Safety Verification for Automated Vehicle Maneuvers

by

Ekaterina Dolginova

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering in Electrical Engineering and Computer Science

and

Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1998

Signature of Author .....
Department of Electrical Engineering and Computer Science
May 22, 1998

Certified by .....
Nancy A. Lynch
NEC Professor of Software Science and Engineering
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Safety Verification for Automated Vehicle Maneuvers

by

Ekaterina Dolginova

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 1998, in partial fulfillment of the
requirements for the degrees of
Master of Engineering in Electrical Engineering and Computer Science
and
Bachelor of Science in Computer Science and Engineering

Abstract

In this thesis we formally model a system consisting of two vehicles moving along a single track, plus controllers that operate the vehicles, plus communication channels. The modeling formalism used is the Hybrid Automata model developed by Lynch, Segala, Vaandrager and Weinberg. We formulate a key safety requirement of such a system, namely, that the two vehicles never collide at a relative velocity greater than a given bound, v_{allow}. We give necessary and sufficient conditions for the controller of the follower vehicle to guarantee that the safety requirement is satisfied regardless of the behavior of the leading vehicle. The model includes handling of communication delays and uncertainty. The proofs use composition, invariants, and levels of abstraction, together with methods of mathematical analysis. This case study is derived from the California PATH intelligent highway project.

Thesis Supervisor: Nancy A. Lynch
Title: NEC Professor of Software Science and Engineering

Acknowledgments

Many people helped me in working on and completing my Master's Thesis. I would like to thank my advisor Professor Nancy A. Lynch. Her knowledge and insights have been an invaluable asset to my research. In addition, her patience in reading and correcting many drafts of this thesis were essential to the successful completion of this work. I would also like to thank Carolos Livadas for answering many of my questions on the HA model and for making many wise suggestions on the prelimenary versions of this thesis. Furthermore, I would like to thank Edward Kogan for proofreading several drafts of this paper.. I am also grateful to Aleksandr Bernshteyn for his continuous support, for bearing with me at times of stress, and for his willingness to talk to me about my research problems. Finally, I would like to thank my family for its continuous love and support and all my friends at MIT for making my student life enjoyable.

Contents

1 Introduction **9**

1.1 Problem Statement and Motivation 9

1.2 Related Work 11

1.3 Our Approach 12

1.4 Contributions of this Research 12

2 Hybrid Automata Model **15**

2.1 Hybrid Automata 15

2.2 Hybrid Executions and Hybrid Traces 16

2.3 Simulation Relations 17

2.4 Composition 18

2.5 Hiding 19

2.6 Modeling Conventions in This Thesis 19

3 Math Preliminaries **21**

3.1 Non-Negativity Theorems 21

3.2 Non-Increasing Functions 24

3.3 Derivatives of The *max* Function 25

4 System Model **29**

4.1 Vehicles 29

4.2 Controllers 33

4.3 Safety Condition 35

5 Safety In The Ideal Case **37**

5.1 Problem Statement 37

5.2 The Model 37

5.3 Correctness of C_2 39

5.4 Optimality of C_2 44

5.5 Results 50

6 Delayed Response **51**

6.1 Delay Buffers 52

6.2 The System with Inbound and Outbound Delays 54

6.3 Correctness of *Delayed-Controller*$_2$ 58

6.4 Optimality of D_2 64

7 Uncertainty **69**

7.1 The Uncertainty Buffer 70

7.2 The System 71

7.3 Correctness of U_2 75

7.4 Optimality 79

8 Conclusion and Future Work **81**

List of Figures

4-1 Vehicles 29

4-2 The *Vehicles* Hybrid Automaton 30

4-3 $Controller_j$ Hybrid Automaton, $j \in \{1, 2\}$ 33

4-4 *Controlled-Vehicles* Model 35

5-1 C_2 Hybrid Automaton 38

5-2 The *Init-Vehicles* Hybrid Automaton 40

5-3 *Necessary-*C_2 Hybrid Automaton 45

6-1 *D-Buffer*(n, S_i, S_o, var, d, $Init$) Hybrid Automaton 52

6-2 *Spec-*D_2 Hybrid Automaton 56

6-3 *Delayed-Controller*$_2$ (D) hybrid automaton 57

6-4 D_2 Hybrid Automaton 59

6-5 The timing diagram of the in- and outbound controller 61

6-6 *Necessary-*D_2 Hybrid Automaton 65

7-1 *U-Buffer* (n, S_i, S_o, var) Hybrid I/O Automaton 70

7-2 Sensor-Uncertainty Vehicles Model 72

7-3 U_2 Hybrid I/O Automaton 73

Chapter 1

Introduction

1.1 Problem Statement and Motivation

The Theory of Distributed Systems research group is currently working on modeling, verifying and analyzing problems arising in automated transit systems. The formal tools used include the standard techniques for distributed algorithms — invariants, simulations (levels of abstraction) and automaton composition, plus standard methods for reasoning about continuous processes — differential equations and mathematical analysis. The work so far suggests that these methods are capable of providing good results about safety and performance of automated transit systems.

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, [16]), which has developed a design for automating the operation of cars in several lanes of selected California highways. This Master of Engineering thesis is a case study of automated car maneuvers arising in the PATH project. We consider two cars traveling in a single lane at a high speed with small distance between them. The goal is for the second (follower) car to preserve safety, namely, that the two vehicles never collide at a relative velocity greater than a given bound, given arbitrary behavior of the first (leader) car.

The system is hybrid in that it involves both discrete and continuous behavior:

discrete behavior appears in the discrete actions of the controllers, points of collision, 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. The problem is further complicated by the presence of delays and uncertainties in the behavior of sensors, brakes and controllers.

The goals of this project are to model this system using Hybrid Automata [12], [13], and to derive and prove necessary and sufficient conditions that a controller of the follower car must satisfy in order to guarantee the safety requirement regardless of the behavior of the leading vehicle. In [4], a proof of such a safety property is outlined, for the specific vehicle maneuver given in that paper. The key to the proof turns out to be that the given maneuver always ensures that either (a) the vehicles are sufficiently far apart that the second vehicle can slow down sufficiently before hitting the first vehicle, or (b) the relative speeds of the two vehicles are already close enough.

Although the outline [4] gives the key ideas, from our point of view, it is incomplete as a safety verification. In particular, Frankel et al. do not include a complete model of all system components — the discrete components are not modeled — and do 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 (success, time optimality). 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. In this thesis, we model the whole system, including delays and uncertainties, and reason about it in a modular fashion, so that the proofs and the approach could be reused in other problems.

1.2 Related Work

In [12], 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, invariant assertions, levels of abstraction, and mathematical analysis for reasoning about continuous behavior. Lynch et al. have developed methods of incorporating standard analysis techniques into automaton-based proofs.

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],[17],[18]. Recently, some more complex systems have been modeled and analyzed with the same approach: Livadas used similar methods in modeling automated vehicle protection subsystems, as used in the Raytheon Personal Rapid Transit project (PRT 2000) [6]; and Lygeros and Lynch modeled and analyzed the Traffic Alert and Collision Avoidance System (TCAS) conflict resolution strategies [9].

Lygeros and Lynch [10] have also worked on a problem similar to the one presented in this thesis. The authors, using a similar approach, modeled a system comprised of a string of vehicles moving along a single track and proved safety requirements of such a system. However, their model involved an ideal system with no delays or uncertainties. In this thesis, these complications are incorporated into the model, but the problem is simplified by dealing only with 2 vehicles, and only worrying about the safety of the first collision. In future work, it will be interesting to extend this model to handle multiple vehicles. An alternative approach to proving safety for a specific vehicle maneuver, based on game theory, is presented in [7],[8].

A representative collection of prior work in the modeling and verification of hybrid systems is available in the proceedings of the workshops on hybrid systems [1],[3],[5],[15]. 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 [14],[2]; these differ from

ours primarily in placing less emphasis on issues of external behavior, composition and abstraction.

1.3 Our Approach

The approach of this research project is to formally model the entire system, including the two vehicles and their sensors and controllers. This way, communication delays, and delay and uncertainty in applying the control commands are included in the model of the system. We use Hybrid Automata, described in [12],[13] as a framework for modeling and reasoning about hybrid systems.

A parameterized safety criterion is formulated in terms of the model. The model and the safety requirement are made very general, so that they can reflect a variety of situations. This approach allows the later reuse of the models and proofs in other problems involving automated vehicles.

Necessary and sufficient safety conditions on controllers are devised and proved for the simplest case (namely, the no delays and no uncertainties case). Then, these results are gradually generalized, using composition and simulation relations, to increasingly complicated cases, until results are obtained for a realistic model with both delays and uncertainties. All the proofs are modular in that they consist of several lemmas and theorems, some of which could be reused in similar problems. More importantly, the approach of starting with the simplest case and then getting to the more complicated ones using simulation relations, should prove very useful. It allows the use of levels of abstraction to reduce the complexity of the problem.

1.4 Contributions of this Research

In this case study, we apply the hybrid automaton model and its associated proof methods to the task of describing and verifying safety for the PATH car maneuvers. This is a relatively complex and realistic example, which has practical implications. 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 different vehicle maneuvers. Our model should also be usable for proving other properties, such as success and time optimality. The system and its proofs should admit decomposition into separate parts, as far as possible, and should be easy to extend.

The contributions of this research are:

- Definition of a reusable model of the automated cars, plus their controllers and sensors, which incorporates delays and uncertainties directly.
- Derivation and proof of necessary and sufficient conditions for satisfying the safety requirement of the cars.
- A demonstration of the power of hybrid automata and its associated proof methods for reasoning about interesting hybrid systems.
- A demonstration of the use of abstraction levels as a means of handling complexity for hybrid systems.

Chapter 2

Hybrid Automata Model

The Hybrid Automata model presented in [12],[13] is capable of describing both continuous and discrete behavior. The model allows communication among components using both shared variables and shared actions. Several HA techniques make them particularly useful in modeling and reasoning about hybrid systems. These include composition, which allows the formation of complex automata from simple building blocks; implementation relations, which make it easy to use levels of abstraction when modeling complex systems; and invariant assertions, which describe the non-changing properties of the system.

For a complete description of the hybrid automata model, its associated methods, and proofs of all HA theorems stated below, please refer to the *Hybrid I/O Automata* paper [12].

2.1 Hybrid Automata

A *state* of a HA 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.

A HA A consists of:

- Two disjoint sets of *external and internal* variables. A valuation of these sets

constitutes a state s of A.

- Two disjoint sets of *external and internal* discrete actions. We assume that there is a special *external, environment action* e, which represents the occurrence of a discrete transition outside the system.
- A nonempty set of *start states*.
- A set of *discrete transition*, i.e. (state, action, state) triples, satisfying

  D: $\forall s : (s, e, s)$ is a valid discrete transition.

- A set T of trajectories w over the variables of A, satisfying
  - T1 (existence of point trajectories): $\forall s$, the point trajectory p that maps 0 to s is in T,
  - T2 (closure under subintervals): $\forall w \in T$ and for all left-closed subintervals J of $dom(w)$: (w restricted to J) $\in T$, and
  - T3 (completeness): $\forall w$ on a left-closed interval J with left endpoint equal to 0 : $(\forall t \in J : (w \text{ restricted to } [0, t)) \in T) \Rightarrow w \in T$.

Axioms T1-3 state some natural conditions on the set of trajectories: existence of point trajectories, closure under subintervals, and the fact that w is a full trajectory if and only if all its prefixes are valid trajectories. (Actually, axiom T3 does not say "if and only if", but the missing direction follows easily from T2.)

2.2 Hybrid Executions and Hybrid Traces

A *hybrid execution fragment* of A is a finite or infinite alternating sequence of trajectories and actions, ending with a trajectory, if it is a finite execution fragment. 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. The *time duration* of a Hybrid execution is the sum of the durations of its trajectories. Hybrid execution fragments are called *admissible* if their time duration is 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. The hybrid traces of a HA A that arise from all the finite and admissible hybrid executions of A describe its external behavior.

2.3 Simulation Relations

HAs A_1 and A_2 are *comparable* if they have the same external interface, i.e., the same external variables and actions. If A_1 and A_2 are comparable then A_1 *implements* A_2, denoted $A_1 \leq A_2$, if the set of hybrid traces of A_1 is a subset of the set of hybrid traces of A_2. Intuitively, this means that any external behavior of A_1 is allowed by A_2, A_1 being more restrictive.

Let A and B be comparable HAs. A *simulation* from A to B is a relation R from states of A to states of B, satisfying the following conditions for states s_A and s_B of A and B, respectively:

- If s_A is a start state of A, then there exists a start state s_B of B, such that $s_A R s_B$.
- 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 a hybrid trace h that is identical to that of (s_A, a, s'_A), and ending with a state s'_B such that $s'_A R s'_B$.
- If w 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 a hybrid trace that is identical to that of w, and ending with a state s'_B, such that $s'_A R s'_B$.

The following theorem describes how one can prove that one HA implements another HA. It's proof may be found in [12].

Theorem 2.1 *If A_1 and A_2 are comparable HAs and there is a simulation from A_1 to A_2, then A_1 implements A_2.*

2.4 Composition

Another HA technique for reducing complexity is *composition.*

HAs A and B are *compatible* if

1. Initial conditions of A and B are consistent. Formally, there exists a valuation s for $V = V_A \cup V_B$, where V_A and V_B are the sets of variables of A and B, respectively, such that the valuation of variables of A and B in s comprise start states of A and B, respectively.

2. Internal actions of A are disjoint from actions of B, and internal variables of A are disjoint from variables of B. Similarly for B and A.

If A and B are compatible then their *composition* $C = A||B$ is defined as follows:

1. External and internal variables of C are the union of external and internal variables, respectively, of A and B.

2. External and internal actions of C are the union of external and internal actions, respectively, of A and B.

3. Start states are states of C that satisfy the initial conditions of both A and B.

4. Discrete transitions and trajectories are the union of the corresponding components of A and B.

We state without proof that C is in fact a HA.

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. This result is also presented here without proof.

Theorem 2.2 *Suppose A_1, A_2, B be HAs such that $A_1 \leq A_2$, and each of A_1 and A_2 is compatible with B. Then $A_1 \parallel B \leq A_2 \parallel B$.*

2.5 Hiding

Two natural hiding operations are defined on any HA A.

Let E, H designate the external and internal actions of a HA, respectively; W, X designate the external and internal variables of a HA, respectively; and e — the environmental action.

1. If $E \subseteq E_A - \{e\}$, then **ActHide(E, A)** is the HA B that is equal to A except that $E_B = E_A - E$ and $H_B = H_A \cup E$.

2. If $W \subseteq W_A$, then **VarHide(W, A)** is the HA B that is equal to A except that $W_B = W_A - W$ and $X_B = X_A \cup W$.

Theorem 2.3 *Let $E \subseteq E_A - \{e\}$ and $W \subseteq W_A$. Then ActHide(E, A), VarHide(W, A) are HAs.*

2.6 Modeling Conventions in This Thesis

A's *visible* behavior is completely described by changes of its external variables. Here, we subdivide the set of *external* variables into two disjoint sets of *input* and *output* variables. This is done for notational convenience only, and does not change automata properties.

In [12], two models are defined: Hybrid Automata and Hybrid I/O Automata. Hybrid I/O Automata are an extension of Hybrid Automata, in that they differentiate between input and output actions. They are also more restrictive because they have more axioms associated with them. In this thesis we use the Hybrid Automata model exclusively.

In all the automata defined in this work we assume, without explicitly specifying, the following:

1. An external environmental action e. It is always enabled (can happen at any time), and it does not change any of the state variables.

2. All point trajectories are included. Only "non-trivial" (not point) trajectories are specified explicitly.

Item (1) satisfies axiom D, and item (2) satisfies axiom $T1$ of HAs. Therefore, we only need to prove that an automaton satisfies axioms $T2$ and $T3$ to claim that it is, in fact, a HA. $T2$ requires closure under subintervals; $T3$ requires completeness. In order to satisfy $T3$, either (1) trajectories are never *required* to stop; or (2) if a trajectory is required to stop, its time domain has to be a *closed* interval. It is easy to see that $T3$ holds in either case.

Chapter 3

Math Preliminaries

3.1 Non-Negativity Theorems

The following theorem gives simple conditions that a differentiable function must satisfy in order to be positive everywhere on a given interval.

Theorem 3.1 *Given a continuous function f on an interval $[a, b]$, if*

1. *$f(a) \geq c$, and*

2. *for all $x \in [a, b)$, if $f(x) \leq c$ then $f'(x)$ exists and $f'(x) \geq 0$,*

then for all $x \in [a, b]$, $f(x) \geq c$.

Proof: We prove this by contradiction. Suppose there exists $b' \in [a, b]$ such that $f(b') < c$. Since $f(a) \geq c$ and f is continuous, there exists $a' \in [a, b')$ such that $f(a') = c$ and for all $x \in (a', b']$, $f(x) < c$ by the Intermediate Value Theorem.

By the Mean Value Theorem, there exists $x \in (a', b')$ such that $f'(x) = \frac{f(b') - f(a')}{b' - a'}$. Since $f(x) < c$, it follows by assumption 2 that $f'(x) \geq 0$. Thus, since $b' > a'$, it follows that $f(b') \geq f(a')$. But since $f(a') = c$, it follows that $f(b') \geq c$, which is a contradiction.

■

We now prove a similar theorem for functions that have right and left derivatives that are not necessarily equal to each other. The right and left derivatives of a continuous function f are defined as follows:

$$f'(x^+) = \lim_{t \to x^+} \frac{f(t) - f(x)}{t - x} \text{ (right derivative), and}$$
$$f'(x^-) = \lim_{t \to x^-} \frac{f(t) - f(x)}{t - x} \text{ (left derivative).}$$

First, we prove a helpful lemma about right and left derivatives of a continuous function at its local maxima and minima.

Lemma 3.2 *Let f be a continuous function defined on $[a, b]$, whose both right and left derivatives are defined on (a, b). Then,*

1. *if $x \in (a, b)$ is a local maximum, then $f'(x^+) \leq 0$ and $f'(x^-) \geq 0$;*
2. *if $x \in (a, b)$ is a local minimum, then $f(x^+) \geq 0$, and $f(x^-) \leq 0$.*

Proof: Let x be a local maximum. Then there exists $\delta > 0$ such that $f(q) \leq f(x)$ for all q such that $|x - q| < \delta$. Choose such δ so that

$$a < x - \delta < x < x + \delta < b.$$

Choose t such that $x - \delta < t < x$. Then

$$\frac{f(t) - f(x)}{t - x} \geq 0.$$

Letting $t \to x^-$, we see that $f'(x^-) \geq 0$.

Now choose t such that $x < t < x + \delta$. Then

$$\frac{f(t) - f(x)}{t - x} \leq 0,$$

which shows that $f'(x^+) \leq 0$.

The statement about local minima can be proven analogously.

■

Using Lemma 3.2 we prove an analog of the Mean Value Theorem for functions whose right and left derivatives are not necessarily equal.

Lemma 3.3 *Let f be a continuous function defined on $[a,b]$ whose right and left derivatives are defined on (a,b). Then there exists $x \in (a,b)$ such that either*

$$\frac{f(b)-f(a)}{b-a} \geq f'(x^+), \text{ or}$$

$$\frac{f(b)-f(a)}{b-a} \geq f'(x^-).$$

Also, there exists $x \in (a,b)$ such that either

$$\frac{f(b)-f(a)}{b-a} \leq f'(x^+), \text{ or}$$

$$\frac{f(b)-f(a)}{b-a} \leq f'(x^-).$$

Proof: Put $h(t) = (f(b)-f(a))t - (b-a)f(t)$. Then h is continuous on $[a,b]$, has right and left derivatives on (a,b), and

$$h(a) = af(b) - bf(a) = h(b).$$

To prove the first half of the lemma it suffices to show that either $h'(x^+) \geq 0$ or $h'(x^-) \geq 0$ for some $x \in (a,b)$, since

$$h'(x^+) = f(b) - f(a) - (b-a)f'(x^+) \text{ and } h'(x^+) = f(b) - f(a) - (b-a)f'(x^+).$$

Case 1: h is constant. Then, $h'(x^+) = h'(x^-) = h'(x) = 0$, so the condition holds for all x.

Case 2: $h(t) > h(a)$ for some $t \in (a,b)$. By continuity of h, there exists $x \in (a,b)$ which is a local maximum in (a,b). Lemma 3.2 shows that $h'(x^-) \geq 0$.

Case 3: $h(t) < h(a)$ for some $t \in (a,b)$. By continuity of h, there exists $x \in (a,b)$ which is a local minimum in (a,b). Lemma 3.2 shows that $h'(x^+) \geq 0$.

The second part of the lemma is proved analogously.

■

Finally, we are able to prove a theorem similar to our first non-negativity theorem (Theorem 3.1), but for functions with unequal right and left derivatives.

Theorem 3.4 *Let f be a continuous function defined on $[a,b]$. If*

1. $f(a) \geq c$;

2. *for all* $x \in [a, b)$ *and* $f(x) \leq c$, $f'(x^+)$ *and* $f'(x^-)$ *exist, with* $f'(x^+) \geq 0$ *and* $f'(x^-) \geq 0$,

then for all $x \in [a, b]$, $f(x) \geq c$.

Proof: Again, we use proof by contradiction. Suppose there exists $b' \in [a, b]$ such that $f(b') < c$. Since $f(a) \geq c$ and f is continuous on $[a, b]$, there exists $a' \in [a, b')$ such that $f(a') = c$ and for all $x \in (a', b']$, $f(x) < c$, by the Intermediate Value Theorem.

By Lemma 3.3 we have that either

$$\frac{f(b') - f(a')}{b' - a'} \geq f'(x'^+) \text{ or } \frac{f(b') - f(a')}{b' - a'} \geq f'(x'^-)$$

for some $x' \in (a', b')$. Since $f(x) < c$ for all $x \in (a', b']$, both derivatives have to be non-negative by property 2, so in either case we have $\frac{f(b')-f(a')}{b'-a'} \geq 0$. Also, $b' > a'$, so we get $f(b') \geq f(a')$. But $f(a') = c$, so $f(b') \geq c$, which contradicts our assumption. ■

3.2 Non-Increasing Functions

The following lemma gives simple conditions for a function with unequal right and left derivatives to be non-increasing.

Lemma 3.5 *Let* f *be a continuous function defined on* $[a, b]$ *whose right and left derivatives are defined on (a,b). Then if for all* $x \in (a, b)$, $f'(x^+) \leq 0$ *and* $f'(x^-) \leq 0$, *then* f *is a non-increasing function.*

Proof: This follows directly from Lemma 3.3. We have that for any $a' < b'$ in $[a, b]$, there exists $x \in [a', b']$ such that either

$$\frac{f(b') - f(a')}{b' - a'} \leq f'(x^+) \text{ or } \frac{f(b') - f(a')}{b' - a'} \leq f'(x^-).$$

But since $b' > a'$ and also $f'(x^+) \leq 0$ and $f'(x^-) \leq 0$, we get $f(b') \leq f(a')$. Since a', b' were chosen arbitrarily, it follows that f is non-increasing.

■

3.3 Derivatives of The *max* Function

We prove a useful theorem about the right and left derivatives of the *max* function.

Theorem 3.6 *Let f and g be differentiable functions, and $m(x) = max(f(x), g(x))$. Then,*

1. *the right derivative of $m(x)$ exists and equals the right derivative of either f or g;*

2. *the left derivative of $m(x)$ exists and equals the left derivative of either f or g.*

Proof: We start with the right derivative. First suppose that there exists $\delta > 0$, such that for all $t \in [x, x+\delta)$, $m(t) = f(t)$. Then,

$$m'(x^+) = \lim_{t \to x^+} \frac{m(t) - m(x)}{t - x} = \lim_{t \to x^+} \frac{f(t) - f(x)}{t - x} = f'(x^+),$$

as needed. Analogously for $m(t) = g(t)$.

Alternatively, suppose that no such δ exists. This means that for all $\delta > 0$, there exists a point $t_1 \in [x, x+\delta)$, such that $m(t_1) = f(t_1) > g(t_1)$, and there also exists a point $t_2 \in [x, x+\delta)$, such that $m(t_2) = g(t_2) > f(t_2)$.

Then the following two statements must be true:

1. $f(x) = g(x) = p = m(x)$, where p is some real number.

   Proof: Suppose this is not so. Let $f(x) > g(x)$, without loss of generality. Then, by continuity of f and g, there exists a neighborhood of x in which for all t, $f(t) > g(t)$. But this contradicts our original assumption that there does not exist a δ-neighborhood of x in which $f(t) > g(t)$ for all $t \in [x, x+\delta)$. ■

2. $f'(x^+) = g'(x^+) = q$, where q is some real number.

   Proof: Suppose this is not so. Then $f'(x^+) = q_1$ and $g'(x^+) = q_2$, where $q_1 \neq q_2$. Without loss of generality, let's assume that $q_1 > q_2$. From the definition of the derivative, we know that for all $\epsilon > 0$, there exist $\delta_1, \delta_2 > 0$, such that $\forall t' \in (x, x+\delta_1)$ and $\forall t'' \in (x, x+\delta_2)$,

$$\left|\frac{f(t')-f(x)}{t'-x} - q_1\right| < \epsilon \text{ and } \left|\frac{g(t'')-g(x)}{t''-x} - q_2\right| < \epsilon.$$

   Let's pick $\epsilon < \frac{1}{2}(q_1 - q_2)$, and let $\delta = min(\delta_1, \delta_2)$. Then, using the result from statement 1, $\forall t \in (x, x+\delta)$,

$$\left|\frac{f(t)-p}{t-x} - q_1\right| < \epsilon \text{ and } \left|\frac{g(t)-p}{t-x} - q_2\right| < \epsilon.$$

   From these inequalities, by choice of ϵ, and using the fact that $t > x$, it follows that

$$\begin{aligned} \frac{f(t)-p}{t-x} &> \frac{g(t)-p}{t-x} \\ f(t)-p &> g(t)-p \\ f(t) &> g(t) \end{aligned}$$

   This means that there exists a δ-neighborhood of x in which $f(t) > g(t)$, which contradicts our original assumption.

■

From statement 2 we have that for all $\epsilon > 0$, there exists $\delta > 0$ such that $\forall t \in (x, x+\delta_1)$ the following hold:

$$\left|\frac{f(t)-f(x)}{t-x} - q\right| < \epsilon \text{ and } \left|\frac{g(t)-g(x)}{t-x} - q\right| < \epsilon.$$

Then $\forall t \in (x, x+\delta)$,

$$\frac{m(t)-m(x)}{t-x} - q = \begin{cases} \left|\frac{f(t)-p}{t-x} - q\right| < \epsilon & \text{if } f(t) \geq g(t) \\ \left|\frac{g(t)-p}{t-x} - q\right| < \epsilon & \text{if } f(t) < g(t). \end{cases}$$

Thus, in both cases, $\left|\frac{m(t)-m(x)}{t-x} - q\right| < \epsilon$. This means that

$$m'(x^+) = \lim_{t \to x} \frac{m(t) - m(x)}{t - x} = q,$$

which is the same as the right derivative of both f and g at x.

The left derivative part is proven analogously. ■

Chapter 4

System Model

We consider two vehicles, moving along a single track. While the behavior of the leading vehicle is arbitrary, the second vehicle's controller must make sure that no "bad" collisions occur. "Bad" collisions are collisions at a high relative speed. This is called the *Safety* requirement for the second controller. This *Safety* requirement is general for all vehicle 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 vehicle maneuver must implement our *safe* controller in order to be correct. This should be useful in formally proving correctness of complicated algorithms.

4.1 Vehicles

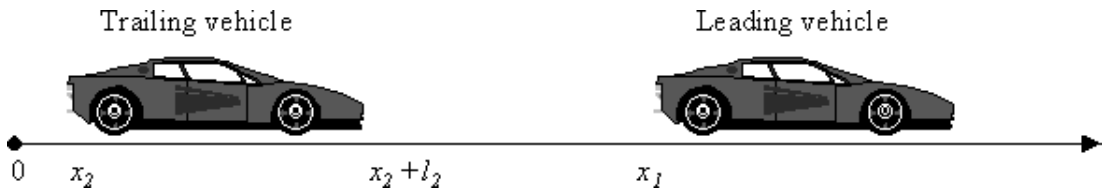

Figure 4-1: Vehicles

Constants:

l_i, the length of the vehicle

$a_{min} \in \mathsf{R}^-$, the maximum emergency deceleration

Variables:

Input: $acc_i \in \mathsf{R}$, initially arbitrary

Output: $x_i \in \mathsf{R}^{\geq 0}$, initially $x_2 = 0$ and x_1 is arbitrary, subject to $x_1 \geq x_2 + l_2$

$\dot{x}_i \in \mathsf{R}^{\geq 0}$, initially arbitrary

$\ddot{x}_i \in \mathsf{R}$, initially arbitrary

now, initially 0

collided, Boolean, initially *false*

Actions:

Internal: *collide*

Pre: $x_1 = x_2 + l_2$

$collided = false$

Effect: $\ddot{x}_i :=$ arbitrary value, subject to $\ddot{x}_i \geq a_{min}$

$\dot{x}_i :=$ arbitrary value

$collided := true$

Trajectories:

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

1. *collided* is unchanged in w
2. $\ddot{x}_i$ is integrable in w
3. for all $t \in I$ the following hold:
   - 3.1. If $collided = false$ in w then
     $w(t).\ddot{x}_i = max(w(t).acc_i, a_{min})$
     else, $w(t).\ddot{x}_i \geq a_{min}$
   - 3.2. $w(t).\dot{x}_i = w(0).\dot{x}_i + \int_0^t w(u).\ddot{x}_i du$
   - 3.3. $w(t).x_i = w(0).x_i + \int_0^t w(u).\dot{x}_i du$
   - 3.4. $w(t).x_2 + l_2 \leq w(t).x_1$
   - 3.5. $w(t).now = w(0).now + t$
   - 3.6. If $w(t).x_1 = w(t).x_2 + l_2$ and t is not the right endpoint of I then
     $collided = true$.

Figure 4-2: The *Vehicles* Hybrid Automaton

We compose our system of a piece modeling the physical vehicles, and two pieces modeling the controllers of each vehicle. Each piece is modeled by a hybrid automaton. The real world piece is called *Vehicles*.

Automaton 1(Vehicles): The automaton represents two vehicles, named 1 and 2, where vehicle 1 precedes vehicle 2 on a single track. Positions on the track are labeled with nonnegative reals, starting with 0 as a designated starting point, as shown in Figure 4-1. The formal HA model for this automaton is given in Figure 4-2. We assume that $i \in 1, 2$ throughout the model.

Constants

- l_i is the length of the i-th vehicle.
- $a_{min} < 0$ is the maximum deceleration rate for the vehicles. We assume here that all vehicles have identical breaking capabilities.

Variables

- acc_i denotes the acceleration commanded by the controller. Note that it can differ from $\ddot{x}_i$, which is the actual acceleration of the vehicle, due to delays and/or uncertainties.
- x_i, $\dot{x}_i$, and $\ddot{x}_i$ model the actual position of the vehicle's rear, its velocity and acceleration data. The dots are used as a syntactic device only, and do not impose differential relationships.
- *now* models the current time. While it is not necessary for modeling the system, it will be used later in stating some of the invariant assertions. Initially, *now* $= 0$.
- *collided* keeps track of the first occurrence of a collision; it will be used in our statement of the correctness property — in this work we only want to guarantee safety for zero or one collisions, the multiple collisions case is not handled.

Actions The action *collide* occurs when the vehicles touch each other for the first time. The vehicles touch when the position of the rear of the leading vehicle, x_1, equals the position of the front of the trailing vehicle, $x_2 + l_2$. The effect is that both the vehicle's acceleration and velocity assume arbitrary values. After the collision, the vehicle's acceleration, $\ddot{x}_i$, is decoupled from what is commanded by the controller, while velocity and position are still obtained by integrating the acceleration, $\ddot{x}_i$.

Trajectories The first condition (1) states that *collided* can only be changed by discrete actions. Condition (2) requires the actual acceleration of the vehicles to be integrable, so that velocity and position can be derived from it. (3) gives conditions on all states of a trajectory. Condition (3.1) ensures that the vehicle implements the controller's acceleration (taking care not to go below a_{min}), before the first collision occurs. Conditions (3.2) and (3.3) give differential relationships between the actual acceleration, velocity and position of the vehicle at all times. (3.4) does not allow vehicles to bypass each other, which is realistic assuming that the vehicles move only in a single lane. (3.5) assigns the variable *now*, and (3.6) makes sure that when the vehicles collide, then either a) it is the right endpoint of the trajectory, and the *collide* action will be scheduled (this happens for the first collision), or b) it is after the first collision, and *collided* already is *true*.

HA By discussion in Section 2.6, we only need to show that axioms $T2$ and $T3$ are satisfied. Since the duration of trajectories is not restricted, "sub"-trajectories are always valid, so $T2$ is satisfied. The only time trajectories are *required* to stop is when a *collide* action has to occur. But by trajectory condition (3.6), we allow these trajectories to be closed, so $T3$ is also satisfied. Therefore, this automaton is an *HA*.

Properties From this definition, several useful properties of all reachable state of the *Vehicles* automaton can be deduced:

1. $\ddot{x}_i \geq a_{min}$, by trajectory condition (3);

2. velocity and position are integrals of $\ddot{x}_i$, except at the time of collision, by trajectory conditions (4) and (5);

3. $x_2 + l_2 \leq x_1$, meaning that vehicles never bypass each other, by trajectory condition (6).

4.2 Controllers

Variables:

Input:	$\ddot{x}_i \in \mathsf{R}$, $i \in \{1,2\}$
	$\dot{x}_i \in \mathsf{R}^{\geq 0}$, $i \in \{1,2\}$
	$x_i \in \mathsf{R}^{\geq 0}$, $i \in \{1,2\}$
Output:	acc_j, initially arbitrary, where $acc_j \geq a_{min}$
Internal:	$\dot{x}_{intj} \in \mathsf{R}^{\geq 0}$, initially $\dot{x}_{intj} = \dot{x}_j$
	$x_{intj} \in \mathsf{R}^{\geq 0}$, initially $x_{intj} = x_j$

Trajectories:

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

1. acc_j is an integrable function
2. for all $t \in I$, in state $w(t)$
   - 2.1. $\dot{x}_{intj} = w(0).\dot{x}_{intj} + \int_0^t w(u).acc_j du$
   - 2.2. $x_{intj} = w(0).x_{intj} + \int_0^t w(u).\dot{x}_{intj} du$
   - 2.3. If $\dot{x}_j \leq 0$ then $acc_j \geq 0$

Figure 4-3: $Controller_j$ Hybrid Automaton, $j \in \{1,2\}$

We now define the controller automaton.

Automaton 2($Controller_j$, $j \in \{1,2\}$): This automaton inputs the current position, velocity and acceleration data of the vehicle (from the *Vehicles* automaton) and outputs new acceleration settings. It is an arbitrary hybrid automaton with the given interface, and it is restricted only by the physical limitations of the vehicles. It does not have any discrete actions. The formal model for this automaton, where j is either 1 or 2, is given in Figure 4-3.

Variables $Controller_j$ receives the real position and velocity data of both vehicles via sensors, which we model by inputing the data from the *Vehicles* automaton. Based on these inputs, the $Controller_j$ decides on a safe acceleration setting and outputs it to *Vehicles*. The internal velocity and position variables ($\dot{x}_{intj}$ and x_{intj}) are approximations to the real data of *Vehicles*, calculated based on the acceleration the $Controller_j$ has commanded. This data is obtained by integrating the acceleration requests of the controller. Since we have not included any delays or uncertainties yet, these variables should correspond exactly to the actual position and velocity of the vehicle, so that $x_{intj} = x_j$ and $\dot{x}_{intj} = \dot{x}_j$. However, when we add uncertainty and delay into our model, the internal variables will be different from the input variables (which are received from sensors); the internal variables will use input variables and account for delays and uncertainties to get better estimates of the actual data.

Trajectories Condition (1) requires that the commanded acceleration be integrable twice, so that the integrals for velocity and position are well defined. Conditions (2.1) and (2.2) define internal velocity and position data to be integrals of commanded acceleration. Finally, condition (2.3) guarantees that once the vehicle has non-positive velocity, the acceleration must be non-negative, keeping the vehicles from going backwards.

HA No restrictions on either the duration of a trajectory, or stopping trajectories are places, so axioms $T2$ and $T3$ are satisfied. Thus, by discussion in Section 2.6, $Controller_j$ is an *HA*.

We model the whole system by composing the *Vehicles* automaton with 2 controllers. These controllers must be implementations of $Controller_j$. Thus, the composed system is a "function" of the given controllers and the given implementation of the *Vehicles* automaton.

Automaton 3(Controlled-Vehicles): *Controlled-Vehicles*(V, A_1, A_2) = *Vehicles* $\| A_1 \| A_2$, where V is an implementation of the *Vehicles* automaton, A_1 is an implementation of $Controller_1$, and A_2 is an implementation of $Controller_2$. The automata

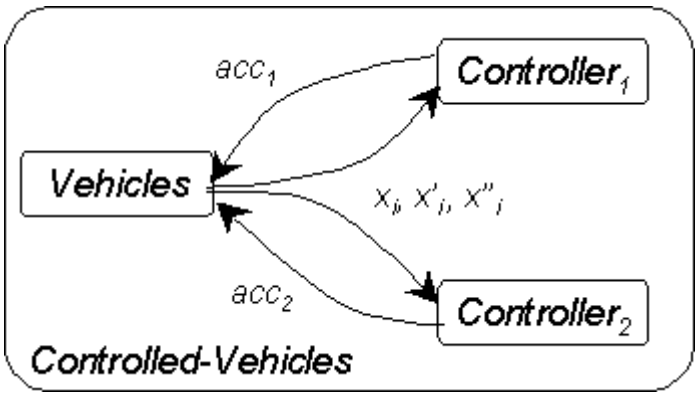

Figure 4-4: *Controlled-Vehicles* Model

are composed using hybrid automata composition rules, resulting in another *HA* automaton. The system models our vehicle system with each vehicle having its own controller. Figure 4-4 shows *Controlled-Vehicles*(*Vehicles*, $Controller_1$, $Controller_2$), by showing the pieces it consists of and the interfaces between them.

4.3 Safety Condition

We define a safety condition for the states of *Controlled-Vehicles*. The safety condition guarantees that if the vehicles 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:

Definition 1(Safety): If $x_1 = x_2 + l_2$ and $collided = false$, then $\dot{x}_2 \leq \dot{x}_1 + v_{allow}$.

We define a new automaton, *Safe-Vehicles*, to serve as a correctness specification.

Automaton 4(Safe-Vehicles): This automaton is exactly the same as *Vehicles* with an added restriction on trajectories: all states are required to satisfy the *Safety* condition. Since this restriction does not violate axioms $T2$ and $T3$, the *Safe-Vehicles* automaton is still a valid *HA*.

Given an implementation V of *Vehicles*, we want to design an implementation A_2 of $Controller_2$ such that for any implementation A_1 of $Controller_1$, the system *Controlled-Vehicles*(V, A_1, A_2) implements the *Safe-Vehicles* automaton. Then we

can say that it satisfies the *Safety* condition. That is enough to ensure that the *Safety* condition of the specification carries over to the implementation. Note that although the *Controlled-Vehicles* automaton includes controllers, it can still implement the *Safe-Vehicles* automaton, since they will have the same external interface (position, velocity, acceleration data, the *now* variable, and the *collided* flag) and the controllers only affect the acceleration settings.

Definition 2(Correctness): Given an implementation V of *Vehicles*, an implementation A_2 of $Controller_2$ is **correct** for V if and only if for every implementation A_1 of $Controller_1$, *Controlled-Vehicles*(V, A_1,A_2) system implements *Safe-Vehicles*.

Chapter 5

Safety In The Ideal Case

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, which include delays and uncertainties, via simulation mappings. By ideal setting we mean that there are no delays and/or uncertainties in either the sensors' data or the controller's directives. In the next chapters we make the model more realistic by relaxing these restrictions.

5.1 Problem Statement

We want to give conditions on an implementation of $Controller_2$ that are both necessary and sufficient to satisfy the *correctness* property of Definition 2. In the next section we present such conditions by showing an implementation of $Controller_2$, called C_2, which guarantees *correctness*. Then, we show that the conditions are sufficient by proving that this conroller is *correct*. Finally, we give slightly less restrictive conditions on the controller and prove that these conditions are necessary.

5.2 The Model

In any state of *Vehicles*, define

$$safe\text{-}measure = \max(x_1 - (x_2 + l_2) + \frac{(\dot{x}_2)^2 - (\dot{x}_1)^2 - (v_{allow})^2}{2a_{min}},$$
$$\dot{x}_1 + v_{allow} - \dot{x}_2).$$

We are not interested in the actual value of *safe-measure*, but only in whether or not it is negative. If it is nonnegative, it means that either (a) the distance between the two vehicles is great enough to allow vehicle 2 to stop before hitting vehicle 1, even if vehicle 1 decelerates at its fastest possible rate, or (b) the relative velocities of the two vehicles are already close enough. Thus, nonnegative *safe-measure* gives us the boundaries of the safe region for the second vehicle.

Variables:

Input, Output, Internal: same as in $Controller_2$

Trajectories:

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

1. it satisfies condition 1 of $Controller_2$, plus
2. for all $t \in I$,
   - 2.1-2.3 of $Controller_2$ are satisfied
   - 2.4. if $collided = false$ and $safe\text{-}measure \leq 0$ then $w(t).acc_2 = a_{min}$

Figure 5-1: C_2 Hybrid Automaton

Automaton 5(C_2): This automaton is exactly like $Controller_2$, with one extra restriction on its trajectories. The formal model is given in Figure 5-1. Condition (2.4) ensures that if the position and velocity parameters are on the boundary defined by *safe-measure*, then C_2 commands maximum deceleration, by setting $acc_2 = a_{min}$. In this ideal (no delays, no uncertainties) setting, vehicle 2 will execute the command exactly, because $\ddot{x}_2 = acc_2$, and so vehicle 2 is decelerating as fast as possible. Since this restriction does not violate axioms $T2$ and $T3$, C_2 is a valid HA. We claim that this restriction is sufficient to guarantee *correctness*.

5.3 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).

Definition 3(Predicate S): If *collided* = *false* then *safe-measure* ≥ 0.

The above definition says that before the first collision occurs, *safe-measure* is non-negative. We will later prove that non-negativity of *safe-measure* guarantees *safety.*

We construct a new automaton, *Init-Vehicles*, which is exactly like *Vehicles*, except that all its start states are restricted to satisfy Predicate S. This guarantees that the system is *safe* initially.

Automaton 6(Init-Vehicles): Exactly like *Vehicles*, but guarantees *safety* initially. The formal model is shown in Figure 5-2, with the new restrictions in bold. Again, it is a valid *HA*, since the modifications do not violate axioms $T2-3$.

C_2 is designed to guarantee explicitly that if S is ever violated, or is even in danger of being violated (because equality holds), vehicle 2 is decelerating as fast as possible. We claim that this strategy is sufficient to guarantee that S is always true. To prove this, we will use the *Non-negativity Theorem* 3.4, which states that all functions satisfying certain conditions must be non-negative.

Lemma 5.1 *Predicate S is true in every reachable state of Controlled-Vehicles(Init-Vehicles, A_1, C_2), where A_1 is any implementation of Controller$_1$.*

Proof: By induction on the number of steps in a hybrid execution. Initially, the claim is true by the restriction on the initial states of *Init-Vehicles.*

The only discrete steps are *collide, e* and the internal steps of A_1. The latter two steps do not change any of the quantities involved. The effect of the *collide* step ensures that *collided* = *true* in the post-state, which makes S true vacuously.

Constants:

l_i, the length of the vehicle

$a_{min} \in \mathsf{R}^-$, the maximum emergency deceleration

Variables:

Input: $acc_i \in \mathsf{R}$, initially arbitrary

Output: $x_i \in \mathsf{R}^{\geq 0}$, initially $x_2 = 0$ and x_1 is arbitrary

$\dot{x}_i \in \mathsf{R}^{\geq 0}$, initially arbitrary

$\ddot{x}_i \in \mathsf{R}$, initially arbitrary

now, initially 0

collided, Boolean, initially *false*

initial state is subject to Predicate S

Actions:

Internal: *collide*

Pre: $x_1 = x_2 + l_2$

$collided = false$

Effect: $\ddot{x}_i :=$ arbitrary value

$\dot{x}_i :=$ arbitrary value

$collided := true$

Trajectories:

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

1. *collided* is unchanged in w
2. $\ddot{x}_i$ is integrable twice in w

for all $t \in I$ the following hold:

3. If $collided = false$ in w then
   $w(t).\ddot{x}_i = max(w(t).acc_i, a_{min})$
4. $w(t).\dot{x}_i = w(0).\dot{x}_i + \int_0^t w(u).\ddot{x}_i du$
5. $w(t).x_i = w(0).x_i + \int_0^t w(u).\dot{x}_i du$
6. $w(t).x_2 + l_2 \leq w(t).x_1$
7. $w(t).now = w(0).now + t$
8. If $w(t).x_1 = w(t).x_2 + l_2$ and t is not the right endpoint of I then
   $collided = true$.

Figure 5-2: The *Init-Vehicles* Hybrid Automaton

Now we consider a trajectory w whose domain is the interval $[0, T]$. Since a trajectory cannot change *collided*, and S is vacuously true if $collided = true$, we only need to consider the case where $collided = false$ throughout w. We may assume (by the induction hypothesis) that S is true in $w(0)$. We must show that S is true in $w(T)$. By definition of S, we may assume that $safe\text{-}measure \geq 0$ in state $w(0)$ and must show that this is also true in $w(T)$.

Here we will use the notation $f(t)$ to mean $w(t).f$, where f is defined in terms of state components of $w(t)$. Let $f(t) = x_1 - (x_2 + l_2) + \frac{(\dot{x}_2)^2 - (\dot{x}_1)^2 - (v_{allow})^2}{2a_{min}}$, $g(t) = \dot{x}_1 + v_{allow} - \dot{x}_2$. Then $s(t) = \max(f(t), g(t)) = safe\text{-}measure(t)$. We now use Theorem 3.4 to prove that if $s(0) \geq 0$, then $\forall t \geq 0, s(t) \geq 0$.

Claim 1. $s(t)$ is continuous.

Proof: By continuity of $f(t)$ and $g(t)$. ■

Claim 2. $s(0) \geq 0$.

Proof: Follows from the induction hypothesis. ■

Claim 3. For all t such that $s(t) \leq c$, it is the case that $s'(t^+)$ and $s'(t^-)$ exist, $s'(t^+) \geq 0$ and $s'(t^-) \geq 0$.

Proof: The right derivative of s equals the right derivative of either $f(t)$ or $g(t)$, and the same is true for the left derivative, by Theorem 3.6. We need to check that for all t such that $s(t) \leq 0$, we have $\dot{f}(t) \geq 0$ and $\dot{g}(t) \geq 0$. So, fix t such that $s(t) \leq 0$. Then, at t, we have:

$$\dot{f} = \dot{x}_1 - \dot{x}_2 + \frac{1}{2a_{min}}(2\dot{x}_2\ddot{x}_2 - 2\dot{x}_1\ddot{x}_1) = \dot{x}_1 - \dot{x}_2 + \frac{1}{a_{min}}(\dot{x}_2\ddot{x}_2 - \dot{x}_1\ddot{x}_1)$$

By definition of C_2 we have that since $s \leq 0$, $acc_2 = \ddot{x}_2 = a_{min}$. Also, by restriction

on *Init-Vehicles*, we have $\ddot{x}_1 \geq a_{min}$. Therefore,

$$\begin{aligned}
\dot{f} &= \dot{x}_1 - \dot{x}_2 + \frac{1}{a_{min}}(\dot{x}_2\ddot{x}_2 - \dot{x}_1\ddot{x}_1) \\
&= \dot{x}_1 - \dot{x}_2 + \dot{x}_2\frac{a_{min}}{a_{min}} - \dot{x}_1\frac{\ddot{x}_1}{a_{min}} \\
&= \dot{x}_1 + \dot{x}_1\frac{\ddot{x}_1}{-a_{min}} \\
&\geq \dot{x}_1 + \dot{x}_1\frac{a_{min}}{-a_{min}} \\
&= 0.
\end{aligned}$$

Now let's do the same for $\dot{g}$:

$$\dot{g} = \ddot{x}_1 - \ddot{x}_2 = \ddot{x}_1 - a_{min} \geq a_{min} - a_{min} = 0.$$

This proves Claim 3. ■

From Claims 1, 2 and 3, s satisfies the conditions of the *Non-negativity Theorem* and, therefore, by Theorem 3.4, $\forall t\ s(t) \geq 0$.

This suffices. ■

As a simple consequence of Lemma 5.1, we prove the *safety* property.

Lemma 5.2 *In any reachable state of Controlled-Vehicles(Init-Vehicles, A_1, C_2), where A_1 is any implementation of Controller$_1$, if $x_1 = x_2 + l_2$ and collided = false, then $\dot{x}_2 \leq \dot{x}_1 + v_{allow}$.*

Proof: Initially, S is true by the restriction on initial states of *Init-Vehicles*. Consider any reachable state in which $x_1 = x_2 + l_2$ and *collided* = *false*. Then Lemma 5.1 implies that *safe-measure* ≥ 0. That is, either

$$x_1 - (x_2 + l_2) \geq \frac{(\dot{x}_2)^2 - (\dot{x}_1)^2 - (v_{allow})^2}{2a_{min}}$$

or

$$\dot{x}_1 + v_{allow} \geq \dot{x}_2.$$

In the latter case, we are done. In the former, setting $x_1 - (x_2 + l_2) = 0$, we get

$$\frac{(\dot{x}_2)^2 - (\dot{x}_1)^2 - (v_{allow})^2}{2a_{min}} \leq 0$$

$$(\dot{x}_2)^2 \leq (\dot{x}_1)^2 + (v_{allow})^2 \leq (\dot{x}_1 + v_{allow})^2$$

$$\dot{x}_2 \leq \dot{x}_1 + v_{allow},$$

as needed. ■

Now we use Lemma 5.2 to prove that the system is in fact *correct*, i.e., that it implements *Safe-Vehicles*.

Lemma 5.3 *Let f be the identity on all state components of Safe-Vehicles(velocities, positions, accelerations, and the collided flag). Then f is a forward simulation from the composed system Controlled-Vehicles(Init-Vehicles, A_1, C_2), where A_1 is any implementation of Controller$_1$, to Safe-Vehicles.*

Proof: By induction on the number of steps in the hybrid execution.

Start States: Suppose s_{IP} is a start state of *Controlled-Vehicles*(*Init-Vehicles*, A_1, C_2), and $(s_{IP}, s_{SP}) \in f$. We have to prove that s_{SP} is a valid start state of *Safe-Vehicles*. By the definition of start states of *Safe-Vehicles*, it must satisfy the conditions of *Init-Vehicles*, which follows from the fact that s_{IP} is a start state of *Controlled-Vehicles* and so it does satisfy all those conditions. Also, by Lemma 5.2, s_{SP} satisfies *Safety*.

Discrete Steps: The only discrete steps are *collide, e* and the internal steps of A_1. The latter two steps cannot change any of the quantities involved. Since the *collide* step is the same for both automata, it respects the simulation relation. Also, the effects of the *collide* step satisfy *safety* vacuously, thus the state reached after the *collide* action is a valid state of *Safe-Vehicles*.

Trajectories: Suppose that w_{IP} is an I-trajectory of *Controlled-Vehicles* and its first state s_{IP} is reachable. Suppose that s_{SP} is a reachable state of *Safe-Vehicles* such that $(s_{IP}, s_{SP}) \in f$. Let the corresponding hybrid execution fragment of *Safe-Vehicles*

consist of a single trajectory w_{SP}, where $w_{SP}(t).\dot{x}_i = w_{IP}(t).\dot{x}_i$, $w_{SP}(t).x_i = w_{IP}(t).x_i$ for $i \in \{1, 2\}$, $w_{SP}(t).collided = w_{IP}(t).collided$ and $w_{SP}(t).now = w_{IP}(t).now$. It is obvious that the two trajectories have the same hybrid trace and that the final states of both trajectories are f-related.

We need to show that w_{SP} is in fact a trajectory of *Safe-Vehicles*. By the definition of a trajectory we must show that it satisfies all the properties of a trajectory of *Init-Vehicles*, but this is trivial, since it is a trajectory of *Controlled-Vehicles* which has all the restrictions of *Init-Vehicles*. We must also show that it always satisfies the *safety* condition, but this follows directly from Lemma 5.2. Therefore, f is also a valid simulation relation for all the trajectories.

■

Theorem 5.4 *C_2 is correct for Init-Vehicles, where correctness is defined by Definition 2.*

Proof: To prove *correctness*, we need to show that the *Controlled-Vehicles*(*Init-Vehicles*, A_1, C_2) automaton, where A_1 is any implementation of $Controller_1$, implements *Safe-Vehicles*. *Controlled-Vehicles*(*Init-Vehicles*, A_1, C_2) and *Safe-Vehicles* are comparable and by Lemma 5.3, there is a simulation relation f from *Controlled-Vehicles* to *Safe-Vehicles*. Therefore, this composed system implements *Safe-Vehicles*. This proves *correctness* of C_2. ■

5.4 Optimality of C_2

We devise a new controller, *Necessary-C_2*, which is slightly less restrictive than C_2 and prove that *Necessary-C_2* gives necessary conditions for satisfying *correctness*.

Automaton 7(Necessary-C_2): This automaton is exactly like C_2, except that condition (2.4) for trajectories is slightly modified. In particular, *Necessary-C_2* commands maximum deceleration when *safe-measure* < 0, while C_2 does it when *safe-measure* ≤ 0. The formal model is given in Figure 5-3; thus the only difference is the boundary in condition (2.4). We claim that this condition is necessary to guarantee *correctness*.

Variables:

Input, Output, Internal: same as in *Controller*$_2$

Trajectories:

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

1. it satisfies condition 1 of *Controller*$_2$, plus
2. for all $t \in I$,
   - 2.1-2.3 of *Controller*$_2$ are satisfied
   - 2.4. if *collided* = *false* and *safe-measure* < *0* then $w(t).acc_2 = a_{min}$

Figure 5-3: *Necessary-C*$_2$ Hybrid Automaton

We define a notion of *Bad* controllers. Let B_1 and B_2 be implementations of *Controller*$_1$ and *Controller*$_2$, respectively. Then B_1 is *Bad* for B_2 if B_1 makes B_2 violate Predicate S by going out of the *safe-measure* region. B_2 is *Bad* if there exists some B_1 which is *Bad* for it.

Definition 4(Bad Controller$_1$): Let B_1 and B_2 be implementations of *Controller*$_1$ and *Controller*$_2$, respectively. Then B_1 is **Bad** for B_2 if and only if in the system *Controlled-Vehicles*(*Init-Vehicles*, B_1, B_2) there exists a reachable state s that does not satisfy Predicate S.

Definition 5(Bad Controller$_2$): B_2 is **Bad** if and only if there exists some B_1 that is Bad for this B_2.

The following lemma shows that if B_2 is *Bad*, then we can construct a B_1' that is *Bad* for B_2 and decelerates at the maximum rate once Predicate S is violated. This B_1' will later be used to show that B_2 can violate *Safety*.

Definition 6(VeryBad): Let an implementation B_1' of *Controller*$_1$ be called **Very-Bad** for an implementation B_2 of *Controller*$_2$ if

1. B_1' is *Bad* for B_2;
2. In any execution α of *Controlled-Vehicles*(*Init-Vehicles*, B_1', B_2), any state s that does not satisfy Predicate S, and any state s' occurring strictly after s, it is the case that $s'.acc_1 = a_{min}$.

Lemma 5.5 *If B_2 is Bad then there exists an implementation B_1' of Controller$_1$, such that B_1' is VeryBad for B_2.*

Proof: Since B_2 is *Bad*, there exists B_1 that is *Bad* for this B_2. Using B_1, we construct an implementation of *Controller$_1$*, called B_1', as follows. We add an extra internal variable *stop*, initially $stop = false$. In any execution of the system *Controlled-Vehicles*(*Init-Vehicles*, B_1', B_2),

1. B_1' behaves exactly like B_1 until Predicate S is violated.

2. Exactly when *safe-measure* < 0, an internal variable *stop* is set to *true*. Note that B_1' has enough information (positions and velocities of both vehicles, a_{min}) to detect when *safe-measure* < 0.

3. If $stop = true$, then $acc_1 = a_{min}$.

B_1' preserves the behavior of B_1 up to the point when Predicate S is violated, so B_1' is also *Bad* for B_2. The second condition of the *VeryBad* definition is satisfied by construction. So B_1' is *VeryBad* for B_2. ■

Lemma 5.6 *Let B_2 be an implementation of Controller$_2$. If there exists an implementation B_1 of Controller$_1$ such that* Controlled-Vehicles(Init-Vehicles, B_1, B_2) *does not implement* Controlled-Vehicles(Init-Vehicles, B_1, *Necessary-C_2*) *then B_2 is Bad.*

Proof: We must show that there exists some implementation B_1 of *Controller$_1$*, such that the system *Controlled-Vehicles*(*Init-Vehicles*, B_1, B_2) has a reachable state s that does not satisfy Predicate S.

Definition 7(Predicate T): If *safe-measure* < 0 then $a_2 = a_{min}$.

The only restriction of *Necessary-C_2* (trajectory condition (2.4)) requires that if *collided* = *false* then Predicate T is satisfied.

Claim 1. There exists B_1 such that the system *Controlled-Vehicles*(*Init-Vehicles*, B_1, B_2) has a reachable state s in which *collided* = *false* and Predicate T is not satisfied.

Proof of Claim 1. Proof by contradiction. Suppose such B_1 does not exist. Then, for all implementations B_1 of $Controller_1$, all the reachable states of *Controlled-Vehicles*(*Init-Vehicles*, B_1, B_2) in which *collided* = *false* satisfy Predicate T. But then all the hybrid traces of *Controlled-Vehicles*(*Init-Vehicles*, B_1, B_2) are allowed by *Controlled-Vehicles*(*Init-Vehicles*, B_1, *Necessary-*C_2). It follows that for all implementations B_1 of $Controller_1$, *Controlled-Vehicles*(*Init-Vehicles*, B_1, B_2) implements *Controlled-Vehicles*(*Init-Vehicles*, B_1, *Necessary-*C_2). This contradicts the hypothesis of the Lemma.

Claim 2. Predicate S is violated in state s of *Controlled-Vehicles*(*Init-Vehicles*, B_1, B_2) in which Predicate T is not satisfied.

Proof of Claim 2. In state s, *safe-measure* < 0 and $a_2 \neq a_{min}$, since Predicate T is not satisfied. But (because *collided* = *false* in s) this means that Predicate S is also violated. This suffices.

B_1 is *Bad* for B_2 by Claim 2. This proves that B_2 is *Bad*. ■

Let B_2 be *Bad* and B_1' be *VeryBad* for B_2. Then Lemma 5.7, shows that in any hybrid execution of *Controlled-Vehicles*(*Init-Vehicles*, B_1', B_2), once Predicate S is violated, it will continue to be violated throughout the hybrid execution. In Lemma 5.8 we show that violation of Predicate S always leads to violation of *safety*. Finally, Theorem 5.9 proves that if for an implementation B_2 there exists an implementation B_1 of $Controller_1$ such that *Controlled-Vehicles*(*Init-Vehicles*, B_1, B_2) does not implement *Controlled-Vehicles*(*Init-Vehicles*, B_1, *Necessary-*C_2) then this B_2 is not a *correct* controller.

Lemma 5.7 *Let B_2 be Bad, and B_1' be VeryBad for B_2. Then, in any execution of Controlled-Vehicles(Init-Vehicles, B_1', B_2), Predicate S is violated in all the states that occur strictly after a state s in which collided = false and Predicate S is violated.*

Proof: By induction on the number of steps in a fixed hybrid execution h. Suppose that there exists a state s in h in which *collided* = *false* and Predicate S is violated.

Initially, the claim is true vacuously.

The only discrete steps are *collide,* e and the internal steps of B_1'. The latter two steps cannot change any of the quantities involved. The effect of the *collide* step ensures that *collided* = *true* which makes the Lemma true vacuously.

Consider any trajectory w of h, whose domain is the interval $[0, t]$, that occurs after Predicate S is violated and in which *collided* = *false* throughout the trajectory. From the definition of B_1', $\ddot{x}_1 = acc_1 = a_{min}$ throughout that trajectory.

Let's apply Lemma 3.5 to our problem. This Lemma states that if both the right and left derivatives of a function on an interval are non-positive, then the function is non-increasing on that interval. Right and left derivatives of *safe-measure* are always the right and left derivatives, respectively, of either f and g, as stated in Theorem 3.6. So we only have to prove that $\dot{f} \leq 0$ and $\dot{g} \leq 0$ throughout the interval $[0, t]$. We have:

$$\dot{f} = \dot{x}_1 - \dot{x}_2 + \frac{1}{a_{min}}(\dot{x}_2\ddot{x}_2 - \dot{x}_1\ddot{x}_1).$$

By the definition of B_1', we have that $\ddot{x}_1 = a_{min}$. Also, by restriction on *Init-Vehicles*, $\ddot{x}_2 = acc_2 \geq a_{min}$. Therefore,

$$\begin{aligned}
\dot{f} &= \dot{x}_1 - \dot{x}_2 + \frac{1}{a_{min}}(\dot{x}_2\ddot{x}_2 - \dot{x}_1 a_{min}) \\
&= \dot{x}_1 - \dot{x}_2 + \left(\frac{\dot{x}_2\ddot{x}_2}{a_{min}} - \dot{x}_1\right) \\
&= \dot{x}_2\left(\frac{\ddot{x}_2}{a_{min}} - 1\right) \\
&\leq \dot{x}_2\left(\frac{a_{min}}{a_{min}} - 1\right) \\
&= 0.
\end{aligned}$$

Similarly for $\dot{g}$:

$$\begin{aligned}
\dot{g} &= \ddot{x}_1 - \ddot{x}_2 \\
&= a_{min} - \ddot{x}_2 \\
&\leq a_{min} - a_{min} \\
&= 0.
\end{aligned}$$

Therefore, by Lemma 3.5, *safe-measure* is a non-increasing function. Since in $w(0)$, *safe-measure* $<$ *0*, *safe-measure* is negative throughout the trajectory, which

means that the *Controlled-Vehicles*(*Init-Vehicles*, B_1', B_2) system violates Predicate S throughout the trajectory. ■

Lemma 5.8 *Let B_1' be VeryBad for B_2. Then there exists a reachable state s' of Controlled-Vehicles(Init-Vehicles, B_1', B_2) that does not satisfy safety.*

Proof: By the fact that B_1' is *VeryBad*, there must exist some reachable state s of this *Controlled-Vehicles* system, in which Predicate S is violated. Then, if there exists a state s', reachable from s, in which the vehicles do collide, then by Lemma 5.7, in s', *safe-measure* < 0. But that means that in s', $\dot{x}_2 > \dot{x}_1 + v_{allow}$, violating *safety*. All we have to prove now is that there exists a reachable state s' in which the vehicles do, in fact, collide.

Suppose they don't collide. Since the first vehicle eventually stops (it is decelerating at its maximum rate), this means that the second one also has to stop. Let x_i, $\dot{x}_i$ represent state components in state s, and ${x_i}'$ represent the state components of s'. Then,

$$ {x_1}' = x_1 + \frac{\dot{x}_1^2}{-2a_{min}} \text{ and } {x_2}' \geq x_2 + \frac{\dot{x}_2^2}{-2a_{min}}. $$

From our non-collision assumption we get,

$$ \begin{aligned} {x_1}' &\geq {x_2}' + l_2 \\ x_1 + \frac{\dot{x}_1^2}{-2a_{min}} &\geq x_2 + \frac{\dot{x}_2^2}{-2a_{min}} + l_2 \\ x_1 - (x_2 + l_2) &\geq -\frac{(\dot{x}_2)^2 - (\dot{x}_1)^2}{2a_{min}} \\ x_1 - (x_2 + l_2) &\geq -\frac{(\dot{x}_2)^2 - (\dot{x}_1)^2 - (v_{allow})^2}{2a_{min}}, \end{aligned} $$

which means that *safe-measure* ≥ 0 in state s. But then Predicate S is true in state s, contrary to our assumption.

Therefore, the vehicles do collide in some reachable state s', and s' does not satisfy *safety*. ■

Theorem 5.4 shows that any controller of the trailing vehicle that does not implement *Necessary-*C_2 , and, therefore, violates Predicate S, violates *safety* for some behavior of the leading vehicle.

Theorem 5.9 *Let B_2 be an implementation of Controller$_2$. If there exists an implementation B_1 of Controller$_1$, such that Controlled-Vehicles(Init-Vehicles, B_1, B_2) does not implement Controlled-Vehicles(Init-Vehicles, B_1, Necessary-C_2), then B_2 is not a correct controller for Init-Vehicles.*

Proof: B_2 must be *Bad* by Lemma 5.6. Then, by Lemma 5.5, there exists an implementation B_1' of *Controller$_1$* that is *VeryBad* for B_2. Lemma 5.8 shows that the system *Controlled-Vehicles*(*Init-Vehicles*, B_1', B_2) has a reachable state s' that violates *safety*, which means that B_2 is not *correct*. ■

5.5 Results

Theorem 5.4 shows that the controller C_2 is sufficient for guaranteeing *correctness*, and Theorem 5.9 proves that the controller *Necessary-C_2* is necessary to guarantee *correctness*. Combining these two results, we can check correctness, in terms of *safety*, of any implementation C of *Controller$_2$*. C is correct if it implements C_2, and is not correct if *Controlled-Vehicles*(*Init-Vehicles*, B_1, C) does not implement *Controlled-Vehicles*(*Init-Vehicles*, B_1, *Necessary-C_2*) for some implementation B_1 of *Controller$_1$*. Since C_2 and *Necessary-C_2* differ in behavior only in the boundary cases, they can be used to check correctness of most controllers.

Chapter 6

Delayed Response

In this chapter we consider the delay between the receipt of information by the controller for vehicle 2 and its resulting action. There are two distinct types of delay to consider — the inbound and the outbound delay; we model them separately. The inbound delay is delay due to the controller's sensors getting the information (about the position and velocity of the leading vehicle). The outbound delay is the delay between the time the controller makes the decision and the time the decision is actually implemented by the vehicle.

These delays are between the vehicle and its controller, and so only the delays in the trailing vehicle are relevant to our analysis, as we only care about the external *behavior* of the leading vehicle, and not about its controller. In particular, if we were to extend our analysis to a multi-car case, each vehicle could have its own delay characteristics; our analysis would still hold up, since we would look at the delays in the trailing vehicle of each vehicle pair.

We use levels of abstractions to deal with the complexity of the delayed case. First, we devise the "delayed" controller, and then we use simulation relations to the controller of the first (ideal) case, to show that this controller is sufficient for the delayed case. We also give a slightly less restrictive controller specification, and prove that it is necessary to guarantee correctness.

6.1 Delay Buffers

We model both the inbound and the outbound delays by special delay buffers. To obtain the delayed system, we compose our new controller with the delay buffers. First, we introduce a generic delay buffer *D-Buffer*, and then specify the inbound and outbound delays as instances of the generic automaton.

Parameters:

n — number of input variables

S_i, S_o - two disjoint sets of variables with n members in each set

Let $V(s)$, where s is a variable, be a valuation of the variable s;

$V(S)$, where S is a set of variables, be a valuation of the entire set.

$var : S_i \rightarrow S_o$, a 1-1 mapping from S_i to S_o

d — the delay of the buffer

$Init : [0, d] \rightarrow V(S_o)$ — a function giving the output of the buffer for the initial d time period

Variables:

Input: S_i

Output: S_o

Internal: $saved : [0, d] \rightarrow V(S_o)$, where $saved$ acts as FIFO queue for outputs;

initially, $saved = Init$

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:

1. for all variables $v_i \in S_i$,

$w(t).V(var(v_i)) = w(t).saved(d).v_i$

2. $\forall t' \in [0, d]$,

$$w(t).saved(t') = \begin{cases} w(0).saved(t' - t) & \text{if } t' > t, \\ w(t - t').V(S_i) & \text{otherwise.} \end{cases}$$

Figure 6-1: *D-Buffer*(n, S_i, S_o, var, d, $Init$) Hybrid Automaton

Automaton 8(D-Buffer(n, S_i, S_o, var, d, $Init$)): The buffer outputs its inputs (the connection between inputs and outputs given by the var function) exactly in the same order as received, and exactly time d later. Initially, it outputs values given by

the $Init$ function. The automaton is described formally in Figure 6-1.

Parameters

- n is the number of (input, output) variable pairs;
- S_i, S_o are two disjoint sets of input and output variables names, respectively;
- $var : S_i \rightarrow S_o$ is a 1-1 mapping from input to output variables. Each pair corresponds to a variable the buffer is "delaying." For convenience, we also define valuations of single variable names and their sets, by the function V.
- d is the maximum absolute delay. It is the same for all variables delayed by this buffer.
- $Init$ is a function that sets up the initial "contents" of the buffer — it tells the buffer what to output for the initial d time period, when no inputs have reached the output yet.

Variables

- S_i is a set of input variables, and S_o is a set of output variables. Both sets are given by the parameters of the automaton.
- *saved* is an internal variable that stores the input variables for the delay duration d. Initially, it is prefed with information using the function *Init*. *saved* acts as a First-In-First-Out continuous queue of the buffers inputs. $saved(0) = V(S_i)$ and represents the most recent input; $saved(d)$ represents the least recent input, the one that is just about to be output.

Trajectories Condition (1) sets up the output variables to take their values from the internal variable *saved*, exactly time d ago. Condition (2) updates the *saved* variable with new information.

HA Trajectories are not restricted in duration, so axiom $T2$ is satisfied; also, trajectories are never *required* to stop, so axiom $T3$ is also satisfied. Thus, by discussion in Section 2.6, *D-Buffer* is a *HA*.

Properties From the automaton definition, it follows that

1. For the initial d time period, output $w(t).V(S_o) = Init(t)$ (from the initialization of internal variable *saved*, and trajectory condition (1)).

2. Afterwards, for all t, $w(t).V(S_o) = w(t-d).V(S_i)$ (from trajectory conditions (1) and (2)).

6.2 The System with Inbound and Outbound Delays

We compose the delayed controller using two instances of the delay buffer *D-Buffer*, and a modified controller.

First, we define two instances of the *D-Buffer* automaton, the inbound and outbound delay buffers.

Automaton 9(D_i, the inbound delay buffer): D_i = *D-Buffer*(3, $\{x_1, \dot{x}_1, \ddot{x}_1\}$, $\{x_{d1}, \dot{x}_{d1}, \ddot{x}_{d1}\}$, var, d_i, $Init$), where $var(x_1) = x_{d1}$, $var(\dot{x}_1) = \dot{x}_{d1}$, and $var(\ddot{x}_1) = \ddot{x}_{d1}$; $d_i \in \mathsf{R}^{\geq 0}$, the inbound delay, is the "information" delay – the time it takes the controller to get the information from the sensors. The inbound delay automaton delays the position, velocity and acceleration data of the first vehicle, with delay d_i.

Given arbitrary initial values for input values for $x_1, \dot{x}_1, \ddot{x}_1$, *Init* is set up so that the $Init(0)$ matches up with these values. The least restrictive conditions on the behavior of the second controller are obtained if we assume that the leading vehicle was decelerating at the maximum possible rate throughout the d_i time period. Then the second controller does not have to push the brakes thinking that there is a "dangerous" situation during the initial d_i time period. Safety is preserved as long

as $Init(0)$ matches up with the actual data at time 0. So, *Init* assumes vehicle 1 was decelerating at a_{min} and fills the position and velocity values accordingly.

Formally, *Init* is set up as follows. For any start values a_1, a_2, a_3 of $x_1, \dot{x}_1, \ddot{x}_1$, respectively, construct a trajectory w of D_i, of length d_i such that

$$\forall t \in [0, d_i), w(t).\ddot{x}_1 = a_{min}, \text{ and } w(d_i).\ddot{x}_1 = a_3,$$
$$\forall t \in [0, d_i], w(t).\dot{x}_1 = a_2 + (t - d_i)a_{min},$$
$$\forall t \in [0, d_i], w(t).x_1 = a_1 + a_2(t - d_i) + \frac{a_{min}(t - d_i)^2}{2}.$$

Then, $w(d_i).\{x_1, \dot{x}_1, \ddot{x}_1\} = \{a_1, a_2, a_3\}$, so that it matches up with real data at time 0. Let $Init(t) = w(d_i - t).(x_1, \dot{x}_1, \ddot{x}_1)$, then $Init(0)$ matches up with the actual values at time 0.

Automaton 10(D_o, the outbound delay buffer): D_o = *D-Buffer*(1, $\{acc_{d2}\}$, $\{acc_2\}$, *var*, d_o, *Init*), where $var(acc_{d2}) = acc_2$; $d_o \in \mathsf{R}^{\geq 0}$, the outbound delay, is the "action delay" – the time that it can take for a vehicle to react to the controller's directives; $\forall t \in [0, d_i], Init(t) = a_{min}$. Setting *Init* so conservatively makes the vehicles safe in the initial d_o time interval even if the first vehicle starts decelerating at the maximum rate. This is the best we can do without any further knowledge. This automaton delays the acceleration commands by d_o.

Finally, we modify the controller specification so that it communicates with the buffers correctly.

Automaton 11(Spec-D_2): The controller *Spec-*D_2 (see Figure 6-2), composed with delay buffers D_i and D_o, implements $Controller_2$. It is an *HA* since the changes to trajectory definitions do not violate axioms $T2-3$. It is similar to $Controller_2$, except that the input and output variables are changed, and the restriction on trajectories is modified.

Variables The new controller gets its data about the first vehicle from the inbound delay buffer D_i , and the "self" data (data about the second vehicle) directly from the *Init-Vehicles* automaton. This models the situation in which there is delay in getting the information via the sensors about the other vehicle, but there is perfect

Variables:

Input: $x_{d1}, \dot{x}_{d1} \in \mathsf{R}^{\geq 0}$, $\ddot{x}_{d1} \in \mathsf{R}$

$x_2, \dot{x}_2 \in \mathsf{R}^{\geq 0}$, $\ddot{x}_2 \in \mathsf{R}$

Output: acc_{d2}, initially arbitrary, where $acc_{d2} \geq a_{min}$

Internal: internal variables of *Controller*$_2$ ($\dot{x}_{int2}$ and x_{int2}),

Trajectories:

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

1. w' is a trajectory of *Controller*$_2$,

   where w' is a trajectory constructed from w such that in all states of w',

   $w'(t).x_1 = w(t).x_{d1}$, $w'(t).\dot{x}_1 = w(t).\dot{x}_{d1}$, $w'(t).\ddot{x}_1 = w(t).\ddot{x}_{d1}$,

   $w'(t).acc_2 = w(t).acc_{d2}$, and all other state components are the same as in *Spec-D*$_2$

Figure 6-2: *Spec-D*$_2$ Hybrid Automaton

self information. The output variable goes into the outbound delay buffer D_o.

Trajectories Condition (1) makes sure that the trajectories of *Spec-D*$_2$ are allowed by *Controller*$_2$, after the variable change. It is needed to ensure that *Spec-D*$_2$ composed with the delay buffers implements *Controller*$_2$.

Finally, we compose this new controller with the delay buffers, to get an automaton that implements *Controller*$_2$.

Automaton 12(Delayed-Controller$_2(D)$): *Delayed-Controller*$_2$ (D) =

VarHide($\{x_{d1}, \dot{x}_{d1}, \ddot{x}_{d1}, acc_{d2}\}$, $D_i \parallel D \parallel D_o$), where D is an implementation of *Spec-D*$_2$(see Figure 6-3). The variables that communicate between the sensors and the controller are hidden so that *Delayed-Controller*$_2$ (D) is comparable to C_2.

The following two theorems state relationships between the variables of *Spec-D*$_2$ at different points in time.

Theorem 6.1 *Let A_1 be any implementation of Controller*$_1$*, s'' be a reachable state of the Controlled-Vehicles(Vehicles, A_1, Delayed-Controller*$_2$*(Spec-D*$_2$*)) system, and s' be a state reachable from s'' such that $s'.now = s''.now + d_i$ and collided = false in s'. Then,*

$$s'.x_{d1} = s''.x_1, s'.\dot{x}_{d1} = s''.\dot{x}_1, \text{ and } s'.\ddot{x}_{d1} = s''.\ddot{x}_1.$$

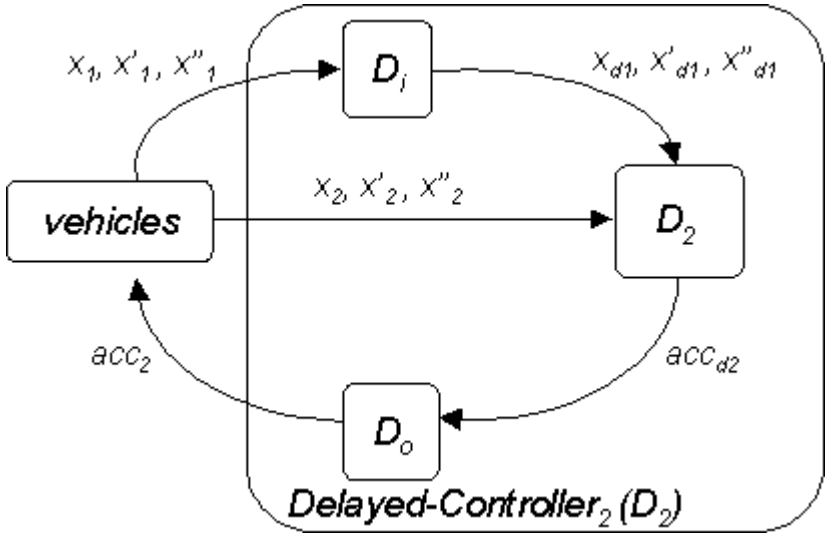

Figure 6-3: *Delayed-Controller*$_2$ (D) hybrid automaton

Proof: The theorem states the relationships between a state s'', and a s' that occurred time d_i after s'' (see Figure 6-5). It states that the data received through the input buffer D_i is exactly the same as the actual data time d_i ago. This follows from Property 2 of *D-Buffer* (outputs of a buffer in state s equal the buffer's inputs in state s' that occurred time d ago). ■

Theorem 6.2 *Let A_1 be any implementation of Controller$_1$, s' be a reachable state of the Controlled-Vehicles(Vehicles, A_1, Delayed-Controller$_2$(Spec-D_2)) system, and s be a state reachable from s' such that $s.now = s'.now + d_o$, collided = false in s, and s is reachable from s'. Then,*

1. $s.\dot{x}_1 \geq s'.\dot{x}_{d1} + a_{min}t'$, *where* $t' = \min(\frac{-s'.\dot{x}_{d1}}{a_{min}}, d_i + d_o)$.

2. $s.x_1 \geq s'.x_{d1} + s'.\dot{x}_{d1}t' + \frac{1}{2}a_{min}t'^2$, *where* $t' = \min(\frac{-s'.\dot{x}_{d1}}{a_{min}}, d_i + d_o)$.

Proof: The theorem states the relationships between a state s', and a state s that occurs time d_o after s' (see Figure 6-5). By Theorem 6.1, the position, velocity and acceleration data in state s' equal the "delayed" values in state s. By Properties 1 ($\ddot{x}_i \geq a_{min}$) and 2 (the fact that position and velocity are integrals of acceleration)

of *Vehicles* (see chapter 4), the position and velocity bounds can be obtained by integrating the maximum deceleration.

In particular, the first clause states that the actual velocity of vehicle 1 in state s is at least as great as its velocity in state s'' that occurred time $d_i + d_o$ in the past, plus the maximum possible decrease in velocity during that time. The second clause states that the actual position of vehicle 1 in state s is at least as great as its position in state s'' that occurred time $d_i + d_o$ ago, adjusted by the velocity time $d_i + d_o$ ago and the maximum allowed deceleration. ■

6.3 Correctness of *Delayed-Controller*$_2$

We give an implementation of *Spec-*D_2 that is sufficient to guarantee *correctness*.

Automaton 13(D_2): The controller D_2 (see Figure 6-4) is a sufficient controller to guarantee *correctness* of *Delayed-Controller*$_2(D_2)$. It is an *HA* since the changes from *Spec-*D_2 do not violate axioms $T2-3$. It is similar to C_2 in that it also tries to keep the second vehicle within the bounds set by *safe-measure*$_d$, which is *safe-measure* redefined for the delayed case.

Definition *safe-measure*$_d$ is exactly like *safe-measure*, modified by replacing the position, velocity and acceleration data of vehicles by their delayed values. For vehicle 2, the delayed values are the values resulting from executing the controller's commands for the outbound delay time d_o, as given by x_{int2} and $\dot{x}_{int2}$; for vehicle 1, the "worst possible" delayed values are generated by decelerating at the maximum possible rate for the last $d_i + d_o$ time units, since the controller's information is $d_i + d_o$ time units "old." In particular, using Theorems 6.2 and 6.3,

replace x_1 by $x_{d1} + \dot{x}_{d1}t' + \dfrac{a_{min}t'^2}{2}$

replace x_2 by x_{int2}

replace $\dot{x}_1$ by $\dot{x}_{d1} + a_{min}t'$

replace $\dot{x}_2$ by $\dot{x}_{int2}$

Definition:

$$\textit{safe-measure}_d = \max((x_{d1} + \dot{x}_{d1}t' + \frac{a_{min}t'^2}{2}) - (x_{int2} + l_2) + \frac{(\dot{x}_{int2})^2 - (\dot{x}_{d1} + a_{min}t')^2 - (v_{allow})^2}{2a_{min}},$$
$$(\dot{x}_{d1} + a_{min}t') - \dot{x}_{int2} + v_{allow}),$$

where $t' = \min(d_i + d_o, -\frac{\dot{x}_{d1}}{a_{min}})$

Variables:

Input: same as *Spec-D_2*

Output: acc_{d2}, initially if $\textit{safe-measure}_d \leq 0$, then $acc_{d2} = a_{min}$,
else arbitrary, where $acc_{d2} \geq a_{min}$

Internal: internal variables of *Spec-D_2*, plus
a_2 - maps from an interval $[0, d_o]$ to R,
initially, $\forall t \in [0, d_o], a_2(t) = a_{min}$

Trajectories:

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

1. w is a trajectory of *Spec-D_2*
2. if $\textit{collided} = \textit{false}$ in $w(0)$ then for all $t \in I$, $t > 0$:
   2.1. if in state $w(t)$ $\textit{safe-measure}_d \leq 0$ then $acc_{d2} = a_{min}$ else $acc_{d2} \geq 0$
   2.2. $\forall t' \in [0, d_o]$,
   $$w(t).a_2(t') = \begin{cases} w(0).a_2(t' - t) & \text{if } t' > t \\ w(t - t').acc_{d2} & \text{otherwise} \end{cases}$$
   2.3. $w(t).\dot{x}_{int2} = w(t).\dot{x}_2 + \int_0^{d_o} w(t).a_2(u)du$
   2.4. $w(t).x_{int2} = w(t).x_2 + \int_0^{d_o} w(t).\dot{x}_{int2}du$

Figure 6-4: D_2 Hybrid Automaton

Variables External variables are the same as in *Spec-D_2*. x_{int2}, $\dot{x}_{int2}$, represent the position and velocity of the second vehicle after time d_o passes, provided *collided* still equals *false*. They are used for $\textit{safe-measure}_d$ calculations. In order to calculate the values of these variables, we add a special buffer, a_2, that stores the controller's acceleration commands that have not been executed yet (due to the outbound delay). Initially, $\forall t \in [0, d_o], a_2(t) = a_{min}$, so that it matches the initial information in the outbound buffer.

Trajectories Condition (1) restricts each trajectory to satisfy *Spec-D_2* requirements. Condition (2.1) is the same as for C_2, substituting the new *safe-measure$_d$* definition. Clause (2.2) sets up a_2 to save acceleration commands output by the controller in the last d_o time units. $a_2(0)$ represents the most recent command issued, and $a_2(d_o)$ represents the command that is going to be executed next. Finally, clauses (2.3) and (2.4) set up x_{int2} and $\dot{x}_{int2}$ to be the integrals of the commanded acceleration. Since there is no uncertainty, these variables represent the actual values of the corresponding variables in *Init-Vehicles*, but at time d_o in the future (see Theorem 6.3).

Theorem 6.3 *Let A_1 be any implementation of Controller$_1$, s' be reachable states of Controlled-Vehicles(Vehicles, A_1, Delayed-Controller$_2$(Spec-D_2)), and s be reachable from s', such that $s.now = s'.now + d_o$, collided = false in s. Then,*

1. $s.\dot{x}_2 = s'.\dot{x}_{int2}$;
2. $s.x_2 = s'.x_{int2}$.

Proof: The theorem states that the actual position and velocity of vehicle 2 in state s' are equal to the "predicted" values (given by the internal variables) in state s that occurred time d_o earlier. This follows from Property 2 of *D-Buffer* (buffer outputs in state s equal buffer inputs in state s' that happened time d before s), Property 2 of *Vehicles* (position and velocity are integrals of acceleration), and Conditions (2.3-4) of D_2 trajectories. ■

We prove that D_2 is sufficient to guarantee *correctness*. Throughout the rest of this section we will use the following notation:

For any implementation A_1 of *Controller$_1$*, let

$CV_I(A_1) =$ *Controlled-Vehicles*(*Init-Vehicles*, A_1, *Delayed-Controller*$_2(D_2)$), and

$CV_D(A_1) =$ *Controlled-Vehicles*(*Init-Vehicles*, A_1, C_2).

We show that $CV_D(A_1)$ implements $CV_I(A_1)$.

The key result, proven in Lemma 6.4 proves that if the old *safe-measure* (the one used in the ideal case) is non-positive in some state of a trajectory of *Delayed-*

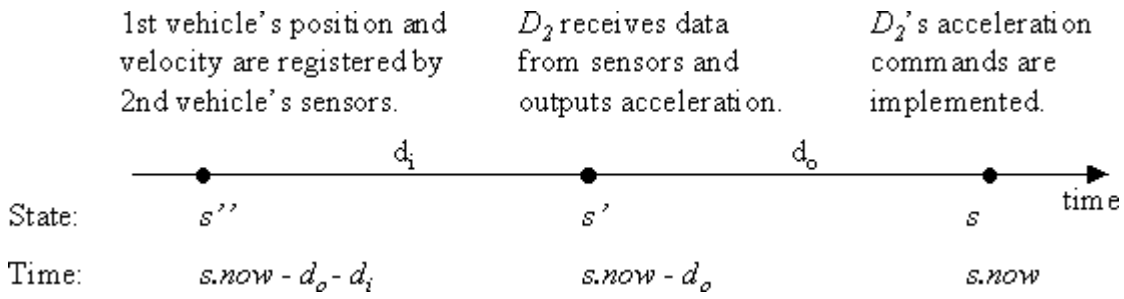

Figure 6-5: The timing diagram of the in- and outbound controller

$Controller_2(D_2)$, then $Delayed\text{-}Controller_2(D_2)$ will also output maximum deceleration, just as the old (ideal) controller C_2 would. Formally,

Lemma 6.4 *Let A_1 be any implementation of $Controller_1$, and let s be a reachable state of the $CV_D(A_1)$ system, such that $s.collided = false$ and $safe\text{-}measure \leq 0$. Then $s.acc_2 = a_{min}$.*

Proof: By induction on the number of discrete and continuous steps in the hybrid execution. Initially, the lemma is true by restriction on the start states of D_2. The discrete steps e and the internal steps of A_1 do not change any of the quantities involved; the *collide* step makes the Lemma true vacuously.

Without loss of generality, consider hybrid executions where all trajectories have duration less than d_o. Let s be any reachable state such that $s.collided = false$ and $safe\text{-}measure \leq 0$ in s. If $s.now < d_o$ (we are still in the initial period when the controller's commands do not reach the vehicle, and D_o just outputs maximum deceleration), then by definition of the outbound buffer D_o(from the $Init$ function), $acc_2 = a_{min}$ and we are done. Otherwise, at s we have

$$x_1 - (x_2 + l_2) \leq -\frac{(\dot{x_2})^2 - (\dot{x}_1)^2 - (v_{allow})^2}{2a_{min}} \tag{6.1}$$

and

$$\dot{x_2} \geq \dot{x}_1 + v_{allow}. \tag{6.2}$$

Let's look at any reachable state s' of this system such that $s'.now = s.now - d_o$ (see Figure 6-5), and from which s is reachable in time d_o (s' is the state in which the

controller D_2 decided what acceleration the vehicle should have in state s). This states s' cannot be in the same trajectory because of our restriction on hybrid executions, so we can use the inductive hypothesis on s'. For such a state, by Theorem 6.3, $s'.x_{int2} = s.x_2$ and $s'.\dot{x}_{int2} = s.\dot{x}_2$. Also, by Theorem 6.2,

$$s.\dot{x}_1 \geq s'.\dot{x}_{d1} + a_{min}t',$$
$$s.x_1 \geq s'.x_{d1} + s'.\dot{x}_{d1}t' + \frac{a_{min}t'^2}{2}$$

where $t' = \min(d_i + d_o, -\frac{\dot{x}_{d1}}{a_{min}})$.

We take inequality 6.1, and substitute the delayed values for the actual ones (using the statements above), still keeping the correctness of the inequality.

$$x_1 - (x_2 + l_2) \leq -\frac{(\dot{x_2})^2 - (\dot{x}_1)^2 - (v_{allow})^2}{2a_{min}} \qquad \text{(at } s\text{)}$$
$$x_{d1} + \dot{x}_{d1}t' + \frac{a_{min}t'^2}{2} - (x_{int2} + l_2) \leq -\frac{(\dot{x}_{int2})^2 - (\dot{x}_{d1} + a_{min}t')^2 - (v_{allow})^2}{2a_{min}}, \qquad \text{(at } s'\text{)}$$

where $t' = \min(d_i + d_o, -\frac{\dot{x}_{d1}}{a_{min}})$.

But this is exactly equivalent to the first part of *safe-measure*$_d$ at s'!

We do the same with inequality 6.2.

$$\dot{x_2} \geq \dot{x}_1 + v_{allow} \qquad \text{(at } s\text{)}$$
$$\dot{x}_{int2} \geq \dot{x}_{d1} + a_{min}t' + v_{allow}, \qquad \text{(at } s'\text{)}$$

where $t' = \min(d_i + d_o, -\frac{\dot{x}_{d1}}{a_{min}})$.

But this is exactly equivalent to the second part of *safe-measure*$_d$ at s'!

We proved that in s', both parts of *safe-measure*$_d$ will be non-positive and, so, *safe-measure*$_d \leq 0$. Then the definition of D_2 guarantees that $s'.acc_{d2} = a_{min}$. And by the definition of D_o, $s.acc_2 = s'.acc_{d2}$. Thus, in *Delayed-Controller*$_2(D_2)$, $s.acc_2 = a_{min}$ as needed. ■

Lemma 6.5 *Let A_1 be any implementation of Controller$_1$, and let f be the identity relation on all the components of the $CV_I(A_1)$ system, except that $CV_I(A_1).x_{int2} = CV_D(A_1).x_2$, and $CV_I(A_1).\dot{x}_{int2} = CV_D(A_1).\dot{x}_2$. Then f is a forward simulation from the composed system $CV_D(A_1)$ to $CV_I(A_1)$.*

Proof: By induction on the number of steps in the hybrid execution.

Start States: Trivial, since all the restrictions on start states of C_2 are also restrictions on start states of *Delayed-Controller*$_2(D_2)$.

Discrete Steps: The only discrete steps are *collide*, e and the internal steps of A_1. The latter two steps cannot change any of the quantities involved. Since the *collide* step is the same for both automata, it respects the simulation relation. Also, the effects of the *collide* step satisfy Predicate S vacuously, thus the state reached after the *collide* action is a valid state of $CV_I(A_1)$.

Trajectories: Suppose that w_D is an I-trajectory of the delayed controller system $CV_D(A_1)$ and its first state s_D is reachable. Suppose that s_C is a reachable state of $CV_I(A_1)$ such that $(s_D, s_C) \in f$. Then let the corresponding hybrid execution fragment of $CV_I(A_1)$ consist of a single trajectory w_C, where $w_C(t) = w_D(t)$ (all variables have the same values). It is obvious that the two trajectories have the same hybrid trace and that the final states of both trajectories are f-related.

The only remaining thing to show is that w_C is in fact a trajectory allowed by C_2. By the definition of a trajectory of C_2 we must show that

1. w_C is a trajectory of *Controller*$_2$. This is trivial, since it is also a restriction on the trajectories of D_2, and the buffers do not change any of the values involved.

2. If *collided* = *false* in $w_C(0)$ then in all reachable states s of trajectory w_C, if *safe-measure* ≤ 0, then $s.acc_2 = a_{min}$. This follows directly from Lemma 6.4.

■

Theorem 6.6 *Delayed-Controller$_2$(D_2) is a correct controller for Init-Vehicles.*

Proof: We need to prove that for any implementation A_1 of *Controller$_1$*, $CV_D(A_1)$ implements the *safety* specification automaton *Safe-Vehicles*. $CV_D(A_1)$ is comparable to $CV_I(A_1)$, and, by Lemma 6.5, there exists a simulation relation from $CV_D(A_1)$ to $CV_I(A_1)$. So $CV_D(A_1)$ implements $CV_I(A_1)$.

$CV_I(A_1)$ implements *Safe-Vehicles* by Theorem 5.4. Thus, $CV_D(A_1)$ also implements *Safe-Vehicles* , which means that *Delayed-Controller$_2$*(D_2) is, in fact, *correct*, by definition 2. ■

6.4 Optimality of D_2

We give and prove necessary conditions for an implementation of *Spec-D_2* to be *correct*. We base the proofs on the fact that if an implementation B_2 of *Controller$_2$* is correct, then for any implementation B_1 of *Controller$_1$*, *Controlled-Vehicles*(*Init-Vehicles*, B_1, B_2) must implement *Controlled-Vehicles*(*Init-Vehicles*, B_1, *Necessary-C_2*) (see section 5.4). First, we define a new automaton, *Necessary-D_2*, which gives necessary conditions for *safety*. Then, we show that if for some implementation D of *Spec-D_2* there exists an implementation B_1 of *Controller$_1$* such that *Controlled-Vehicles*(*Init-Vehicles*, B_1, *Delayed-Controller$_2$*(D)) does not implement *Controlled-Vehicles*(*Init-Vehicles*, B_1, *Delayed-Controller$_2$*(*Necessary-D_2*)), then the *Delayed-Controller$_2$*(D) system will not be *correct*. This should be intuitively clear, since we only changed *safe-measure* to account for the "worst-case" (but possible) behavior of the vehicles during the last d_i+d_o time period of the delays. Relying on the fact that *Necessary-C_2* is necessary simplifies the proofs: we only need to show that a controller that would let *safe-measure$_d$* get negative, will eventually lead to a state in which *safe-measure* itself is negative. Then we can use necessity of *Necessary-C_2* to show that any such controller would not be *correct*.

Automaton 14(Necessary-D_2): This automaton is exactly like D_2, except that condition (2.1) for trajectories is slightly modified. In particular, *Necessary-D_2* commands maximum deceleration when *safe-measure$_d$* < 0, while D_2 does it when

Variables:

Input, Output, Internal: same as in D_2

Trajectories:

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

it satisfies condition 1 of D_2, plus

2. for all $t \in I$,

2.1. if *collided* = *false* and *safe-measure*$_d$ < *0* then $w(t).acc_2 = a_{min}$

conditions 2.2-2.4 of D_2 are satisfied

Figure 6-6: *Necessary-D_2* Hybrid Automaton

safe-measure $\leq$ *0*. The formal model is given in Figure 6-6; thus the only difference is the boundary in condition (2.1). We claim that this condition is necessary to guarantee *correctness*.

Definition 8(Predicate S_d): If *collided* = *false* then *safe-measure*$_d \geq 0$.

We now define Bad_d controllers, similar to the Bad controllers in the ideal case in the last chapter.

Definition 9(Bad_d): Let B_1 and B_{d2} be implementations of *Controller*$_1$ and *Spec-D_2*, respectively. Then B_1 is Bad_d for B_{d2} if and only if in the system *Controlled-Vehicles*(*Init-Vehicles*, B_1, *Delayed-Controller*$_2$(B_{d2})) there exists a reachable state s that violates Predicate S_d. B_{d2} is Bad_d if and only if there exists some B_1 that is Bad_d for this B_{d2}.

Definition 10(VeryBad$_d$): Let an implementation B_1' of *Controller*$_1$ be called **VeryBad**$_d$ for an implementation D of *Controller*$_2$ if

1. B_1' is Bad_d for B_{d2};

2. In any execution of Controlled-Vehicles(Init-Vehicles, B_1', B_{d2}), for any state s that does not satisfy Predicate S, strictly after the occurrence of s, $acc_1 = a_{min}$.

Based on the above definitions, we state an existence lemma, very similar to Lemma 5.5.

Lemma 6.7 *If B_{d2} is Bad_d then there exists an implementation B_1' of Controller$_1$, such that B_1' is VeryBad for B_{d2}.*

Proof: The proof is identical to proof of a similar Lemma 5.5 in the previous Chapter. ■

Next, we prove that if an implementation of *Spec-D_2* does not implement D_2 then it must be Bad_d.

Lemma 6.8 *Let B_{d2} be an implementation of Spec-D_2. If there exists an implementation B_1 of Controller$_1$ such that Controlled-Vehicles(Init-Vehicles, B_1, Delayed-Controller$_2$(B_{d2})) does not implement Controlled-Vehicles(Init-Vehicles, B_1, Delayed-Controller$_2$(Necessary-D_2)) then B_{d2} is Bad_d.*

Proof: Identical to that of Lemma 5.6. ■

Lemma 6.9 relates the Bad_d and Bad terms.

Lemma 6.9 *Let B_{d2} be an implementation B_{d2} of Spec-D_2. If B_{d2} is Bad_d, then Delayed-Controller$_2$(B_{d2}) is Bad.*

Proof: Let B_{d2} be a Bad_d implementation of *Spec-D_2*. Then, by Lemma 6.7, there must exist an implementation B_1' of *Controller$_1$* that is *VeryBad* for B_{d2}. We need to prove that B_1' is *Bad* for *Delayed-Controller$_2$*(B_{d2}), and not only Bad_d for B_{d2}. In particular, we need to show that there exists a reachable state of *Controlled-Vehicles*(*Init-Vehicles*, B_1', *Delayed-Controller$_2$*(B_{d2})) that violates Predicate S, and not only Predicate S_d (violation of Predicate S_d follows from the definition of Bad_d).

Let's look at a reachable state s of this *Controlled-Vehicles* system that violates Predicate S_d; existence of this state follows from the fact that B_{d2} is Bad_d. For such s, *collided* = *false* and *safe-measure$_d$* < 0, meaning that

$$x_{d1} + \dot{x}_{d1}t' + \frac{a_{min}t'^2}{2} - (x_{int2} + l_2) < -\frac{(\dot{x}_{int2})^2 - (\dot{x}_{d1} + a_{min}t')^2 - (v_{allow})^2}{2a_{min}} \tag{6.3}$$

and

$$\dot{x}_{d1} + a_{min}t' < \dot{x}_{int2} - v_{allow}, \tag{6.4}$$

where $t' = \min(-\frac{\dot{x}_{d1}}{a_{min}}, d_i + d_o)$.

By the definition of B_1', it always decelerates at the maximum allowable rate, a_{min}. Let's look at a state s' reachable from s in time d_o, such that $s'.now = s.now + d_o$. Then in s',

$$s'.\dot{x}_1 = s.\dot{x}_{d1} + a_{min}t',$$

$$s'.x_1 = s.x_{d1} + s.\dot{x}_{d1}t' + \frac{1}{2}a_{min}t'^2,$$

$$s'.\dot{x}_2 = s.\dot{x}_{int2} \text{ and } s'.x_2 = s.x_{int2},$$

where $t' = \min(-\frac{\dot{x}_{d1}}{a_{min}}, d_i + d_o)$. The first 2 equations follow from the fact that position and velocity are integrals of acceleration (Property 2 of *Vehicles*); the last 2 follow from Theorem 6.3. Note that these equalities also hold in the initial d_i time interval, because our data in the inbound delay buffer assumes that the first controller is decelerating at the maximum rate.

Substituting these equations into the above inequalities will yield the two parts of *safe-measure* in state s'; moreover, both parts turn out to be negative, which means that in state s', *safe-measure* < 0. But *safe-measure* < 0 means that Predicate S is violated, and, therefore, *Delayed-Controller*$_2(B_{d2})$ is *Bad*. ■

Since we have just shown that the delayed controller (composed with delay buffers) implements the non-delayed one, we can use the necessity property of the ideal case controller, to easily prove the necessity of the delayed controller:

Theorem 6.10 *Let B_{d2} be an implementation of Spec-D_2. If there exists an implementation B_1 of Controller$_1$ such that Controlled-Vehicles(Init-Vehicles, B_1, Delayed-Controller$_2$(B_{d2})) does not implement Controlled-Vehicles(Init-Vehicles, B_1, Delayed-Controller$_2$(Necessary-D_2)) then Delayed-Controller$_2$(B_{d2}) is not correct.*

Proof: B_{d2} must be Bad_d, by Lemma 6.8. By Lemma 6.9, *Delayed-Controller*$_2(B_{d2})$ is *Bad*. But then, by Theorem 5.9, there exists an implementation B_1 of *Controller$_1$*,

such that a system *Controlled-Vehicles*(*Init-Vehicles*, B_1, *Delayed-Controller*$_2(B_{d2})$) has a reachable state that violates *safety*. Since this *Controlled-Vehicles* system violates *safety*, it does not implement *Safe-Vehicles*, which means that *Delayed-Controller*$_2(B_{d2})$ is not *correct*, by Definition 2. ■

Theorem 6.6 proves that *Delayed-Controller*$_2(D_2)$ is sufficient, and Theorem 6.10 proves that *Delayed-Controller*$_2$(*Necessary-*D_2) is necessary to guarantee *correctness*. Since the distinction between D_2 and *Necessary-*D_2 is very small, they can serve as the *correctness* specification.

Chapter 7

Uncertainty

Our model already includes both the inbound and the outbound delays in sending and receiving information between the controller and *Init-Vehicles*. We introduce extra complexity which makes the model even more realistic: the inbound and outbound uncertainty (inexactness) in information. The inbound uncertainty is the maximum absolute difference between the actual position and velocity data of the vehicle and the data reported by the sensors to the controller; it arises from inexact sensors that communicate data to the controllers. The outbound uncertainty is the maximum absolute difference between the acceleration commanded by the controller and the acceleration actually implemented by the vehicle; it is due to the inherent inexactness in the performance of the brakes and accelerators.

We use similar methods to the ones used in the delay case. A special "uncertainty buffer" automaton is defined, similar to the previous *D-Buffer* automaton. We use two instances of this parameterized buffer to get the inbound and outbound uncertainty. Then, we compose these two buffers with the modified controller that accounts for uncertainties, and prove that this new composed controller is sufficient to guarantee safety. The proof uses the fact that the $Delayed\text{-}Controller_2(D_2)$ is sufficient. This use of levels of abstraction makes the proofs for this complicated case, involving both delays and uncertainties, easier to write and understand.

7.1 The Uncertainty Buffer

We introduce a parameterized uncertainty buffer, similar in function to the delay buffer.

Parameters:

n - the number of input variables

S_i, S_o - two disjoint sets of variables with n members in each set

Let V be a valuation function, same as in *D-Buffer*

$var : S_i \rightarrow S_o \times \mathsf{R}^+$, with selectors v_o and Δu

Variables:

Input: S_i

Output: S_o

Trajectories:

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

1. for all $v_o \in S_o$, $v_o(t)$ is an integrable function
2. for all $t \in I$, $t > 0$ the following holds for all $v_i \in S_i$:

$V(var(v_i).v_o) \in [V(v_i) - var(v_i).\Delta u, V(v_i) + var(v_i).\Delta u]$

Figure 7-1: *U-Buffer* (n, S_i, S_o, var) Hybrid I/O Automaton

Automaton 15(U-Buffer(n, S_i, S_o, var)): The *U-Buffer* automaton nondeterministically perturbs all input variables within given bounds to produce output variables.

Parameters

- n is the number of input variables;
- S_i, S_o are two disjoint sets of input and output variables, respectively;
- $var : S_i \rightarrow S_o \times \mathsf{R}^+$ is a 1-1 mapping from input to pairs consisting of an output variable and an uncertainty bound. So, the input variable v_i becomes output variable $var(v_i).v_o$, with maximum uncertainty $var(v_i).\Delta u$, where $v_i \in S_i, v_o \in S_o, \Delta u \in \mathsf{R}^+$.

Variables The input and output variables are parameterized by the S_i and S_o sets. It does not matter for the generalized automaton which variables it perturbs.

Trajectories The first restriction guarantees that all the data is still integrable after the uncertainty buffer. This is important, because both the controllers and the *Init-Vehicles* automata integrate the data from the buffers to obtain position and velocity data. Condition (2) lets outputs vary within given bounds of the inputs. The bounds are given by the $var(v_i).\Delta u$ function and represent the maximum absolute value of uncertainty in the data for variable v_i.

Theorem 7.1 *In all reachable states s of U-Buffer, for all $v_i \in S_i$,*

1. $s.V(var(v_i).v_o) \geq s.V(v_i) - var(v_i).\Delta u$,
2. $s.V(var(v_i).v_o) \leq s.V(v_i) + var(v_i).\Delta u$.

Proof: By restriction (2) on trajectories of *U-Buffer*. ■

7.2 The System

The controller D_2 is implemented by a composition of three hybrid automata: another controller U_2 and two instances of the uncertainty buffer — the inbound and outbound uncertainty buffers.The composed system is called *Uncertain-Controller*$_2$ (see Figure 7-2).

We define two instances of the *U-Buffer* automaton – the inbound and outbound uncertainty buffers. These buffers use the following constants:

- δ - the maximum absolute uncertainty in position data;
- $\dot{\delta}$ - the maximum absolute uncertainty in velocity data;
- $\ddot{\delta}$ - the maximum absolute uncertainty in acceleration.

Automaton 16(U_i): U_i = *U-Buffer*(3, $\{x_{d1}, \dot{x}_{d1}, \ddot{x}_{d1}\}$, $\{x_{u1}, \dot{x}_{u1}, \ddot{x}_{u1}\}$, var), where $var(x_{d1}) = (x_{u1}, \delta)$, $var(\dot{x}_{d1}) = (\dot{x}_{u1}, \dot{\delta})$, and $var(\ddot{x}_{d1}) = (\ddot{x}_{u1}, \ddot{\delta})$. This automaton inputs the delayed position, velocity and acceleration data from the inbound delay buffer D_i, and outputs them with "uncertainty" to the controller U_2.

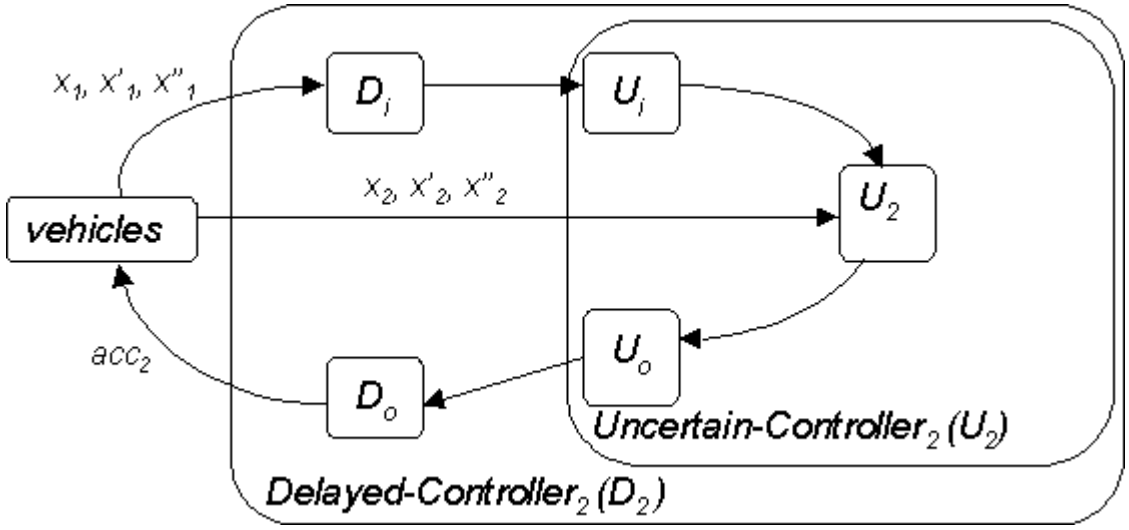

Figure 7-2: Sensor-Uncertainty Vehicles Model

Automaton 17(U_o): U_o = *U-Buffer*(1, $\{acc_{u1}\}$, $\{acc_{d1}\}$, var), where $var(acc_{u1}) = (acc_{d1}, \ddot{\delta})$. The outbound uncertainty buffer automaton inputs the acceleration commands from the controller U_2, and outputs the perturbed values to the outbound delay buffer D_o.

In the uncertainty case we give a specific controller U_2 that accounts for uncertainties, and prove that it is sufficient to guarantee correctness. No specification controller, or parameterized composition automaton is given in this case — optimality results are not presented, and only sufficiency of a specific controller is proved. This controller is presented below.

Automaton 18(U_2): The controller U_2 (shown in figure 7-3) is the same as D_2 except that:

1. Input and output variables communicate through uncertainty buffers; the a_2 buffers remembers the new output variable acc_{u2}.

2. $\dot{x}_{int2}$ is redefined to account for the "worst" possible uncertainty in the brake performance, i.e., it assumes that the vehicle accelerates at $acc_{u2} + \ddot{\delta}$.

3. $safe\text{-}measure_u$ is defined to account for the uncertainties.

Definition:

$$\begin{aligned} \textit{safe-measure}_u = \max(((x_{u1} - \delta) + (\dot{x}_{u1} - \dot{\delta})t'' + \tfrac{a_{min}t''^2}{2}) - (x_{int2} + \dot{x}_{int2}\,d_o + \tfrac{\ddot{\delta}d_o^2}{2} + l_2) \\ + \tfrac{(\dot{x}_{int2} + \ddot{\delta}d_o)^2 - (\dot{x}_{u1} - \dot{\delta} + a_{min}t'')^2 - (v_{allow})^2}{2a_{min}}, \\ (\dot{x}_{u1} - \dot{\delta}) + a_{min}t'' - (\dot{x}_{int2} + \ddot{\delta}d_o) + v_{allow}), \end{aligned}$$

where $t'' = \min(d_i + d_o, -\frac{\dot{x}_{u1} + \dot{\delta}}{a_{min}})$

Variables:

Input: $x_{u1}, \dot{x}_{u1} \in \mathsf{R}^{\geq 0}$, $\ddot{x}_{u1} \in \mathsf{R}$

$x_2, \dot{x}_2 \in \mathsf{R}^{\geq 0}$, $\ddot{x}_2 \in \mathsf{R}$

Output: acc_{u2}, initially if $\textit{safe-measure}_u \leq 0$, then $acc_{u2} = a_{min} - \ddot{\delta}$,

else arbitrary, where $acc_{u2} \geq a_{min}$

Internal: a_2 - maps from an interval $[0, d_o]$ to R,

initially, $\forall t \in [0, d_o], a_2(t) = a_{min} - \ddot{\delta}$

x_{int2}, $\dot{x}_{int2}$ - the position and velocity of the second vehicle after time d_o passes,

provided $\textit{collided} = \textit{false}$;

initially, $\dot{x}_{int2} = \dot{x}_2 + a_{min}t, x_{int2} = x_2 + \dot{x}_2 t + \frac{a_{min}t^2}{2}$, where $t = \min(d_o, -\frac{\dot{x}_2}{a_{min}})$.

Trajectories:

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

1. w is a trajectory of $\textit{Controller}_2$
2. if $\textit{collided} = \textit{false}$ in $w(0)$ then for all $t \in I$, $t > 0$:
   - 2.1. if in $w(t)$, $\textit{safe-measure}_u \leq 0$ then $acc_{u2} = a_{min} - \ddot{\delta}$ else $acc_{u2} \geq a_{min}$
   - 2.2. $\forall t' \in [0, d_o]$,
     $$w(t).a_2(t') = \begin{cases} w(0).a_2(t' - t) & \text{if } t' > t \\ w(t - t').acc_{d2} & \text{otherwise} \end{cases}$$
   - 2.3. $w(t).\dot{x}_{int2} = w(t).\dot{x}_2 + \int_0^{d_o}(w(t).a_2(u) + \ddot{\delta})du$
   - 2.4. $w(t).x_{int2} = w(t).x_2 + \int_0^{d_o} w(t).\dot{x}_{int2}du$

Figure 7-3: U_2 Hybrid I/O Automaton

The only changes from *safe-measure*$_d$ are that the first vehicle's data is adjusted to the "worst possible" uncertainty in the behavior of the first vehicle. This "worst possible" behavior is defined by the following inequalities:

1. $x_{d1} \geq x_{u1} - \delta$;
2. $\dot{x}_{d1} \geq \dot{x}_{u1} - \dot{\delta}$;
3. $x_{int2}^{D_2} \leq x_{int2}^{D_2} + \dot{x}_{int2}^{U_2} d_o + \frac{\ddot{\delta} d_o^2}{2}$;
4. $\dot{x}_{int2}^{D_2} \leq \dot{x}_{int2}^{U_2} + \ddot{\delta} d_o$;

These inequalities are used in changing *safe-measure*$_d$ to *safe-measure*$_u$. The proof of these relationships is given below, in Theorem 7.2.

Automaton 19(Uncertain-Controller$_2$): *Uncertain-Controller*$_2$ = VarHide($\{x_{u1}$, $\dot{x}_{u1}$, $\ddot{x}_{u1}$, $acc_{u1}\}$, $U_i \parallel U_2 \parallel U_o$), is the composition of the uncertainty buffers with the new controller. We show that it implements D_2in the context of the *Controlled-Vehicles* system. This automaton is not parameterized by the choice of the controller, unlike the previous composed controllers. No parameterization is necessary because only sufficiency of the specific controller U_2 is proven.

Theorem 7.2 *Let A_1 be any implementation of Controller$_1$. Then, in any reachable state s of the Controlled-Vehicles(Vehicles, A_1, Delayed-Controller$_2$(Uncertain-Controller$_2$)) system such that $s.collided = false$, the following hold:*

1. $acc_{d2} \leq acc_{u2} + \ddot{\delta}$;
2. $x_{d1} \geq x_{u1} - \delta$;
3. $\dot{x}_{d1} \geq \dot{x}_{u1} - \dot{\delta}$;
4. $\ddot{x}_{d1} \geq \ddot{x}_{u1} - \ddot{\delta}$;
5. $x_{int2}^{D_2} \leq x_{int2}^{U_2} + \dot{x}_{int2}^{U_2} d_o + \frac{\ddot{\delta} d_o^2}{2}$;
6. $\dot{x}_{int2}^{D_2} \leq \dot{x}_{int2}^{U_2} + \ddot{\delta} d_o$.

Proof: Claim (1) follows from Property 2 of *U-Buffer*; claims (2)-(4) follow from Property 1 of *U-Buffer*. Finally, claims (5) and (6) follow from claim (1) of this theorem and Property 2 of *Vehicles* (position and velocity are integrals of acceleration). ■

Note that the symmetric properties also hold, but these are the relationships that are used later on in the proofs.

7.3 Correctness of U_2

As in the delayed case, we want to simulate the previous delayed system using the new uncertain system, and thus show that the new controller is sufficient.

Throughout this section we will use the following notation: for any implementation A_1 of $Controller_1$, let

$CV_U(A_1) = Controlled\text{-}Vehicles(Init\text{-}Vehicles, A_1, Delayed\text{-}Controller_2(Uncertain\text{-}Controller_2))$, and

and $CV_D(A_1) = Controlled\text{-}Vehicles(Init\text{-}Vehicles, A_1, Delayed\text{-}Controller_2(D_2))$.

First we show that if the old $safe\text{-}measure_d$ (the one used in the delayed case) is non-positive in some state of $Uncertain\text{-}Controller_2$, then the new controller U_2 (the one that has inbound and outbound uncertainty), also outputs maximum deceleration. Formally,

Lemma 7.3 *Let A_1 be any implementation of $Controller_1$, and let s be a reachable state of the $CV_U(A_1)$ system, such that $s.collided = false$ and $safe\text{-}measure_d \leq 0$. Then, $safe\text{-}measure_u \leq 0$ and $s.acc_{u2} = a_{min} - \ddot{\delta}$.*

Proof: Initially, the lemma is true by restriction on initial conditions of $Uncertain\text{-}Controller_2$. Consider any reachable state s of $CV_U(A_1)$, such that $s.collided = false$ and $safe\text{-}measure_d \leq 0$. At s we have

$$(x_{d1} + \dot{x}_{d1}t' + \frac{a_{min}t'^2}{2}) - (x_{int2}^{D_2} + l_2) + \frac{(\dot{x}_{int2}^{D_2})^2 - (\dot{x}_{d1} + a_{min}t')^2 - (v_{allow})^2}{2a_{min}} \leq 0 \tag{7.1}$$

and

$$\dot{x}_{d1} + a_{min}t' + v_{allow} - \dot{x}_{int2}^{D_2} \leq 0 \tag{7.2}$$

where $t' = \min(d_i + d_o, -\frac{\dot{x}_{d1}}{a_{min}})$.

Let $t'' = min(d_i + d_o, -\frac{\dot{x}_{int2}^{D_2} + \dot{\delta}}{a_{min}})$. Then, by Theorem 7.2, in s,

1. $x_{d1} \geq x_{u1} - \delta$;
2. $\dot{x}_{d1} \geq \dot{x}_{u1} - \dot{\delta}$;
3. $x_{int2}^{D_2} \leq x_{int2}^{U_2} + \dot{x}_{int2}^{U_2} d_o + \frac{\ddot{\delta} d_o^2}{2}$;
4. $\dot{x}_{int2}^{D_2} \leq \dot{x}_{int2}^{U_2} + \ddot{\delta} d_o$.

Using the above inequalities, we replace each delayed variable (x_{d1}, $\dot{x}_{d1}$, $x_{int2}^{D_2}$, $\dot{x}_{int2}^{D_2}$) in inequalities 7.1 and 7.2 with an expression that is smaller than the delayed variable, and using only the "uncertain" values, which are the ones known to U_2. Then, we get exactly the two parts of *safe-measure*$_u$; moreover, since we used only smaller values, the resulting expressions are still non-positive. So,

$$\textit{safe-measure}_u \leq 0$$

Then, by the definition of U_2, $acc_{u2} = a_{min} - \ddot{\delta}$, as needed. ■

Lemma 7.4 *Let A_1 be any implementation of Controller$_1$, and let f be an identity relation on all state components of $CV_D(A_1)$, except that $acc_{d2}^{CV_D} = \max(acc_{u2}^{CV_U}, a_{min})$. Then f is a simulation from $CV_U(A_1)$ to $CV_D(A_1)$.*

Proof: By induction on the number of steps in the hybrid execution.

Start States: The restrictions on start states of $CV_D(A_1)$ and $CV_U(A_1)$ are identical.

Discrete Steps: The only discrete steps are *collide,* e and the internal steps of A_1. The latter two steps cannot change any of the quantities involved. Since the *collide* step is the same for both automata, it respects the simulation relation. Also, the effects of the *collide* step satisfy Predicate S_d vacuously, thus the state reached after the *collide* action is a valid state of $CV_D(A_1)$.

Trajectories: Suppose that w_U is an I-trajectory of the uncertainty-buffered system $CV_U(A_1)$ and its first state s_U is reachable. Suppose that s_D is a reachable state of $CV_D(A_1)$ such that $(s_U, s_D) \in f$. Then let the corresponding hybrid execution fragment of $CV_D(A_1)$ consist of a single trajectory w_D, where all the state components in all the states of w_D are equal to corresponding components in w_U, except that $w_D.acc_{d2} = \max(w_U.acc_{d2}, a_{min})$. It is clear that the two trajectories have the same hybrid trace and that the final states of both trajectories are f-related.

The only remaining thing to show is that w_D is in fact a trajectory of $CV_D(A_1)$. In particluar, we must show that the projections of w_D on the components of $CV_D(A_1)$ are allowed by these components.

First, we show that w_D is allowed by the *Delayed-Controller*$_2(D_2)$ controller. By the definition of a trajectory we must show that

1. w_D is allowed by *Controller*$_2$.

   This is trivial, since it is also a restriction on the trajectories of U_2, and the buffers preserve these conditions by preserving integrability.

2. If *collided* = *false* in $w_D(0)$ then $\forall t \in I$ such that *safe-measure*$_d \leq 0$ we have $w_D(t).acc_{d2} = a_{min}$.

   Consider the trajectory w_U of $CV_U(A_1)$. Since *safe-measure*$_d$ uses the same variables with the same values in both the $CV_D(A_1)$ and the $CV_U(A_1)$ systems, we can apply Lemma 7.3, so that $w_U(t).acc_{u2} = a_{min} - \ddot{\delta}$. By Theorem 7.2, in all reachable states s of $CV_U(A_1)$, $s.acc_{d2} \leq s.acc_{u2} + \ddot{\delta}$, so $w_U(t).acc_{d2} \leq a_{min}$. Then, using the definition of trajectory w_D,

$$w_D(t).acc_{d2} = \max(w_U(t).acc_{d2}, a_{min}) = a_{min},$$

as needed.

We also show that the projection of w_D on *Init-Vehicles* is allowed by *Init-Vehicles*, and has the same hybrid trace as w_U.

We know that w_U is allowed by *Init-Vehicles*. But w_D is exactly the same as w_U, except for the input variable acc_2. Also, since all the variables are the same, the hybrid trace of w_D and w_U is the same. Thus, we only need to show that the condition (3) of *Init-Vehicles* trajectories, namely, that $w_D.\ddot{x}_2 = \max(w_D.acc_2, a_{min})$, is preserved.

By the definition of w_D, $w_D(t).\ddot{x}_2 = w_U(t).\ddot{x}_2$ for all t throughout the trajectory. By restriction (3) on trajectories of *Init-Vehicles*, $w_U(t).\ddot{x}_2 = \max(w_U(t).acc_2, a_{min})$. Also, by the definition of w_D, $w_D(t).acc_2 = \max(w_U(t).acc_2, a_{min})$. Putting these equations together we get

$$w_D(t).\ddot{x}_2 = w_U(t).\ddot{x}_2 = \max(w_U(t).acc_2, a_{min}) = w_D(t).acc_2 = \max(w_D(t).acc_2, a_{min}).$$

Therefore, w_D is a valid trajectory of *Init-Vehicles*.

■

Theorem 7.5 *Delayed-Controller$_2$(Uncertain-Controller$_2$) is a correct controller for Init-Vehicles.*

Proof: We need to prove that for any implementation A_1 of *Controller$_1$*, $CV_U(A_1)$, implements *Safe-Vehicles*. By Lemma 7.4, there is a simulation relation f from $CV_U(A_1)$ to $CV_D(A_1)$. Since $CV_U(A_1)$ and $CV_D(A_1)$ are comparable, $CV_U(A_1)$ implements $CV_D(A_1)$.

By Theorem 6.6, *Delayed-Controller$_2$*(D_2) is *correct*, which means that $CV_D(A_1)$ implements *Safe-Vehicles*. Then, since $CV_U(A_1)$ implements $CV_D(A_1)$, $CV_U(A_1)$ also implements *Safe-Vehicles*. It follows that *Delayed-Controller$_2$*(*Uncertain-Controller$_2$*) is *correct*. ■

7.4 Optimality

We do not present a necessary controller here, as opposed to the controllers considered in the previous chapters. Before, we always made *safe-measure* to account for the "worst-case", but possible, conditions. Then, we were able to prove necessity by making the first vehicle have its worst possible behavior. However, with the uncertainty involved, we did not make *safe-measure* as tight as possible. The problem is that in our analysis, the position and velocity data are used independently. However, the position and velocity data are dependent upon each other, and thus we could use the relationship between the two values to get tighter approximations to their real values, resulting in a more optimal controller. We did not model it this way because it is not realistic: in most situations it is impractical to deduce the tighter bounds, since just calculating these bounds takes too much time, eliminating any benefits obtained from using tighter bounds. This would ultimately decrease the performance, instead of improving it.

Chapter 8

Conclusion and Future Work

The system consisting of two vehicles moving on a single track has been modeled using hybrid automata, including all the components (physical vehicles, controllers, delay and uncertainty buffers), and the interactions among 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 levels of abstraction, significantly simplifying the proofs.

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

There are two important results of this research. Generally, it demonstrates the power of the hybrid automata model, the associated proof methods in reasoning about interesting hybrid systems, and the use of abstraction levels as a way of handling complexity. More specifically, we give a reusable model of the automated vehicles, including their controllers and sensors, which incorporates delays and uncertainties directly, and we derive and prove necessary and sufficient conditions for satisfying the safety requirement of the vehicles.

Future work will address the following problems:

1. In reality, the controllers can only control the jerk, and not the acceleration of the vehicles. Without further complicating the models, we can still model the controllers as controlling the acceleration, but the outbound delay (and, possibly, uncertainty) have to be increased to account for the fact that it takes some time for the controller to reach desired acceleration. Using this approach, the outbound delay and/or uncertainty might become functions of current acceleration, and not constants.

2. We have developed necessary and sufficient conditions for a controller to guarantee safety in the presence of delays and uncertainties. This controller can now serve as a correctness specification. We could prove correctness of "real" algorithms, for merging or splitting, by testing them with our controller.

3. In this thesis we only handled the first collision, when in fact, even in the case of two vehicles, multiple collisions can occur. Although all of the models do not have to be changed, most of the proofs would have to be reworked to handle this case. This would be similar to Lygeros and Lynch's work in [10], but would have more detailed vehicle models, including delays and uncertainties.

4. Finally, it would be interesting to extend the models and the proofs to the multiple vehicle case. If we limit the analysis to only pairwise collisions (excluding simultaneous collisions of three or more vehicles), then the models and some of the analysis from this thesis can be reused, but many new problems would arise. In order to remove the pairwise-only collisions restriction, some of the models would have to be reworked to model these collisions. The results would be more general than in [10], since delays and uncertainties would be included in the model.

Bibliography

[1] *Second European Workshop on Real-Time and Hybrid Systems.* Grenoble, France, May 1995.

[2] 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.

[3] J. W. de Bakker et al., editors. *Real-Time: Theory in Practice* (REX Workshop, Mook, The Netherlands, June 1991), volume 600 of *Lecture Notes in Computer Science*. Springer-Verlag, 1992.

[4] Jonathan Frankel, Luis Alvarez, Roberto Horowitz, and Perry Li. Robust platoon maneuvers for AVHS. Manuscript, Berkeley, November 10, 1994.

[5] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[6] 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.

[7] John Lygeros. *Hierarchical Hybrid Control of Large Scale Systems.* PhD thesis, University of California, Department of Electrical Engineering, Berkeley, California, 1996.

[8] 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.

[9] John Lygeros and Nancy Lynch. On the formal verification of the TCAS conflict resolution algorithms. In *36th IEEE Conference on Decision and Control*, pages 1829–1834, San Diego, CA, December 1997. Extended abstract.

[10] 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.

[11] 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.

[12] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O automata. Journal version in progress.

[13] 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.

[14] 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.

[15] Oded Maler, editor. *Hybrid and Real-Time Systems* (International Workshop, HART'97, Grenoble, France, March 1997), volume 1201 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

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

[17] 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.

[18] 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.