. *If $now \neq a$ then $\dot{v} \in [acc - \epsilon, acc]$.*

Lemma 6.3 *In every reachable state of D_1 in which $now \neq a$, the following are true.*

1. $\frac{v-f_1(now)}{a-now} \geq \dot{f}_1(now) - \dot{v}$.

2. $\frac{g(now)-v}{a-now} \geq \dot{v} - \dot{g}(now)$.

Proof: We only prove part 1; part 2 is unchanged from the corresponding proof for D. By definition of f_1, we have that:

$$\dot{f}_1(now) = \frac{b - \epsilon d - f_1(now)}{a - now} - \epsilon.$$

By Lemma 6.2, we have that:

$$\dot{v} \geq acc - \epsilon \geq \frac{b - \epsilon d - v}{a - now} - \epsilon.$$

Therefore,

$$\dot{f}_1(now) - \dot{v} \leq \frac{b - \epsilon d - f_1(now)}{a - now} - \epsilon - (\frac{b - \epsilon d - v}{a - now} - \epsilon) = \frac{v - f_1(now)}{a - now}.$$

This is as needed. ■

Lemma 6.4 *Let w be any trajectory of D_1 whose now values do not include a, and that starts from a reachable state of D_1. Then:*

1. *The ratio $\frac{v-f_1(now)}{a-now}$ is monotone nondecreasing in w.*

2. *The ratio $\frac{g(now)-v}{a-now}$ is monotone nondecreasing in w.*

Now define the relation $fsim_1$ from states of D_1 to states of V_1 as follows. If s_{D_1} is a state of D_1 and s_{V_1} is a state of V_1, then we say that $(s_{D_1}, s_{V_1}) \in fsim_1$ provided that the following hold.

1. $s_{D_1}.now = s_{V_1}.now$.
2. $s_{D_1}.v = s_{V_1}.v$.

This definition is essentially the same as that for $fsim$, from D to V.

Lemma 6.5 *$fsim_1$ is a simulation from D_1 to V_1.*

Proof: Similar to the proof of Lemma 5.6. ■

Theorem 6.6 *If α_{D_1} is a hybrid execution of D_1, then there is a hybrid execution α_{V_1} of V_1 having the same hybrid trace.*

Theorem 6.6 says that the changes in *now* and v that are exhibited by D_1 are allowed by V_1.

Note that the modifications we did to include this uncertainty are quite simple and systematic. A good general strategy for constructing proofs for implementations involving uncertainty is to first carry out the development without the uncertainty, then try to incorporate the uncertainty later, by making simple modifications throughout.

7 The Implementation *Impl*

Now we are (finally) ready to describe the actual implementation in which we are interested. This one consists of two components, a *Vehicle* and a *Controller*, interacting by discrete actions. Each component is, formally, an HIOA, and the combination is a composition of HIOA's, interacting via discrete actions only, with the common actions hidden.

7.1 *Vehicle*

The *Vehicle* HIOA represents the motion of the vehicle, including its velocity and acceleration. It reports the velocity (accurately, we assume) every d units of time, starting at time d. It is capable of receiving control signals that set an *acc* variable, representing the desired acceleration. However, the actual acceleration can be slightly less than this – within amount ϵ.

The actions are:

Input:
accel(c), $c \in \mathsf{R}$

Internal:
none

Output:
sample(u), $u \in \mathsf{R}$

The variables are the same as those of D_1, with the addition of an internal "deadline variable" *last-sample*. This deadline variable just keeps track of the next (absolute) time at which a *sample* output is scheduled to occur. Also, the initialization of *acc* is more constrained than it is in D_1, reflecting the assumption that the correct acceleration is in effect at the beginning. We can think of the system as if we initialized it with an initial sample output and control signal.

Input:
none

Output:
now $\in [0, a]$, initially 0
$v \in \mathsf{R}$, initially 0

Internal:
acc $\in \mathsf{R}$, initially $\frac{b}{a}$
$\dot{v} \in \mathsf{R}$, initially any value in $[\frac{b}{a} - \epsilon, \frac{b}{a}]$
last-sample $\in \mathsf{R}^{\geq 0}$, initially d

The non-e discrete steps are:

accel(c)
Effect:
acc $:= c$
$\dot{v} :=$ any value $\in [acc - \epsilon, acc]$

sample(u)
Precondition:
now = *last-sample*
$u = v$
Effect:
last-sample $:=$ *now* $+ d$

Thus, an *accel* step just sets the *acc* control variable, and resets the actual acceleration $\dot{v}$ accordingly. A *sample* step just announces the current velocity – the only information needed by the controller component. It does so exactly at the time scheduled in *last-sample*. Then it reschedules the sampling time to be exactly d in the future.

The trajectories of *Vehicle* are all the mappings w from left-closed subintervals I of $[0, a]$ to states of *Vehicle* such that:

1. *acc* and *last-sample* are unchanged in w.
2. $\dot{v}$ is an integrable function in w.
3. For all $t \in I$, the following conditions hold in state $w(t)$.
   (a) *now* $= w(0).now + t$.
   (b) *now* $\leq$ *last-sample*.
   (c) $\dot{v} \in [acc - \epsilon, acc]$.
   (d) $v = w(0).v + \int_0^t w(x).\dot{v}dx$.

These trajectories are quite similar to those that are permitted in D_1. The most important difference is that *acc* is now not permitted to change during trajectories; instead, it changes only as a result of discrete inputs (from the controller, presumably). However, $\dot{v}$ can change, as long as it stays within the required bounds. There is also a condition that prevents time from passing beyond the *last-sample* deadline. The following invariants are straightforward to prove.

Lemma 7.1 *In every reachable state of Vehicle, the following are true.*

1. $\dot{v} \in [acc - \epsilon, acc]$.
2. $last\text{-}sample \in [now, now + d]$.

7.2 *Controller*

The *Controller* HIOA represents the controller that decides on the desirable acceleration, i.e., the value that should be placed into *Vehicle*'s variable *acc*. It receives reports from the *Vehicle* of its current velocity v, and uses each such report to calculate a desired new acceleration. It sends this, before any further time passage, to the *Vehicle* in an *accel* action.

The external actions of the *Controller* form the "mirror image" of those of the *Vehicle*:

Input:
  $sample(u)$, $u \in \mathsf{R}$
Internal:
  none
Output:
  $accel(c)$, $c \in \mathsf{R}$

The variables are:

Input:
  none
Internal:
  $now \in [0, a]$, initially 0
Output:
  $sampled\text{-}vel \in \mathsf{R}$, initially 0
  none
  $last\text{-}accel \in \mathsf{R}^{\geq 0} \cup \{\infty\}$, initially ∞

Here, *sampled-vel* is intended to hold the sampled velocity, when the *Controller* receives a report about it. The *last-accel* variable is another deadline variable, intended to keep track of the next scheduled (absolute) time for an *accel* signal. Initially (until the *Controller* receives some velocity report), there is no scheduled signal, so $last\text{-}accel = \infty$.

The non-e discrete steps are:

$sample(u)$
  Effect:
    $sampled\text{-}vel := u$
    $last\text{-}accel := now$

$accel(c)$
  Precondition:
    $last\text{-}accel = now$
    $now \neq a$
    $c = \frac{b - sampled\text{-}vel}{a - now}$
  Effect:
    $last\text{-}accel := \infty$

The *sample* action just records the reported velocity, and schedules an *accel* action to happen before any further real time elapses. (We could alternatively have modelled a system in which there is some bounded delay before the *accel* action occurs.) The *accel* action recalculates the desired velocity, using the same formula as in D – pointing at the desired goal (a, b) – but this time, the calculation is based on the sampled velocity instead of the actual velocity. After the *accel* action, no further *accel* is scheduled, until a new *sample* occurs.

The trajectories of *Controller* are trivial – time just passes up to any time that does not exceed any current deadline. There is no interesting continuous behavior to be modelled. That is, the trajectories are all the mappings w from left-closed subintervals I of $[0, a]$ to states of *Controller* such that:

1. *sampled-vel* and *last-accel* are unchanged in w.
2. For all $t \in I$, the following conditions hold in state $w(t)$.
   (a) $now = w(0).now + t$.
   (b) $now \leq last\text{-}accel$.

7.3 *Impl*

The complete implementation *Impl* is the composition of the two HIOA's *Vehicle* and *Controller*, identifying the *sample* and *accel* actions, and then hiding those actions (making them internal).

We close this section with some properties of *Impl*. The first lemma gives simple invariants about *last-accel*. It says that *last-accel* is only used to schedule an event immediately, and that when it is being used, the recorded and actual velocities are identical.

Lemma 7.2 *In every reachable state of Impl, the following are true.*

1. $last\text{-}accel \in \{now, \infty\}$.
2. *If* $last\text{-}accel = now$ *then* $v = sampled\text{-}vel$.

The next lemma is a key lemma for the simulation proof. It expresses bounds on the *acc* variable, no matter where the reference point is in a sampling interval. The *acc* variable is set accurately initially, and at each sampling time. But in between, the accuracy of the value of *acc* can degrade. Lemma 7.3 gives appropriate guarantees at all times, even within the sampling intervals. Some general statement of this sort is needed for the inductive proof of the simulation of D_1 by *Impl*.

In the statement of Lemma 7.3, the assumption that $last\text{-}accel = \infty$ is used to avoid the case where the implementation automaton is in the middle of processing a new sampling output.

Lemma 7.3 *In every reachable state of Impl, the following are true.*

1. *If* $now \neq a$ *and* $last\text{-}accel = \infty$ *then* $acc \geq \frac{b-\epsilon(now+d-last\text{-}sample)-v}{a-now}$.
2. *If* $now \neq a$ *then* $acc \leq \frac{b-v}{a-now}$.

Notice that the lower bound expressed in case 1 varies during each sampling interval. At the beginning of the interval, we have $now + d = last\text{-}sample$, so the bound simplifies to $\frac{b-v}{a-now}$. At the other extreme, at the end of the interval, we have $now = last\text{-}sample$, and the bound simplifies to $\frac{b-\epsilon d-v}{a-now}$. The complete statement fills in guarantees for the intermediate points as well.

Proof:

1. The lower bound is proved by induction on the length of a hybrid execution, as usual. The lower bound claim is true initially, since initially $acc = \frac{b}{a}$, $now = 0$, $last\text{-}sample = d$, and $v = 0$.

   Now consider a discrete step starting from a reachable state. A *sample* step makes $last\text{-}accel = \infty$, which makes the claim vacuously true. On the other hand, an *accel* step explicitly sets *acc* to $\frac{b-sampled\text{-}vel}{a-now}$, which is equal to $\frac{b-v}{a-now}$ by Lemma 7.2, which suffices to show the inequality. (This uses the fact that $last\text{-}sample \leq now + d$, which follows from Lemma 7.1.)

   Finally, consider a $[0, t]$-trajectory w whose first state is reachable. In w, *acc* is unchanged, and $\dot{v} \geq acc - \epsilon$ everywhere, by Lemma 7.1. Therefore,

   $$\frac{w(t).v - w(0).v}{t} \geq acc - \epsilon,$$

   that is,

   $$w(t).v - w(0).v \geq (acc - \epsilon)t.$$

We know by inductive hypothesis that

$$acc \geq \frac{b - \epsilon(w(0).now + d - last\text{-}sample) - w(0).v}{a - w(0).now}.$$

In other words,

$$w(0).v \geq b - \epsilon(w(0).now + d - last\text{-}sample) - acc(a - w(0).now).$$

Adding, we get:

$$w(t).v \geq b - \epsilon(w(t).now + d - last\text{-}sample) - acc(a - w(t).now).$$

In other words,

$$acc \geq \frac{b - \epsilon(w(t).now + d - last\text{-}sample) - w(t).v}{a - w(t).now}$$

This is what we needed to show.

2. For the upper bound, the argument is similar. The upper bound claim is true initially, since initially $acc = \frac{b}{a}$, $now = 0$ and $v = 0$.

   Now consider a discrete step starting from a reachable state. A *sample* step does not change any of the quantities mentioned in the inequality, and so it preserves the inequality. On the other hand, an *accel* step explicitly sets acc to $\frac{b - sampled\text{-}vel}{a - now}$, which is equal to $\frac{b-v}{a-now}$ by Lemma 7.2, which suffices to show the inequality.

   Finally, consider a $[0, t]$-trajectory w whose first state is reachable. In w, acc is unchanged, and $\dot{v} \leq acc$ everywhere, by Lemma 7.1. Therefore,

$$\frac{w(t).v - w(0).v}{t} \leq acc,$$

that is,

$$w(t).v - w(0).v \leq acc \cdot t.$$

We know by inductive hypothesis that

$$acc \leq \frac{b - w(0).v}{a - w(0).now}.$$

In other words,

$$w(0).v \leq b - acc(a - w(0).now).$$

Adding, we get:

$$w(t).v \leq b - acc(a - w(t).now).$$

In other words,

$$acc \leq \frac{b - w(t).v}{a - w(t).now}.$$

This is what we needed to show.

■

7.4 *Impl* Implements D_1

We show that *Impl* implements D_1 (see Theorem 7.5 for the formal statement), using a simulation from *Impl* to D_1.

Define the relation $fsim_2$ from states of *Impl* to states of D_1 as follows. If s_{Impl} is a state of *Impl* and s_{D_1} is a state of D_1, then we say that $(s_{Impl}, s_{D_1}) \in fsim_2$ provided that:

1. $s_{Impl}.now = s_{D_1}.now$.
2. $s_{Impl}.v = s_{D_1}.v$.
3. $s_{Impl}.acc = s_{D_1}.acc$.
4. $s_{Impl}.\dot{v} = s_{D_1}.\dot{v}$.

That is, $fsim_2$ is the identity mapping on all the state components of D_1. Note that all the state components of D_1 are derived from the *Vehicle* state in *Impl*. This is because the abstract system only mentions vehicle behavior, not controller behavior.

Lemma 7.4 *$fsim_2$ is a simulation from Impl to D_1.*

Proof: For the start condition, note that any combination of initial values allowed for all the state components in *Impl* is also allowed in D_1.

Next, consider a discrete step $(s_{Impl}, \pi, s'_{Impl})$ of *Impl*, where s_{Impl} and s_{D_1} are reachable states of *Impl* and D_1, respectively, and $(s_{Impl}, s_{D_1}) \in fsim_2$. There are two cases (again ignoring the trivial e case):

1. π is a *sample* action.

   Then we take the corresponding hybrid execution fragment to be trivial – just the trivial trajectory containing the single state s_{D_1}. It is easy to see that the step and the trivial trajectory have the same hybrid trace. Also, $(s'_{Impl}, s_{D_1}) \in fsim_2$, since this step does not change anything that affects any of the state components of D_1.

2. $\pi = accel(c)$.

   Now we take the corresponding hybrid execution fragment of D_1 to consist of a single *reset* step, $(s_{D_1}, reset, s'_{D_1})$. The state s'_{D_1} is obtained from the state s_{D_1} by modifying the *acc* and $\dot{v}$ components to their values in s'_{Impl}. The two steps have the same hybrid trace. Since π does not modify *now* or v, it should be clear that $(s'_{Impl}, s'_{D_1}) \in fsim_2$. It remains to show that $(s_{D_1}, reset, s'_{D_1})$ is in fact a step of D_1.

   The step of *Impl* causes *acc* to be set to $\frac{b - sampled\text{-}vel}{a - now}$, which is equal to $\frac{b-v}{a-now}$ by Lemma 7.2. It also causes $\dot{v}$ to be set to something in the range $[acc - \epsilon, acc]$. These changes are permitted in a *reset* step of D_1.

Finally, we consider a $[0, t]$-trajectory w_{Impl} whose first state is reachable. We allow this to correspond to a trajectory w_{D_1} of D_1, defined by simply projecting the states of *Impl* on the state components of D_1. The correspondence between the trajectories is then immediate. It remains to show that w_{D_1} is in fact a trajectory of D_1. Specifically, we show:

1. $\dot{v}$ is an integrable function in w_{D_1}.

   This follows from the definition of a trajectory of *Vehicle*.

2. For all $t \in I$, the following conditions hold in state $w(t)$.

(a) $now = w(0).now + t$.
This follows from the definition of a trajectory of *Vehicle*.

(b) If $now \neq a$ then $acc \in [\frac{b-\epsilon d-v}{a-now}, \frac{b-v}{a-now}]$.
The upper bound follows from Lemma 7.3, part 2. For the lower bound, Lemma 7.3, part 1, implies that, throughout w_{Impl} (except possibly at the right endpoint, if $now = a$ there), we have:

$$acc \geq \frac{b - \epsilon(now + d - last\text{-}sample) - v}{a - now}.$$

(This uses the fact that $last\text{-}accel = \infty$ throughout a trajectory; this is true because if not, then *last-accel* must be equal to *now* at the beginning of the trajectory, which would not permit time to pass.) Then the fact that $last\text{-}sample \geq now$, stated in Lemma 7.1, yields the result.

(c) If $now \neq a$ then $\dot{v} \in [acc - \epsilon, acc]$.
This follows from the definition of a trajectory of *Vehicle*.

(d) $v = w(0).v + \int_0^t w(x).\dot{v}dx$.
This follows from the definition of a trajectory of *Vehicle*.

■

Now we can give the basic theorem relating *Impl* to D_1:

Theorem 7.5 *If α_{Impl} is a hybrid execution of Impl, then there is a hybrid execution α_{D_1} of D_1 having the same hybrid trace.*

Theorem 7.5 implies that the changes in *now* and v that are exhibited by *Impl* are allowed by D_1. The theorem does not mention the values of the other variables of D_1, *acc* and $\dot{v}$, but of course those correspond as well. We could have obtained this conclusion simply by regarding *acc* and $\dot{v}$ as output variables instead of internal variables.
Proof: By Lemma 7.4 and Theorem 2.1. ■

We can combine the results stated in Theorems 7.5 and 6.6 to obtain the following result, which relates the implementation *Impl* to the high-level specification automaton V_1. This is the main result of the paper.

Theorem 7.6 *If α_{Impl} is a hybrid execution of Impl, then there is a hybrid execution α_{V_1} of V_1 having the same hybrid trace.*

Theorem 7.6 implies that the changes in *now* and v that are exhibited by *Impl* are allowed by V_1.
Proof: By Theorem 7.5 and Theorem 6.6. ■

8 Discussion

We have described a simple vehicle deceleration maneuver as a composition *Impl* of hybrid I/O automata. In this maneuver, deceleration is accomplished using a controller that receives accurate velocity information at equally spaced times, and instantly responds with control signals containing the desired acceleration. However, there is some uncertainty, in that the proposed acceleration might not be exhibited exactly by the vehicle.

We have also given a correctness specification for the range of allowed velocities at various times, as another HIOA V_1. V_1 gives, in a simple closed form, an "envelope" that includes the allowed velocities. The envelope is sufficiently large to encompass the effects of both the acceleration uncertainty and the sampling delays.

We have verified, using extensions of standard computer science techniques (methods for reasoning about discrete systems), that the implementation *Impl* meets the specification V_1. In particular, our proof uses invariants and levels of abstraction. Invariants involve real-world quantities such as the velocity and acceleration, as well as state components of the controller. Our proof interposes an additional level of abstraction between the implementation and the specification, in which the system's behavior is represented using differential equations; uncertainty is included at this level also. Again, the representation is sufficient to encompass the effects of both acceleration uncertainty and sampling delay. Ideas from differential equations and from discrete analysis fit clearly into the appropriate places in the proof.

Our proof that *Impl* satisfies the specification V_1 is broken down into separate pieces, corresponding to different facts to be shown and different types of mathematical tools. It combines continuous and discrete reasoning cleanly, in a single framework. It gives a completely accurate description of the system's guarantees, including correct handling of the uncertainty and the effects of sampling delays.

Note that some complications of continuous mathematics – definability of derivatives, proper handling of infinities, etc. – arise at the intermediate level only, not at the top and bottom level. The top level just gives an envelope demarcated by explicitly-defined continuous functions. The bottom level gives a discrete algorithm. It is only the intermediate level of abstraction that uses the differential representation, and at which the complications of infinities arise.

Of course, this example is very simplified. It remains to generalize it to cases that include more uncertainty: the sampling times might be only approximately known, or velocity information might be inexact or out-of-date, or the control signal might be sent only after some approximately known delay. We have only considered uncertainty in the lower bound, but of course there could also be uncertainty in the upper bound. None of these cases appears to introduce any ideas that are different in principle, so we expect that the proofs we have given should extend to these cases. Another extension is that the implementation might be subject to a limit on the achievable acceleration (because of physical limitations or passenger comfort). It should be possible to use our techniques to reason about this situation also.

It should also be possible to continue our example by refining further. A natural extension would be to implement the discrete *Controller* using a more complicated algorithm, e.g., a distributed algorithm with its own difficulties of communication, uncertainty, etc. Techniques of discrete reasoning (only) could be used to show the correspondence between the more detailed controller and the more abstract controller of this paper. Then general composition theorems about HIOA's could be used to show that the combination of the new controller implementation and the given *Vehicle* automaton still guarantee the proper behavior of the vehicle, as expressed by V_1.

Our general strategy can be described as: using levels of abstraction to represent the relationship between a derivative and explicit form of a system representation, and also between a discrete and a continuous form, while incorporating uncertainties accurately throughout. It remains to use the same general strategy to model and verify other maneuvers, in particular, more complex ones. These two splits seem likely to be useful in many other examples.

We could use more levels of abstraction to represent more levels of derivatives. For example, if vehicle position at various times were the important consideration, then vehicle position only might be constrained at the top level, with velocity at the next level, acceleration at another level below

that, and jerk at a fourth level, below the acceleration level. The correspondence between each successive pair of levels related by differentiation would just use standard methods of reasoning about differential equations (for the continuous parts of the correspondence).

Finally, the sort of reasoning we are doing in this paper admits assistance by mechanical reasoning tools. These could be a combination of a theorem-prover, for carrying out the discrete reasoning, and a tool for manipulating continuous function expressions, such as Mathematica. It is necessary to integrate the two types of tools so that they can be used together, using a single representation of the system.

References

[1] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In *DIMACS Workshop on Verification and Control of Hybrid Systems*, October 1995. To appear in R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science, Springer-Verlag.

On the Formal Verification of the TCAS Conflict Resolution Algorithms [1]

John Lygeros and Nancy Lynch

Laboratory for Computer Science
Massachusetts Institute of Technology
545 Technology Square, Cambridge, MA 02139
lygeros, lynch@lcs.mit.edu

Abstract

TCAS is an on-board protocol for detecting conflicts between aircraft and providing resolution advisories to the pilots. Because of its safety-critical role the TCAS software should ideally be "verified" before it can be deployed. The verification task is challenging, due to the complexity of the TCAS code and the hybrid nature of the system. We show how the essence of this very complicated problem can be captured by a relatively simple hybrid model, amenable to formal analysis. We then outline a methodology for establishing conditions under which the advisories issued by TCAS are safe.

1 Introduction

The Traffic Alert and Collision Avoidance System (TCAS) [1, 2] is an on-board aircraft conflict detection and resolution algorithm. Its task is to monitor air traffic in the vicinity of the aircraft and provide the pilot with information about neighboring aircraft that may pose a threat and advisories on how to resolve these conflicts. TCAS is a complex, safety-critical system that should be tested, or, even better, formally verified before it can be deployed. The TCAS software was developed through a sequence of progressive refinements, starting with abstract, high-level specifications that got refined down to a Statechart description, pseudo-code and finally regular computer code. Part of the verification problem involves proving that each level in this process implements the high-level specifications. Motivated by this example (and other applications to software development for large scale systems) techniques have been developed [3] for systematically carrying out this process. In addition, one also needs to investigate the performance of the closed loop system formed when the proposed algorithm is coupled with the aircraft dynamics. So far the primary verification technique used in this context has been simulation [4]. Successful results in extensive simulations provide a certain level of confidence in the algorithm. More importantly, unsuccessful simulation runs point to situations where performance is insufficient and often suggest modifications to improve it.

We believe that formal methods may be useful in this setting. The advantage of formal analysis over simulation is that it provides absolute guarantees about the system performance, under a set of assumptions. In addition, formal analysis may prove to be more efficient in the long run, as the results may be modified to accommodate changes in the algorithm; in comparison, a large number of simulations may have to be reexamined even for minor changes. So far the application of formal methods to this problem has been limited, primarily because of the complexity of the algorithm. Much of this complexity, however, is due to considerations such as human factors, which should be secondary to safety. In this paper we show how one can extract a relatively simple protocol, that encapsulates the essence of the TCAS algorithm from the safety point of view. The model we derive (outlined in Section 2) is amenable to formal analysis. This is illustrated in Section 3, where some preliminary analysis of the safety of the algorithm is conducted. We hope that once the analysis for this simple model is complete the complexity of the original algorithm can be gradually reintroduced, allowing us to prove more involved safety properties.

The TCAS system is *hybrid*, involving both continuous and discrete dynamics. The former arise from the aircraft, the sensors and the pilot reaction and the latter from the thresholds and discrete message passing used by the TCAS algorithm[1]. Therefore any verification effort will have to involve hybrid techniques. Our work makes use of a combination of techniques from control theory and distributed algorithms to tackle the verification problem. The methodology presented here has been successfully applied to other safety-critical transportation systems, such as automated highways [6, 7], personal rapid transit systems [8], train gate controllers

[1]Research supported by ARPA under F19628-95-C-0118, by AFOSR under F49620-97-1-0337, by UTC under DTRS95G-0001-YR8 and by PATH, under MOU-238.

[1]There are also important probabilistic effects, arising from sensor noise, uncertainty in the pilot response etc. These effects will be mostly suppressed in our work. For a discussion of probabilistic analysis for this problem the reader is referred to [5].

[9, 10] and aircraft conflict resolution [11].

2 System Model

2.1 Overview of the TCAS System

In cases of potential conflict the TCAS system enters one of two levels of alertness. In the lower level the system issues a *Traffic Advisory* (TA), to inform the pilot of a potential threat, without providing any suggestions on how to resolve the situation. If the danger of collision increases a *Resolution Advisory* (RA) is issued, providing the pilot with a maneuver that is likely to resolve the conflict. In this study we do not address TA's, because of the uncertainty in the pilot response and the low level of hazard involved.

The RA's issued by the TCAS II 6.04A version currently in use are restricted to the vertical plane. Maneuvers involve either climbing or descending at one of a finite number of fixed rates. If both aircraft are TCAS equipped, the algorithm [1, 2] uses a symmetry-breaking communication protocol to uniquely determine the maneuver that each aircraft should follow to resolve the conflict. Once a decision is reached the maneuver is presented to the pilots and is not altered until the conflict is resolved. TCAS II 6.04A has been extensively tested in simulation [4] and in practice.

A newer TCAS II version that is currently being tested also allows for *reversals*. RA's are still restricted to the vertical plane, but TCAS may change the advisory during a conflict. This feature was added primarily because of nondeterminism in the pilot response. If one (or both) of the pilots chooses not to follow the advisory, the original RA may become unsafe. TCAS detects this and changes the RA if necessary. Clearly, this type of algorithm is in greater need of verification; potential problems include live-lock and unnecessary reversals.

Future TCAS versions (TCAS IV) will produce RAs both in the horizontal and the vertical plane, while still maintaining the possibility of reversals. Our approach may be even more useful in this case, to provide design guidelines for TCAS versions that are still at a conceptual stage.

2.2 Overview of the Modeling Formalism

Following [12], we view a hybrid automaton, A, as a dynamical system that describes the evolution of a finite collection of variables, V_A. Variables are typed; for each $v \in V_A$ let $type(v)$ denote the type of v. For $Z \subseteq V_A$, a *valuation* of Z is a function that to each $v \in Z$ assigns a value in $type(v)$. Let $\mathbf{Z}$ denote the set of valuations of Z; we refer to $s \in \mathbf{V_A}$ as the *state* of A. In this paper we assume that the evolution of the variables is over the set of times $T^{\geq 0} = \{t \in \mathbb{R} | t \geq 0\}$. The evolution of the variables involves both continuous and discrete dynamics. Continuous dynamics are encoded in terms of *trajectories* over V_A, that is functions that map intervals of time to $\mathbf{V_A}$. Discrete dynamics are encoded by *actions*; upon the occurrence of an action the state instantaneously "jumps" to a new value.

More formally, a *hybrid automaton*, A is a collection, $(U_A, X_A, Y_A, \Sigma_A^{in}, \Sigma_A^{int}, \Sigma_A^{out}, \Theta_A, \mathcal{D}_A, \mathcal{W}_A)$, of three disjoint sets U_A, X_A, and Y_A of variables (called *input*, *internal*, and *output variables*, respectively) three disjoint sets Σ_A^{in}, Σ_A^{int}, and Σ_A^{out} of actions (called *input*, *internal*, and *output actions*, respectively) a nonempty set $\Theta_A \subseteq \mathbf{V_A}$ of *initial states*, a set $\mathcal{D}_A \subseteq \mathbf{V_A} \times \Sigma_A \times \mathbf{V_A}$ of *discrete transitions* and a set $\mathcal{W}_A$ of *trajectories* over V_A, where $V_A = U_A \cup X_A \cup Y_A$ and $\Sigma_A = \Sigma_A^{in} \cup \Sigma_A^{int} \cup \Sigma_A^{out}$. Some technical conditions need to be imposed on the above sets to guarantee that the definitions are consistent; see [12] for a discussion.

Let $fstate(w)$ ($lstate(w)$) denote the initial (final) state of a trajectory $w \in \mathcal{W}_A$ defined over a left (right) closed interval. An *execution*, α, of A is an alternating sequence $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$, with $w_i \in \mathcal{W}_A$ defined over a left closed time interval, $a_i \in \Sigma_A$, $fstate(w_0) \in \Theta_A$, and if w_i is not the last trajectory in α then its domain is a right-closed interval and $(lstate(w_i), a_{i+1}, fstate(w_{i+1})) \in \mathcal{D}_A$. If α is a finite sequence we assume it ends with a trajectory. An execution is called *finite* if it is a finite sequence and the domain of its final trajectory is right-closed. A state $s \in \mathbf{V_A}$ is called *reachable* if it is the last state of a finite execution.

Hybrid automata "interact" through shared variables and shared actions. Consider two automata A and B with $X_A \cap V_B = X_B \cap V_A = Y_B \cap Y_A = \emptyset$ and $\Sigma_B^{int} \cap \Sigma_A = \Sigma_A^{int} \cap \Sigma_B = \Sigma_A^{out} \cap \Sigma_B^{out} = \emptyset$. Under some mild technical assumptions, the *composition*, $A \times B$, of A and B can be defined as a new hybrid automaton with $U_{A\times B} = (U_A \cup U_B) \setminus (Y_A \cup Y_B)$, $X_{A\times B} = X_A \cup X_B$, $Y_{A\times B} = Y_A \cup Y_B$ (similarly for $\Sigma_{A\times B}$). $\Theta_{A\times B}$, $\mathcal{D}_{A\times B}$ and $\mathcal{W}_{A\times B}$ are defined so that the executions of $A \times B$ are executions of both A and B when restricted to the corresponding variables and actions.

A *derived variable* of A is a function on $\mathbf{V_A}$. Derived variables are used to simplify the system description and to facilitate the analysis. A *property* of A is a boolean derived variable. A property is *stable* if, whenever it is true at some state, it is also true at all states reachable from that state. A property is *invariant* if it is true at all reachable states. Typically properties will be shown to be stable or invariant by induction on the length of the executions. It is easy to see that:

Lemma 1 *If for all reachable states s, P is true at s implies that P is true for all s' such that there exists $a \in \Sigma_A$ with $(s, a, s') \in \mathcal{D}_A$ or there exists $w \in \mathcal{W}_A$ with right closed domain and* $\mathrm{fstate}(w) = s$ *and* $\mathrm{lstate}(w) = s'$, *then P is a stable property of A. If further P is true at all $s \in \Theta_A$, then P is an invariant property of A.*

In some places differential equations will be used to simplify the description of the set $\mathcal{W}_A$ (or at least parts of it). In this case, $\mathcal{W}_A$ is assumed to be populated by

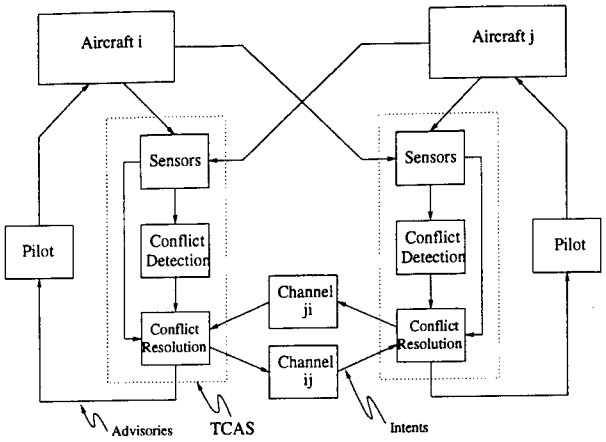


Figure 1: TCAS system components

all trajectories generated by the differential equation in the usual way. To simplify the description of $\mathcal{D}_A$, we will assign a *precondition* and an *effect* to each action. The precondition is a predicate on $\mathbf{V_A}$ while the effect is a predicate on $\mathbf{V_A} \times \mathbf{V_A}$. The corresponding transition can take place only from states that satisfy the precondition; moreover, the states before and after the transition should satisfy the effect.

2.3 The TCAS Model

We model TCAS by a composition of components (Figure 1). For each component a model was extracted from the TCAS documentation. The overall model is *closed*, in the sense that input variables and actions of one component are outputs of other components.

2.3.1 Aircraft Model: The system we consider consists of N aircraft, labeled $1, \ldots, N$. Each aircraft, i, is modeled by an HA, A_i. We assume $\Sigma^{in}_{A_i} = \Sigma^{int}_{A_i} = \Sigma^{out}_{A_i} = \emptyset$ and hence $\mathcal{D}_{A_i} = \emptyset$. Each aircraft is identified by a unique Mode_S number, stored in an output variable $Mode\_S_i \in \mathbb{N}$. Each aircraft may or may not be equipped with an altitude reporting transponder; if it is, it may also be equipped with TCAS. This hardware information is stored on an output variable $Equipment_i \in \{\text{None}, \text{Report}, \text{TCAS}\}$.

The physical movement of the aircraft is summarized by the trajectories of its position and velocity. Let $p_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, $v_i = (v_i^x, v_i^y, v_i^z) \in \mathbb{R}^3$ and $a_i = (a_i^x, a_i^y, a_i^z) \in \mathbb{R}^3$ be the position, velocity and acceleration of the aircraft with respect to some fixed reference frame on the ground. We assume that all trajectories in $\mathcal{W}_{A_i}$ satisfy the differential equation:

$$\begin{bmatrix} \dot{p}_i(t) \\ \dot{v}_i(t) \end{bmatrix} = \begin{bmatrix} v_i(t) \\ a_i(t) \end{bmatrix} \qquad (1)$$

We assume that the aircraft acceleration is under the direct control of the pilot and set $Y_{A_i} = \{Mode\_S_i, Equipment_i, p_i, v_i\}$, $U_{A_i} = \{a_i\}$ and $X_{A_i} = \emptyset$. The dynamics of equation (1) are very simple and ignore important aircraft characteristics such as the details of the aerodynamic forces, high frequency modes, the effect of structural controls and input constraints. Equation (1) should be sufficient in our case, however, as the maneuvers required by TCAS are rather mild.

2.3.2 Sensors: Each aircraft is equipped with sensors that return information about its state and the state of neighboring aircraft. The sensor of aircraft i is modeled by an HA, S_i with $U_{S_i} = \{p_j, v_j\}_{j=1}^N$. The output variables of S_i are estimates of the altitude, $h_i^j \in \mathbb{R}^+$, and vertical rate, $\dot{h}_i^j \in \mathbb{R}$, for all aircraft and the distance (range), $R_i^j \in \mathbb{R}^+$, and its rate $\dot{R}_i^j \in \mathbb{R}$ between aircraft i and each neighboring aircraft j. We set $\Sigma^{in}_{S_i} = \Sigma^{int}_{S_i} = \emptyset$.

The information that the sensors provide about the aircraft state is quantized spatially and sampled temporally. We assume that the output variables of the sensor automaton fall within an interval centered at the "correct" values dictated by the actual state of the system. Let n_A, n_{AR}, n_R and n_{RR} denote the width of the intervals for h_i^j, $\dot{h}_i^j$, R_i^j and $\dot{R}_i^j$ respectively. The output variables of the sensors are updated every T_s seconds, upon the occurrence of an output action $Sample_i$. An internal variable $T_i \in \mathbb{R}$ keeps track of the time that has elapsed since the last sample.

2.3.3 Conflict Detection: The role of the conflict detection automaton, D_i, is to determine whether neighboring aircraft pose a threat. The input variables of D_i are the output variables of S_i, as well as boolean variables $Threat_i^j$ which indicate whether the conflict resolution automaton is already aware of the threat. We set $\Sigma^{in}_{D_i} = \Sigma^{int}_{D_i} = \emptyset$ and $X_{D_i} = Y_{D_i} = \emptyset$.

Aircraft j is declared a threat by aircraft i upon the occurrence of an output action $Declare_i^j$ and ceases to be regarded as a threat upon the occurrence of an output action $Undeclare_i^j$. Two derived boolean variables, *Range_Test* and *Altitude_Test*, are used to determine the preconditions of these actions. The *Range_Test* encodes the conditions that the range and range rate need to satisfy for aircraft j to be declared a threat:

$$Range\_Test = (\dot{R}_i^j > 10ft/s \wedge \hat{R}_1) \vee (\dot{R}_i^j \leq 10ft/s \wedge \hat{R}_2)$$

where:

$$\hat{R}_1 = (R_i^j \leq \text{DM}) \wedge (R_i^j \dot{R}_i^j \leq \text{H1})$$
$$\hat{R}_2 = (R_i^j \leq 12nmi) \wedge \left(\frac{R_i^j - \text{DM}^2/R_i^j}{\min\{\dot{R}_i^j, -10ft/s\}} \leq \text{TR} \right)$$

The *Altitude_Test* is based on the predicted vertical separation at $\tau = |R_i^i / \min\{\dot{R}_i^j, -10ft/s\}|$, the "time of closest approach".

$$Altitude\_Test = \left| (h_i^i - h_i^j) - (\dot{h}_i^i - \dot{h}_i^j)\tau \right| \leq \text{ZT}$$

DM, H1, TR and ZT are TCAS parameters that depend on the current altitude.

At this stage we assume that j is declared a threat by i as soon as it "passes" both range and altitude tests. In practice a number of exceptions to this rule

are introduced in the TCAS implementation, mostly to reduce the number of false alarms. Once declared a threat, j continues to be considered a threat until it fails the range test. At this point the action $Undeclare_i^j$ takes place.

2.3.4 Conflict Resolution: Conflict resolution is modeled by an HA, R_i, (Appendix A) with $U_{R_i} = Y_{S_i} \cup \{Mode\_S_j, Equipment_j\}_{j=1}^{N}$. The output variables of R_i are $Threat_i^j$ and a resolution advisory for the pilot, consisting of a $Sense_i \in \{\text{Climb, Descend}, \perp\}$ and a $Strength_i \in \{$ -2000, -1000, -500, 0, 1500, 2500 $\}$ (in ft/min). The sense indicates whether i should try to pass above (Climb) or below (Descend) the intruding aircraft. $Sense_i = \perp$ (undefined) indicates that no action is needed. Strength provides a bound on the vertical speed to ensure sufficient vertical separation at time τ. R_i maintains two internal variables, the boolean $Reversed_i$ that keeps track of whether the sense selection has already been reversed during the current encounter and $Intent\_Sent_i^j \in \{\text{Climb, Descend}, \perp\}$ that keeps track of the last intent message sent by i to j. Intent messages can be thought of as "commands" to j as to which sense it should select[2].

R_i has no internal actions. Sense selection can happen when j is first declared a threat (upon the occurrence of input action $Declare_i^j$), whenever an intent message is received from another TCAS equipped aircraft (upon occurrence of input action $Receive_i^j(\text{dir})$ with dir $\in \{\text{Climb, Descend}\}$, and whenever new data comes in from the sensors (upon occurrence of input action $Sample_i$). The advisory is removed whenever the intruding aircraft ceases to be considered a threat (upon occurrence of input action $Undeclare_i^j$). i communicates its intent to j through an output action, $Send_i^j(\text{dir})$ with dir $\in \{\text{Climb, Descend}\}$.

Sense selection is based on the predicted vertical separation at time τ. Consider first the case of a climb advisory. To predict the vertical separation TCAS assumes that the intruding aircraft will maintain its current speed. If $\dot{h}_i^i \geq 1500$, TCAS assumes that the pilot will maintain the current climb rate. The vertical separation at time τ is then given by:

$$\Delta z(\text{Climb}) = (h_i^i - h_i^j) + (\dot{h}_i^i - \dot{h}_i^j)\tau$$

If $\dot{h}_i^i < 1500$ on the other hand, TCAS assumes that the pilot will respond to the advisory after a delay d by applying a constant vertical acceleration $a_i^z \equiv a$ until $\dot{h}_i^i = 1500 ft/min$. A similar expression produces the value of $\Delta z(\text{Climb})$ in this case. The climb separation is adequate if $\Delta z(\text{Climb})$ is above a threshold ALIM. The predicted separation in case of a descent advisory, $\Delta z(\text{Descend})$, can be similarly calculated.

Aircraft i issues an advisory against aircraft j for the first time when either the conflict detection automaton declares it a threat or when j sends an intent message (indicating that it has already issued an advisory against i). In the former case, i (the first of the two to detect the conflict) chooses an advisory sense based on a derived variable $Indep\_Choice$. If neither climb nor descent provide adequate separation, the one that produces the largest separation is chosen[3]. If one produces adequate separation but the other does not, the one that does is chosen. If both produce adequate separation preference is given to the non-crossing advisory (a climb if i is already higher and a descent if it is lower). If j has already issued an advisory, the complementary sense (encoded by the received intent) is typically chosen. The only exception is if i has a lower Mode_S number, the received intent is crossing (j is higher and has requested i to Climb or it is lower and has requested i to Descend) and i believes a non-crossing advisory is possible.

The sense may be reversed later on if, for example, one (or both) of the pilots thwarts the advisory. If j is not TCAS equipped, i reverses its advisory whenever it is predicted that the current advisory will not lead to adequate separation, while the reversed advisory will. The same is more or less true if j is TCAS equipped but i has a lower Mode_S number[4]. The only difference is that in this case i can only reverse once and then only if the current advisory is crossing. The new intent is communicated to j which is forced to change its advisory accordingly.

The advisory strength is updated every time new data becomes available. The choice of $Strength_i$ depends on the predicted vertical separation at time τ. The new strength is chosen according to a derived variable $Strength\_Choice$, which returns the smallest strength that will provide separation at least $ALIM$ at time τ. For example, if $Sense_i = \text{Climb}$, $(h_i^i - h_i^j) + (-500 - \dot{h}_i^j)\tau \geq ALIM$ and $(h_i^i - h_i^j) + (-1000 - \dot{h}_i^j)\tau < ALIM$ then $Strength\_Choice = -500$.

2.3.5 Communication Channel: Communication of intents is achieved through a communication channel automaton, C_{ij}. The automaton has an input action $Send_i^j(\text{dir})$, whose effect is to store the intent, dir, together with a time stamp in an internal multiset. The message is delivered (and removed from the multi set) upon occurrence of the output action $Receive_i^j(\text{dir})$. Delivery is guaranteed by at most d_{ij} time units from the time the message was sent.

2.3.6 Pilot: The pilot is modeled by an HA, P_i, with $U_{P_i} = \{Sense_i, Strength_i, \dot{h}_i^i\}$ and $Y_{P_i} = \{a_i\}$. The pilot may choose not to follow a particular advisory or to follow it after some delay. This information is stored

[2] In the TCAS code a Climb intent is referred to as a "Do not Descend" and a Descend intent as a "Do not Climb".

[3] It is assumed that conflict detection will take place early enough so that this case will never have to be exercised. We only include it here for completeness.

[4] The aircraft with the higher Mode_S number can not initiate a reversal.

in the boolean variable $Follow_i$ and the real variable d_i. The pilot automaton has no input or output actions. An internal action $New\_Advisory_i$ takes place whenever the advisory changes.

We assume that the pilot can apply a range of accelerations in each of the three directions, $a_i(t) \in [\underline{a}_i, \overline{a}_i] = [\underline{a}_i^x, \overline{a}_i^x] \times [\underline{a}_i^y, \overline{a}_i^y] \times [\underline{a}_i^z, \overline{a}_i^z]$. We also assume that the pilot tries to keep v_i^z in $[\underline{v}_i^z, \overline{v}_i^z]$. The width of the ranges reflects considerations such as passenger comfort and standard pilot practice. To ensure that all advisories can be followed we assume that $\underline{a}_i \leq -a < 0 < a \leq \overline{a}_i$, $[-2500, 2500] \subseteq [\underline{v}_i^z, \overline{v}_i^z]$ and $v_i^z(0) \in [\underline{v}_i^z, \overline{v}_i^z]$.

Whenever a new advisory comes in the pilot decides if it will be followed and chooses a delay $d_i \in [\underline{d}_i, \overline{d}_i]$. We assume that if the pilot chooses not to follow an advisory (or when none is present) he/she arbitrarily sets the vertical acceleration in the interval $[\underline{a}_i, \overline{a}_i]$. If the pilot chooses to follow the advisory, he/she is assumed to respond by at most d_i, by applying a constant vertical acceleration $a_i^z = a$ until $Strength_i$ is reached; a pilot is assumed to set $a_i^z = 0$ if the current vertical rate meets the advisory strength. One can show that:

Lemma 2 *$v_i^z(t) \in \left[\underline{v}_i^z - \frac{n_{AR}}{2}, \overline{v}_i^z + \frac{n_{AR}}{2}\right]$ for all $t \geq 0$.*

3 Verification Example

To illustrate how safety properties of the TCAS algorithm may be analyzed, consider a *pair of well-behaved aircraft*, defined as a system that satisfies:

Assumption 1 *$N = 2$, $Equipment_i =$ TCAS, $a_i^x(t) = a_i^y(t) = 0$ and $Follow_i(t) =$ True for $t \geq 0$ and $i = 1, 2$.*

Let $\Delta x = x_1 - x_2$, $\Delta v_x = v_1^x - v_2^x$, etc. Consider the case where after a finite number of advisory changes, the TCAS algorithm converges to a fixed pair of advisories $(Sense_1, Strength_1)$ and $(Sense_2, Strength_2)$. Assume that the final advisories are "consistent":

Assumption 2 *There exists $d_a \geq 0$ such that for all $t \geq d_a$ and for $i = 1, 2$, $\text{Sense}_i(t)$ are constant, $\text{Sense}_i(t) \neq \perp$ and $\text{Sense}_1(t) \neq \text{Sense}_2(t)$.*

Without loss of generality assume that $Sense_1 =$ Climb. Let $\Delta v_z^a = Strength_1 + Strength_2$ represent the minimum difference in vertical speed dictated by the advisory. One can show that:

Lemma 3 *There exists $d \geq 0$ such that for all $t \geq d$, $a_i^z(t) = 0$ and $\Delta v_z(t) \geq \Delta v_z^a - n_{AR}$.*

Let $\delta = d - t$ and consider the derived variable:

$$S_{\Delta v_z^a} = \Delta z + \delta(\underline{v}_1^z - \overline{v}_2^z - n_{AR}) - (\Delta v_z^a - n_{AR}) \frac{(\Delta x + \delta \Delta v_x)\Delta v_x + (\Delta y + \delta \Delta v_y)\Delta v_y}{\Delta v_x^2 + \Delta v_y^2}$$

$$S_{\Delta v_z^a} = \Delta z - (\Delta v_z^a - n_{AR}) \frac{\Delta x \Delta v_x + \Delta y \Delta v_y}{\Delta v_x^2 + \Delta v_y^2}$$

if $t \leq d$ and $t > d$ respectively.

Lemma 4 *$(S_{\Delta v_z^a} \geq ALIM)$ is a stable property of a pair of well behaved aircraft.*

Proof (sketch): None of the quantities in the right hand side of $S_{\Delta v_z^a}$ are affected by any of the system actions. Therefore, if $(S_{\Delta v_z^a} \geq ALIM)$ is true at the pre-state of an action it is also true at its post-state. Note that $S_{\Delta v_z^a}$ is continuous as a function of time and $\dot{S}_{\Delta v_z^a} = \Delta v_z - (\underline{v}_1^z - \overline{v}_2^z - n_{AR})$ if $t < d$ and $\dot{S}_{\Delta v_z^a} = \Delta v_z - (\Delta v_z^a - n_{AR})$ if $t > d$. In either case $\dot{S}_{\Delta v_z^a}(t) \geq 0$ (by Lemmas 2 and 3 respectively). Therefore, if $(S_{\Delta v_z^a} \geq ALIM)$ is true at the first state of a trajectory, it will also be true at the last state. ∎

The quantity $S_{\Delta v_z^a}$ is related to the safety of the system. Consider the horizontal separation of the two aircraft $R_{xy} = \sqrt{\Delta x^2 + \Delta y^2}$. Consider the time $T = -\frac{\Delta x(0)\Delta v_x + \Delta y(0)\Delta v_y}{\Delta v_x^2 + \Delta v_y^2}$ and assume that $T > d$; this simply requires that the aircraft be far enough for the pilots to implement the advisory before the point of closest horizontal approach.

Theorem 1 *If $S_{\Delta v_z^a}(0) \geq ALIM$ then the vertical separation at the point of closest horizontal approach will be at least ALIM.*

Proof (sketch): At time T, R_{xy} achieves its minimum value. By Lemma 4, $S_{\Delta v_z^a}(0) \geq ALIM$ implies $(S_{\Delta v_z^a} \geq ALIM)$ is an invariant property. At time T, $\Delta x(T)\Delta v_x + \Delta y(T)\Delta v_y = 0$. Therefore, $S_{\Delta v_z^a}(T) \geq ALIM$ implies $\Delta z(T) \geq ALIM$, i.e. the vertical separation when the horizontal separation becomes minimum being at least $ALIM$. ∎

4 Current & Future Research

Section 3 contains only a small part of the argument needed to show safety even for this simplified system. Assumption 2 clearly needs to be shown to be a property of the algorithm. This will complete a safety theorem for a pair of well-behaved aircraft. The proof then needs to be extended by relaxing Assumption 1: we need to investigate what happens if multiple aircraft are present, if the pilots accelerate in the x and y directions and if one of the pilots chooses not to follow the advisory. The last extension should also provide insight into the case of an unequipped threat. The analysis is complicated further in this case as multiple reversals are possible.

All proofs discussed so far will be based on the assumption that the model of Section 2 adequately captures the system. This model contains a number of simplifications, in the aircraft dynamics, the TCAS algorithm and the pilot response. These simplifications can be progressively removed. We hope that once a proof for the above nominal case is available, it can be extended to other cases, possibly using *abstraction relations.*

References

[1] Radio Technical Commission for Aeronautics, "Minimum operational performance standards for

TCAS airborn equipment", Tech. Rep. RTCA/DO-185, RTCA, September 1990, Consolidated Edition.
[2] The MITRE Corporation, "TCAS II collision avoidance subsystem requirements specification", 1996.
[3] Nancy Leveson, *SafeWare : system safety and computers*, Addison-Wesley, 1995.
[4] A.C. Drumm, "Lincoln laboratory evaluation of TCAS II logic version 6.04a", Tech. Rep. ATC-240, Lincoln Laboratory, MIT, February 1996.
[5] James K. Kuchar, *A Unified Methodology for the Evaluation of Hazard Alerting Systems*, PhD thesis, Massacussets Institute of Technology, 1995.
[6] John Lygeros, Datta N. Godbole, and Shankar Sastry, "A verified hybrid controller for automated vehicles", in *CDC*, 1996, pp. 2289–2294.
[7] Ekaterina Dolginova and Nancy Lynch, "Safety verification for automated platoon maneuvers: a case study", in *Proceedings of HART97*, number 1201 in LNCS, pp. 154–170. Springer Verlag, 1997.
[8] H.B. Weinberg, Nancy Lynch, and Norman Delisle, "Verification of automated vehicle protection systems", in *Hybrid Systems III*, number 1066 in LNCS, pp. 101–113. Springer Verlag, 1996.
[9] C. Heitmeyer and N. Lynch, "The generalized railroad crossing: A case study in formal verification of real-time systems", in *Proc. ICCC Real-Time Systems Symposium*, San Juan, Puerto Rico, 1994.
[10] John Lygeros, Datta N. Godbole, and Shankar Sastry, "A game theoretic approach to hybrid system design", in *Hybrid Systems III*, number 1066 in LNCS, pp. 1–12. Springer Verlag, 1996.
[11] George J. Pappas, Claire Tomlin, and Shankar Sastry, "Conflict resolution for multi-agent hybrid systems", in *CDC*, 1996.
[12] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H.B. Weinberg, "Hybrid I/O automata", in *Hybrid Systems III*, number 1066 in LNCS, pp. 496–510. Springer Verlag, 1996.

A Conflict Resolution Automaton R_i

Data Types:
Dir = $\{Climb, Descend\}$, $\text{Dir}_\perp = \text{Dir} \cup \{\perp\}$
Strengths = $\{-2000, -1000, -500, 0, 1500, 2500\}$
Aircraft = $\{1, \ldots, N\}$, $\text{Others}_i = \text{Aircraft} \setminus \{i\}$

Input Variables:
$Mode\_S_j \in \mathbb{N}$, $j \in$ Aircraft
$Equipment_j \in \{\text{None}, \text{Report}, \text{TCAS}\}$, $j \in$ Aircraft
$h_i^j \in \mathbb{R}^+$ and $\dot{h}_i^j \in \mathbb{R}$ for $j \in$ Aircraft
$R_i^j \in \mathbb{R}^+$ and $R_i^j \in \mathbb{R}$, for $j \in \text{Others}_i$

Internal Variables:
$Reversed_i \in$ **Bool**, initially False
$Intent\_Sent_i^j \in \text{Dir}_\perp$, $j \in \text{Others}_i$, initially $\perp$

Output Variables:
$Sense_i \in \text{Dir}_\perp$, initially $\perp$
$Threat_i^j \in$ **Bool**, $j \in \text{Others}_i$, initially False
$Strength_i \in$ Stengths, initially 0

Derived Variables (see text):
$\Delta z(\text{dir}) \in \mathbb{R}$ and $OK(dir) \in$ **Bool**, dir $\in$ Dir
$Indep\_Choice \in$ Dir
$Strength\_Choice \in$ Strengths

Input Actions:
$Declare_i^j$ and $Undeclare_i^j$ for $j \in \text{Others}_i$
$Receive_i^j(\text{dir})$, $j \in \text{Others}_i$, dir $\in$ Dir
$Sample_i$

Output Actions:
$Send_i^j(\text{dir})$, $j \in \text{Others}_i$, dir $\in$ Dir

Discrete Transitions:

$Declare_i^j$:
Effect: if $\neg Threat_i^j$ then
$Threat_i^j := \text{True}$; $Sense_i = Indep\_Choice$

$Undeclare_i^j$:
Effect: if $Threat_i^j$ then
$Threat_i^j := \text{False}$; $Intent\_Sent_i^j = \perp$
$Sense_i := \perp$; $Reversed_i := \text{False}$

$Receive_i^j(\text{dir})$:
Effect: if $(Mode\_S_i > Mode\_S_j)$ then $Sense_i :=$ dir
if $\neg Threat_i^j$ then
$Threat_i^j := \text{True}$
if $(Mode\_S_i < Mode\_S_j)$ then
if $(\text{dir} = \text{Climb} \wedge h_i^i \geq h_i^j)$
then $Sense_i := \text{Climb}$
elseif $(\text{dir} = \text{Descend} \wedge h_i^i \leq h_i^j)$
then $Sense_i := \text{Descend}$
else $Sense_i = Indep\_Choice$

$Sample_i$:
Effect: if $Threat_i^j$ then
if $Equipment_j \neq TCAS \wedge OK(\text{Climb})$
$\wedge \neg OK(\text{Descend})$ then $Sense_i := \text{Climb}$
if $(Equipment_j \neq TCAS \wedge \neg OK(\text{Climb})$
$\wedge OK(\text{Descend})$ then $Sense_i := \text{Descend}$
if $Equipment_j = TCAS \wedge Mode\_S_i < Mode\_S_j$
$\wedge \neg Reversed)$ then
if $Sense_i = \text{Descend} \wedge OK(\text{Climb})$
$\wedge \neg OK(\text{Descend}) \wedge h_i^i \geq h_i^j$ then
$Sense_i := \text{Climb}$; $Reversed_i := \text{True}$
if $Sense_i = \text{Climb} \wedge \neg OK(\text{Climb})$
$\wedge OK(\text{Descend}) \wedge h_i^i \leq h_i^j)$ then
$Sense_i := \text{Descend}$; $Reversed := \text{True}$
$Strength_i = Strength\_Choice$

$Send_i^j(\text{dir})$:
Precondition:
$(Sense_i = \text{Climb} \wedge Intent\_Sent_i^j \neq \text{Descend}$
$\wedge \text{dir} := \text{Descend}) \vee$
$(Sense_i = \text{Descend} \wedge Intent\_Sent_i^j \neq \text{Climb}$
$\wedge \text{dir} := \text{Climb})$
Effect: $Intent\_Sent_i^j :=$ dir

Trajectories:
Input variables follow arbitrary trajectories.
Internal and output variables remain constant.
Trajectories stop as soon as the precondition of $Send_i^j(\text{dir})$ becomes true.

Reproduced by NTIS

National Technical Information Service
Springfield, VA 22161

This report was printed specifically for your order from nearly 3 million titles available in our collection.

For economy and efficiency, NTIS does not maintain stock of its vast collection of technical reports. Rather, most documents are printed for each order. Documents that are not in electronic format are reproduced from master archival copies and are the best possible reproductions available. If you have any questions concerning this document or any order you have placed with NTIS, please call our Customer Service Department at (703) 605-6050.

About NTIS

NTIS collects scientific, technical, engineering, and business related information — then organizes, maintains, and disseminates that information in a variety of formats — from microfiche to online services. The NTIS collection of nearly 3 million titles includes reports describing research conducted or sponsored by federal agencies and their contractors; statistical and business information; U.S. military publications; multimedia/training products; computer software and electronic databases developed by federal agencies; training tools; and technical reports prepared by research organizations worldwide. Approximately 100,000 *new* titles are added and indexed into the NTIS collection annually.

For more information about NTIS products and services, call NTIS at 1-800-553-NTIS (6847) or (703) 605-6000 and request the free *NTIS Products Catalog,* PR-827LPG, or visit the NTIS Web site **http://www.ntis.gov**.

NTIS
Your indispensable resource for government-sponsored information—U.S. and worldwide

U.S. DEPARTMENT OF COMMERCE
Technology Administration
National Technical Information Service
Springfield, VA 22161 (703) 605-6000