Correctness of Vehicle Control Systems: A Case Study

by

Henri B. Weinberg

B.S., Computer Science
Yale University, 1992

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

Master of Science in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1996

Author ....

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

Accepted by ....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Correctness of Vehicle Control Systems: A Case Study

by
Henri B. Weinberg

Submitted to the Department of Electrical Engineering and Computer Science
on March 20, 1996, in partial fulfillment of the
requirements for the degree of
Master of Science in Electrical Engineering and Computer Science

Abstract

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

One of the formal tools that has been developed is the *hybrid I/O automaton* (HIOA) model [1]. In this case study, we show how this model can be used to specify and verify part of an automated transportation system — a vehicle deceleration maneuver. We investigate how techniques such as automata composition, invariant assertions, and simulation mappings can be applied to systems of communicating digital and analog components. The purpose of the case study is to test the applicability of these computer science based techniques to the area of automated transit. In particular, we are concerned that HIOA techniques express hybrid systems faithfully and that they allow clear and scalable proofs of significant properties of these systems.

In the deceleration maneuver, digital controller slows a train to a target velocity range within a given distance. We examine four versions of the deceleration maneuver, each with a different model of the communication between controller and train: plain, delay, feedback, and feedback with delay. For each case we give a model of the non-controller portion of the system, define correctness of a controller, give an example of a correct controller, and prove that it is correct. This case study contains full proofs of the correctness of the various controllers. However, some of the proofs are only sketched, when similar formal proofs appear in other chapters.

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

Acknowledgments

Thesis supervisor seems a title too antiseptic for Nancy Lynch who gave so generously of herself in the effort to produce this thesis. I have grown and learned under her guidance more than these pages can tell. She and the members of her Theory of Distributed Systems group provided the friendly and stimulating environment that fostered my work. I am especially grateful to Victor Luchangco, Anna Pogosyants, and Rainer Gawlick for their daily advice, support, and friendship.

My research is supported in part by a National Science Foundation graduate fellowship.

I would like to thank my family — my parents, Emil and Caroline, my brothers, Misha and Peter, and above all, my wife, Meg — for their unswerving belief in me.

To the memory of Anna Pogosyants.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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

One of the formal tools that has been developed is the *hybrid I/O automaton* model [1]. In this case study, we show how this model can be used to specify and verify part of an automated transportation system — a vehicle deceleration maneuver. We investigate how techniques such as automata composition, invariant assertions, and simulation mappings can be applied to systems of communicating digital and analog components. The purpose of the case study is to test the applicability of these computer science based techniques to the area of automated transit. In particular, we are concerned that HIOA techniques express hybrid systems faithfully and that they allow clear and scalable proofs of significant properties of these systems.

Formal Framework

The hybrid I/O automaton model is an extension of the *timed I/O automaton* model of [2, 3, 4, 5] inspired by the phase transition system model of [6] and the similar hybrid system model of [7]. A hybrid I/O automaton (HIOA) is a (possibly) infinite state labeled transition system. The states of a HIOA are the valuations of a set of variables. Certain states are distinguished as start states. The transitions of a HIOA are of two types: continuous and discrete. A HIOA's discrete transitions are labeled with actions. Both the variables and the actions of a HIOA are partitioned into three categories: *input*, *output*, and *internal.* A *hybrid execution* of a HIOA is a sequence of transitions that describes a possible behavior of the system over time. A *hybrid trace* of a HIOA is the externally visible part of an execution (i.e. the non-internal part).

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

As has been the case in previous work with timed I/O automata, most of the proofs in this HIOA based case study use *invariant assertions* and *simulations*. An assertion is a predicate on states; an invariant assertion is one that is true in every reachable state. Invariant assertions are usually proved by induction on the length of an execution. A simulation is a mapping between states of two HIOA that can be used to show that that one HIOA implements another. The proof that a given mapping is a simulation is another form of induction on the length of an execution of the implementation; the induction matches individual steps in the implementation with corresponding steps or sequences of steps in the specification. Even proofs of timing properties can be performed using these techniques; the key idea is to build timing information into the state where it can be tested by assertions.

This type of formalism has several benefits. First, the inductive structure and stylized nature of the proofs makes them easy to write, check, and understand. In some cases, this structure has allowed the proofs to be checked using automated theorem proving techniques. Second, the implementation relation allows the description of a system at different levels of abstraction. Assertions proved on the high level models extend to the lower level models via the simulation mapping. This hierarchy helps manage the complexity of the overall specification and it helps simplify the proofs because assertions are usually easier to prove on the more abstract models. Third and finally, the methods are not completely automatic. They require the user to supply invariants and simulations, which serve as useful documentation of the system. In an exploratory work such as this case study, the insight gained through a manual process is particularly useful because it may lead to developments in the underlying models and methods.

The Deceleration Maneuver

Typical examples of automated transportation systems include the Raytheon Personal Rapid Transit System and the California PATH project [8, 9, 10]. In these hybrid systems, a number of computer controlled vehicles share a network of tracks or highways. The digital part of the system is the computer vehicle controller and the analog part of the system is the vehicle, its engine, the guideway, and so forth. In [8] the control of the transportation system is described hierarchically. The higher levels of such a hierarchical system coordinate and determine strategy while the lowest level performs specific maneuvers.

This case study focuses on a single maneuver: the task of decelerating a vehicle to a target speed within a certain distance. Such a maneuver might be invoked, for example, when a vehicle is approaching an area whose maximum allowable velocity is lower than the vehicle's current velocity. We model a vehicle and its controller as two communicating HIOAs. We do not model the invocation of the maneuver nor do we investigate either complex vehicle physics or complex control schemes. Instead we have considered four variations on the communication between vehicle and controller. The four variations arise from the inclusion or exclusion of two parameters: feedback and delay. The first case is the simplest: no feedback and no delay. The second case introduces a communication delay between the controller and the vehicle. The third case introduces feedback without delay; the vehicle periodically sends sensory information to the controller. The fourth case involves both feedback and delay. For each case, we give a formal specification of what it means for a controller to correctly implement the deceleration maneuver, then we give an example implementation of such a controller and formally verify that it correctly implements the maneuver.

Related Work

This case study is part of a long-term project in the M.I.T. Theory of Distributed Systems research group on modeling, verifying, and analyzing problems arising in automated transit systems. A survey of the project appears in [11]. The case study, [12, 13], examines the train and gate problem from traditional railroad control. In [14], the author uses abstraction to relate continuous and discrete control of a vehicle maneuver. Safety systems for automated transit are examined in [15].

The development of models and verification methods for timing-based systems is an active research area within computer science. The timed I/O automaton model is similar, for example, to a model of Alur and Dill [16], to one of Lamport [17] and to one of Henzinger, Manna and Pnueli [18]. In contrast to those formalisms, the development and use of the timed I/O automaton model has focused on compositional properties [19], implementation relations [20], and semi-automated proof checking [21] with less emphasis on syntactic forms, temporal logics, and fully automatic analysis. Just as timed I/O automata have been extended to hybrid I/O automata to treat hybrid systems, so have other real-time models. For example, the timed transition system model of [18] is extended to the phase transition system model in [6]. Phase transition systems are analogous to hybrid I/O automata: their transitions correspond to our discrete steps; their activities correspond to our trajectories. The hybrid system model of [7] is similar to the phase transition system model except that it includes synchronization labels that correspond to our actions. This allows a notion of parallel composition in the hybrid system model. The hybrid system model differs from the

HIOA model because it has no input/output distinction on either labels (actions) or variables.

The methods of invariant assertions, abstraction mappings, forward and backward simulations, history and prophecy variables are used in many places in computer science. We will not attempt to attribute all these notions. An overview of these methods, for untimed and timed systems, appears in [22, 2, 3].

Roy Johnson and Steve Spielman at Raytheon are leading the design and development of a prototype advanced personal rapid transit system, based partly on concepts developed by Dr. Edward Anderson of the Taxi2000 Corp. Prof. Shankar Sastry and his colleagues at Berkeley have studied intelligent highway systems [8, 9, 10] and specific scenarios that arise therein. For example, they have considered equipping cars with "smart" cruise controls that can adapt to other cars in the vicinity [9]. Another project involving formal modeling of train control systems, using some computer science techniques, was carried out by Schneider and co-workers [23]. Their emphasis was on the use of an extension of Dijkstra's weakest-precondition calculus to *derive* correct solutions. Other case studies in modeling hybrid systems include two analyses of steam boiler controllers — one using timed I/O automaton methods [24] and another using the automated proof checker PVS [25] — and a project using a variety of techniques to model and verify controllers for aircraft landing gear [26].

Outline

In Chapter 2 we give a complete but terse treatment of the HIOA model and the notational conventions used in this case study. In Chapters 3, 4, 5, and 6, we present a succession of different variations on the deceleration maneuver: no delay and no feedback in Chapter 3; delay and no feedback in Chapter 4; feedback and no delay in Chapter 5; and both feedback and delay in Chapter 6. We conclude in Chapter 7.

Chapter 2

Model: Hybrid I/O Automata

The hybrid I/O automaton model [1] is based on the timed I/O automaton model of [2, 3, 4, 5], but includes more explicit treatment of continuous behavior. To make this report self contained, this chapter gives a complete but terse treatment of the HIOA model with an emphasis on those aspects used in subsequent chapters. The presentation is based on [1] and [27].

The chapter is organized as follows. We begin by introducing the notion of a *trajectory*; trajectories are functions that represent the continuous evolution of state. We proceed to define hybrid I/O automata (HIOA) and their executions and traces. Next, we define a *simulation* relation between a pair of HIOAs and the operations of *composition* and of action and variable *hiding*. We conclude by presenting two notational forms for automata: standard and MMT-specifications.

2.1 Trajectories

Throughout this chapter, we fix a *time axis* T, which is a subgroup of $(\mathsf{R}, +)$, the real numbers with addition. In subsequent chapters we use $T = \mathsf{R}$ exclusively, but the model permits $T = \mathbb{Z}$ and the degenerated time axis $T = \{0\}$. An *interval* I is a convex subset of T. We denote intervals as usual: $[t_1, t_2] = \{t \in T \mid t_1 \leq t \leq t_2\}$, etc. For I an interval and $t \in T$, we define $I + t \triangleq \{t' + t \mid t' \in I\}$.

We assume a universal set $\mathcal{V}$ of *variables*. Variables in $\mathcal{V}$ are typed, where the type of a variable, such as *reals*, *integers*, etc., indicates the domain over which the variable ranges. Let $Z \subseteq \mathcal{V}$. A *valuation* of Z is a mapping that associates to each variable of Z a value in its domain. We write $\mathbf{Z}$ for the set of valuations of Z. Often, valuations will be referred to as *states*.

A *trajectory* over Z is a mapping $w : I \to \mathbf{Z}$, where I is a left-closed interval of T with left endpoint equal to 0. With $dom(w)$ we denote the domain of w and with $trajs(Z)$ the collection of all trajectories over Z. We say w is an I-trajectory if it is a trajectory with domain I. If w is a trajectory then $w.ltime$, the *limit time* of w, is the supremum of $dom(w)$. Similarly, define $w.fstate$, the *first state* of w, to be $w(0)$, and if $dom(w)$ is right-closed, define $w.lstate$, the *last state* of w, to be $w(w.ltime)$. A trajectory with domain $[0,0]$ is called a *point* trajectory. If s is a state then define $\wp(s)$ to be the point trajectory that maps 0 to s.

For w a trajectory and $t \in T^{\geq 0}$, we define $w \trianglelefteq t \triangleq w \lceil [0,t]$ and $w \triangleleft t \triangleq w \lceil [0,t)$. (Here $\lceil$ denotes the restriction of a function to a subset of its domain.) Note that $w \triangleleft 0$ is not a trajectory. By convention, $w \trianglelefteq \infty = w \triangleleft \infty \triangleq w$. Similarly we define, for w a trajectory and I a left-closed interval with minimal element l, the *restriction* $w \dagger I$ to be the function with domain $(I \cap dom(w)) - l$ given by $w \dagger I\ (t) \triangleq w(t+l)$. Note that $w \dagger I$ is a trajectory iff $l \in dom(w)$.

If w is a trajectory over Z and $Z' \subseteq Z$, then the *projection* $w \downarrow Z'$ is the trajectory over Z' with domain $dom(w)$ defined by $w \downarrow Z'\ (t)(z) \triangleq w(t)(z)$. The projection operation is extended to sets of trajectories by pointwise extension. Also, if w is a trajectory over Z and $z \in Z$, then the *projection* $w \downarrow z$ is the function from $dom(w)$ to the domain of z defined by $w \downarrow z\ (t) \triangleq w(t)(z)$.

If w is a trajectory with a right-closed domain $I = [0,u]$, w' is a trajectory with domain I', and if $w.lstate = w'.fstate$, then we define the *concatenation* $w \frown w'$ to be the trajectory with domain $I \cup (I' + u)$ given by

$$w \frown w'\ (t) \quad \triangleq \quad \begin{cases} w(t) & \text{if } t \in I, \\ w'(t-u) & \text{otherwise.} \end{cases}$$

We extend the concatenation operator to a countable sequence of trajectories: if w_i is a trajectory with domain I_i, $1 \leq i < \infty$, where all I_i are right-closed, and if $w_i.lstate = w_{i+1}.fstate$ for all i, then we define the *infinite concatenation*, written $w_1 \frown w_2 \frown w_3 \ldots$, to be the least function w such that $w(t + \sum_{j<i} w_j.ltime) = w_i(t)$ for all $t \in I_i$.

A trajectory w is *closed* if its domain is a (finite) closed interval and *full* if its domain equals $T^{\geq 0}$. For W a set of trajectories, $Closed(W)$ and $Full(W)$ denote the subsets of closed and full trajectories in W, respectively. Trajectory w is a *prefix* of trajectory w', notation $w \leq w'$, if either $w = w'$ or $w' = w \frown w''$, for some trajectory w''. With $Pref(W)$ we denote the *prefix-closure* of W: $Pref(W) \triangleq \{w \mid \exists w' \in W : w \leq w'\}$. Set W is *prefix closed* if $W = Pref(W)$. A trajectory in W is *maximal* if it is not a prefix of any other trajectory in W. We write $Max(W)$ for the subset of maximal trajectories in W.

2.2 Hybrid I/O Automata

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

- Three disjoint sets U, X and Y of variables, called *input*, *internal* and *output* variables, respectively.

Variables in $E \triangleq U \cup Y$ are called *external*, and variables in $L \triangleq X \cup Y$ are called *locally controlled*. We write $V \triangleq U \cup L$.

- Three disjoint sets Σ^{in}, Σ^{int}, Σ^{out} of *input*, *internal* and *output actions*, respectively.

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

- A nonempty set $\Theta \subseteq \mathbf{V}$ of *initial states* satisfying

  Init (start states closed under change of input variables)
  $\forall s, s' \in \mathbf{V} : s \in \Theta \wedge s\lceil L = s'\lceil L \implies s' \in \Theta$

- A set $\mathcal{D} \subseteq \mathbf{V} \times \Sigma \times \mathbf{V}$ of *discrete transitions* satisfying

  D1 (input action enabling)
  $\forall s \in \mathbf{V}, a \in \Sigma^{in}\ \exists s' \in \mathbf{V} : s \xrightarrow{a} s'$

  D2 (environment action only affect inputs)
  $\forall s, s' \in \mathbf{V} : s \xrightarrow{e} s' \implies s\lceil L = s'\lceil L$

  D3 (input variable change enabling)
  $\forall s, s', s'' \in \mathbf{V}, a \in \Sigma : s \xrightarrow{a} s' \wedge s'\lceil L = s''\lceil L \implies s \xrightarrow{a} s''$

  Here we used $s \xrightarrow{a} s'$ as shorthand for $(s, a, s') \in \mathcal{D}$.

- A set $\mathcal{W}$ of trajectories over V satisfying

  T1 (existence of point trajectories)
  $\forall s \in \mathbf{V} : \wp(s) \in \mathcal{W}$

  T2 (closure under subintervals)
  $\forall w \in \mathcal{W}, I$ left-closed, non-empty subinterval of $dom(w)$: $w \dagger I \in \mathcal{W}$

  T3 (completeness)
  $(\forall t \in T^{\geq 0} : w \dagger [0, t] \in \mathcal{W}) \Longrightarrow w \in \mathcal{W}$

Axiom **Init** says that a system has no control over the initial values of its input variables: if one valuation is allowed then any other valuation is allowed also.

Axiom **D1** is a slight generalization of the input enabling condition of the (classical) I/O automaton model: it says that in each state each input action is enabled, including the environment action e. The second axiom **D2** says that e cannot change locally controlled variables. Axiom **D3** expresses that, since input variables are not under control of the system, these variables may be changed in an arbitrary way after any discrete action. The three axioms together imply the converse of **D2**, i.e., if two states only differ in their input variables then there exists an e transition between them. Axioms **D1-3** play a crucial role in our study of parallel composition. In particular **D2** and **D3** are used to avoid cyclic constraints during the interaction of two systems.

Axioms **T1-3** state some natural conditions on the set of trajectories that we need to set up our theory: existence of point trajectories, closure under subintervals, and the fact that a full trajectory is in $\mathcal{W}$ iff all its prefixes are in $\mathcal{W}$.

Notation Let A be a HIOA as described above. If $s \in \mathbf{V}$ and $l \in \mathbf{L}$, then we write $s \xrightarrow{a} l$ iff there exists an $s' \in \mathbf{V}$ such that $s \xrightarrow{a} s'$ and $s' \lceil L = l$. In the sequel, the components of a HIOA A will be denoted by V_A, U_A, Σ_A, Θ_A, etc. Sometimes, the components of a HIOA A_i will also be denoted by V_i, U_i, Σ_i, Θ_i, etc.

2.3 Hybrid Executions

A *hybrid execution fragment* of A is a finite or infinite alternating sequence $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$, where:

1. Each w_i is a trajectory in $\mathcal{W}_A$ and each a_i is an action in Σ_A.

2. If α is a finite sequence then it ends with a trajectory.

3. If w_i is not the last trajectory in α then its domain is a right-closed interval and $w_i.\mathit{lstate} \xrightarrow{a_{i+1}}_A w_{i+1}.\mathit{fstate}$.

An execution fragment records all the discrete changes that occur in the evolution of a system, plus the "continuous" state changes that take place in between. The third item says that the discrete actions in α span between successive trajectories. We write *h-frag*(A) for the set of all hybrid execution fragments of A.

If $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$ is a hybrid execution fragment then we define the *limit time* of α, notation $\alpha.\mathit{ltime}$, to be $\sum_i w_i.\mathit{ltime}$. Further, we define the *first state* of α, $\alpha.\mathit{fstate}$, to be $w_0.\mathit{fstate}$.

We distinguish several sorts of hybrid execution fragments. A hybrid execution fragment α is defined to be

- an *execution* if the first state of α is an initial state,

- *finite* if α is a finite sequence and the domain of its final trajectory is a right-closed interval,

- *admissible* if $\alpha.\mathit{ltime} = \infty$,

- *Zeno* if α is neither finite nor admissible, and

- a *sentence* if α is a finite execution that ends with a point trajectory.

If $\alpha = w_0 a_1 w_1 \cdots a_n w_n$ is a finite hybrid execution fragment then we define the *last state* of α, notation $\alpha.lstate$, to be $w_n.lstate$. A state of A is defined to be *reachable* if it is the last state of some finite hybrid execution of A.

A finite hybrid execution fragment $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots a_n w_n$ and a hybrid execution fragment $\alpha' = w'_0 a'_1 w'_1 a'_2 w'_2 \cdots$ of A can be *concatenated* if $w_n \frown w'_0$ is defined and a trajectory of A. In this case, the *concatenation* $\alpha \frown \alpha'$ is the hybrid execution fragment defined by

$$\alpha \frown \alpha' \quad \triangleq \quad w_0 a_1 w_1 a_2 w_2 \cdots a_n (w_n \frown w'_0) a'_1 w'_1 a'_2 w'_2 \cdots$$

A variable v of a HIOA A is called *continuous* if v is not modified by any discrete steps of A and for all trajectories w of A, $w \downarrow \{v\}$ is a continuous function. Let $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$ be a hybrid execution fragment of A. Then we define $\alpha \downarrow \{v\}$ as follows:

$$\alpha \downarrow \{v\} = (w_0 \downarrow \{v\}) \frown (w_1 \downarrow \{v\}) \frown (w_2 \downarrow \{v\}) \ldots$$

The following theorem is simple to prove.

Theorem 2.3.1 *If v is a continuous variable of HIOA A and α is an execution fragment of A, then $\alpha \downarrow \{v\}$ is a continuous function.*

2.4 Hybrid Traces

Suppose $\alpha = w_0 a_1 w_1 a_2 w_2 \cdots$ is a hybrid execution fragment of A. In order to define the *hybrid trace* of α, let

$$\gamma \quad = \quad (w_0 \downarrow E_A) vis(a_1) (w_1 \downarrow E_A) vis(a_2) (w_2 \downarrow E_A) \cdots ,$$

where, for a an action, $vis(a)$ is defined equal to τ if a is an internal action or e, and equal to a otherwise. Here τ is a special symbol which, as in the theory of process algebra, plays the role of the 'generic' invisible action. An occurrence of τ in γ is called *inert* if the final state of the trajectory that precedes the τ equals the first state of the trajectory that follows it (after hiding of the internal variables). The *hybrid trace* of α, written $htrace(\alpha)$, is defined to be the sequence obtained from γ by removing all inert τ's and concatenating the surrounding trajectories.

The *hybrid traces* of A are the hybrid traces that arise from all the finite and admissible hybrid executions of A. We write *h-traces*(A) for the set of hybrid traces of A.

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

2.5 Simulation Relations

Let A and B be comparable HIOA's. A *simulation* from A to B is a relation $R \subseteq \mathbf{V}_A \times \mathbf{V}_B$ satisfying the following conditions, for all states r and s of A and B, respectively:

1. If $r \in \Theta_A$ then there exists $s \in \Theta_B$ such that $r\,R\,s$.

2. If $r \xrightarrow{a}_A r'$ and $r\,R\,s$ and both r and s are reachable states then B has a finite execution fragment α with $s = \alpha.\mathit{fstate}$, $\mathit{htrace}(\wp(r)\ a\ \wp(r')) = \mathit{htrace}(\alpha)$ and $r'\ R\ \alpha.\mathit{lstate}$.

3. If $r\,R\,s$ and w is a closed trajectory of A with $r = w.\mathit{fstate}$ and both r and s are reachable states then B has a finite execution fragment α with $s = \alpha.\mathit{fstate}$, $\mathit{htrace}(w) = \mathit{htrace}(\alpha)$ and $w.\mathit{lstate}\ R\ \alpha.\mathit{lstate}$.

Note that by Condition 3 and the existence of point trajectories (axiom **T1**), $r\,R\,s$ and r and s reachable implies that $r\lceil E_A = s\lceil E_B$.

Theorem 2.5.1 *If A and B are comparable HIOA's and there is a simulation from A to B, then $A \leq B$.*

The definition of simulation given above is weaker than the one given in [1]. We have added the restriction that r and s be reachable states in Conditions 2 and 3. Theorem 2.5.1 is true with or without this restriction.

2.6 Parallel Composition and Hiding

We say that HIOA's A_1 and A_2 are *compatible* if, for $i \neq j$,

$$X_i \cap V_j = Y_i \cap Y_j = \Sigma_i^{int} \cap \Sigma_j = \Sigma_i^{out} \cap \Sigma_j^{out} = \emptyset.$$

If A_1 and A_2 are compatible then their *composition* $A_1 \| A_2$ is defined to be the tuple $A = (U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, \mathcal{D}, \mathcal{W})$ given by

- $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$, $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$
- $\Sigma^{in} = (\Sigma_1^{in} \cup \Sigma_2^{in}) - (\Sigma_1^{out} \cup \Sigma_2^{out})$, $\Sigma^{int} = \Sigma_1^{int} \cup \Sigma_2^{int}$, $\Sigma^{out} = \Sigma_1^{out} \cup \Sigma_2^{out}$
- $\Theta = \{s \in \mathbf{V} \mid s\lceil V_1 \in \Theta_1 \wedge s\lceil V_2 \in \Theta_2\}$
- Define, for $i \in \{1, 2\}$, projection function $\pi_i : \Sigma \to \Sigma_i$ by $\pi_i(a) \triangleq a$ if $a \in \Sigma_i$ and $\pi_i(a) \triangleq e$ otherwise. Then $\mathcal{D}$ is the subset of $\mathbf{V} \times \Sigma \times \mathbf{V}$ given by

$$(s, a, s') \in \mathcal{D} \iff s\lceil V_1 \xrightarrow{\pi_1(a)}_1 s'\lceil V_1 \ \wedge\ s\lceil V_2 \xrightarrow{\pi_2(a)}_2 s'\lceil V_2$$

- $\mathcal{W}$ is the set of trajectories over V given by

$$w \in \mathcal{W} \iff w \downarrow V_1 \in W_1 \wedge w \downarrow V_2 \in W_2$$

Notation We extend the projection notation π_i ($i = 1, 2$) to states, trajectories and hybrid executions in the obvious way.

Proposition 2.6.1 *$A_1 \| A_2$ is a HIOA.*

Theorem 2.6.2 *Suppose A_1, A_2 and B are HIOA's with $A_1 \leq A_2$, and each of A_1 and A_2 is compatible with B. Then $A_1 \| B \leq A_2 \| B$.*

Two natural hiding operations can be defined on any HIOA A:
(1) If $S \subseteq \Sigma_A^{out}$, then $\mathsf{ActHide}(S, A)$ is the HIOA B that is equal to A except that $\Sigma_B^{out} = \Sigma_A^{out} - S$ and $\Sigma_B^{int} = \Sigma_A^{int} \cup S$.
(2) If $Z \subseteq Y_A$, then $\mathsf{VarHide}(Z, A)$ is the HIOA B that is the equal to A except that $Y_B = Y_A - Z$ and $X_B = X_A \cup Z$.

Theorem 2.6.3 *Suppose A and B are HIOA's with $A \leq B$, and let $S \subseteq \Sigma_A^{out}$ and $Z \subseteq Y_A$.*
Then $\mathsf{ActHide}(S, A) \leq \mathsf{ActHide}(S, B)$ and $\mathsf{VarHide}(Z, A) \leq \mathsf{VarHide}(Z, B)$.

2.7 Standard HIOA Notation

In this section we introduce the notational conventions for defining HIOAs that are standard for this case study. An example HIOA called SKEW-TIMER described in standard notation appears in Table 2.1. The automaton SKEW-TIMER models a faulty count-down timer with an inaccurate clock. The table identifies the actions, variables, discrete transitions, and trajectories of SKEW-TIMER. We explain each of these in turn.

- The actions are classified as input, output, and internal. A set of actions may be defined by giving an action name with a parameter and a range for the parameter. The actions `set-timer`(x) for $x \in \mathbb{R}^{\geq 0}$ are an example. We say "the action `set-timer`" to mean the set of related actions "`set-timer`(x) for $x \in \mathbb{R}^{\geq 0}$".

- The variables are also classified as input, output, and internal. Since there are no input variables to SKEW-TIMER, that category does not appear. Variables are specified with a name and a type; an initial value is given for internal and output variables.

- The discrete transitions are specified using precondition-effect, Pascal-like code as in [28, 29]. Each set of transitions which shares an action label (or set of related action labels) is specified as one precondition-effect block. For example, the first block describes all `set-timer` labeled transitions. Because `set-timer` is an input action there is no precondition for this block — in other words, the precondition is `true` (see Axiom **D1**). The notation $:=$ is the usual Pascal assignment notation. The notation $:\in$ is similar but denotes assignment from a set. If a variable is not mentioned in the effect clause, then it is unchanged by the transition.

- The trajectories are specified as all the trajectories w that satisfy the given set of conditions. The expression $w.rate$ denotes the projection of w onto the variable *rate*.

Informally, the behavior of SKEW-TIMER is as follows: it has a clock whose rate varies non-deterministically between 0 and 2; when it receives a `set-timer`(x) input action, it will later output `alarm` when its clock says that x time has passed; however, there may be an internal `fault` action, which causes the timer to be non-deterministically set to any value; the *togo* output variable reports the time remaining until the timer expires. The variable *deadline* is used to encode the value of *clock* that will trigger the expiration of the timer.

2.8 MMT Specifications

The HIOA model is powerful; however, a useful subclass of HIOA can be specified in a convenient notation called an MMT-specification. The name "MMT" derives from the names Merritt, Modugno, and Tuttle, the authors of [30] where they present a model which corresponds to this subclass. We prefer to view it as a subclass with a particular notation, rather than as a separate formalism. This section is based on a similar exposition in [27]. We give a formal definition of an MMT-specification, of a mapping from an MMT-specification to a HIOA, and an example MMT-specification together with its translation into standard notation.

An *MMT-specification* $M = (A, T, b_l, b_u)$ consists of the following components:

- A HIOA A with no external variables and only point trajectories.

- A task set T which is a collection of disjoint subsets of locally controlled actions of A.

- A lower bound map $b_l : T \rightarrow \mathbb{R}^{\geq 0}$.

- An upper bound map $b_u : T \rightarrow \mathbb{R}^{\geq 0}$.

Table 2.1 The SKEW-TIMER automaton.

Actions:	Input:	`set-timer`(x) for $x \in \mathbb{R}^{\geq 0}$
	Output:	`alarm`
	Internal:	`fault`
Vars:	Output:	$togo \in \mathbb{R}^{\geq 0} \cup \{\infty\}$, initially ∞
	Internal:	$clock \in \mathbb{R}^{\geq 0}$, initially 0
		$rate \in [0, 2]$, initially 1
		$deadline \in \mathbb{R}^{\geq 0} \cup \{\infty\}$, initially ∞

Discrete Transitions:

`set-timer`(x):
- Eff: $togo := x$
  - $deadline := clock + x$

`alarm`:
- Pre: $deadline = clock$
- Eff: $deadline := \infty$
  - $togo := \infty$

`fault`:
- Pre: $togo \neq 0$
- Eff: $togo :\in \mathbb{R}^{\geq 0}$
  - $deadline := clock + togo$

Trajectories:

$w.rate$ is an integrable function
for all $t \in dom(w)$
- $w(t).deadline = w(0).deadline$
- $w(t).clock = w(0).clock + \int_0^t w(s).rate\, ds$
- $w(t).clock \leq w(t).deadline$
- if $w(0).deadline = \infty$ then
  - $w(t).togo = \infty$
- else
  - $w(t).togo = w(t).deadline - w(t).clock$

The HIOA A specifies the behavior of the automaton which is *not* related to timing; its trajectories are irrelevant so we assume they are point trajectories. The remaining elements of the MMT-specification define its timing behavior. The tasks are sets of actions of A that have related timing behavior; we denote individual tasks by C_i where i ranges over an index set. The bound functions specify the timing behavior of tasks by giving a lower and upper time bound for the execution of each task. We require that for each tasks $C_i \in T$, $b_l(C_i) \leq b_u(C_i)$. An action a is *enabled* in state s when for some s', (s, a, s') is a discrete step of A. A task C_i is enabled in a state if at least one of its actions is enabled. The lower time-bound on a task specifies how long the task *must* be continuously enabled before one of its actions *can* be performed. The upper time-bound on a task specifies how long the task *can* be continuously enabled before one of its actions *must* be performed. We formalize this description by describing the equivalent hybrid I/O automaton.

Let $M = (A, T, b_l, b_u)$ be an MMT-specification where and let

$$A = (U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, \mathcal{D}, \mathcal{W})$$

and $V = U \cup X \cup Y$. By our assumption that M is an MMT-specification we know that $U = Y = \emptyset$ and $\mathcal{W}$ contains only point trajectories.

Then $A' = \mathit{hybrid}(M)$ is a hybrid I/O automaton with the following components:

- The variables of A' are the same as those of A plus the following internal variables: now of type $\mathbb{R}^{\geq 0}$; and $\mathit{first}(C_i)$ and $\mathit{last}(C_i)$ of type $\mathbb{R} \cup \{\infty\}$ for all $C_i \in T$.

- The actions of A' are the same as those of A.

- The start states A' are all the states s of A' where $s \lceil V \in \Theta$, $s.\mathit{now} = 0$, and for each $C_i \in T$ if C_i is enabled in $s \lceil V$ then $\mathit{first}(C_i) = b_l(C_i)$ and $\mathit{last}(C_i) = b_u(C_i)$; otherwise, $\mathit{first}(C_i) = 0$ and $\mathit{last}(C_i) = \infty$.

- The discrete steps of A' are all (s, a, s') where:

  1. $s'.\mathit{now} = s.\mathit{now}$
  2. $(s \lceil V, a, s' \lceil V) \in \mathcal{D}$
  3. for each $C_i \in T$
     (a) If $a \in C_i$, then $s.\mathit{first}(C_i) \leq s.\mathit{now}$.
     (b) If C_i is enabled in both $s \lceil V$ and $s' \lceil V$, and $a \notin C_i$, then $s'.\mathit{first}(C_i) = s.\mathit{first}(C_i)$ and $s'.\mathit{last}(C_i) = s.\mathit{last}(C_i)$.
     (c) If C_i is enabled in $s' \lceil V$ and either $a \in C_i$ or C_i is not enabled in $s \lceil V$, then $s'.\mathit{first}(C_i) = s'.\mathit{now} + b_l(C_i)$ and $s'.\mathit{last}(C_i) = s'.\mathit{now} + b_u(C_i)$.
     (d) If C_i is not enabled in $s' \lceil V$ then $s'.\mathit{first}(C_i) = 0$ and $s'.\mathit{last}(C_i) = \infty$.

- The trajectories of A' are exactly those trajectories w where the following hold for all $t \in dom(w)$:

  1. $w(t).now = w(0).now + t$ (now is a clock variable)
  2. $w(t) \downarrow V = w(0) \downarrow V$ (original variables remain unchanged)
  3. for all $C_i \in T$
     (a) $w(t).now \leq w(0).last(C_i)$ (time does not pass deadlines)
     (b) $w(t).first(C_i) = w(0).first(C_i)$ (deadlines remain unchanged)
     (c) $w(t).last(C_i) = w(0).last(C_i)$

One difference between the exposition here and in [27], is that we do not require that the upper bound of a task be non-zero. Such a requirement would guarantee certain properties that are required in [27] but that are beyond the scope of this exposition.

A simple example MMT-specification PING-PONG appears in Table 2.2; its corresponding HIOA *hybrid*(PING-PONG) appears in Table 2.3 in standard notation. The notation $PING = \{\texttt{ping}\} : [3, 4]$, means that task $PING$ consists of the singleton set of actions $\{\texttt{ping}\}$ and has lower and upper time bounds of 3 and 4, respectively. Informally, the behavior of PING-PONG is as follows: it alternates performing `ping` and `pong` output actions; it begins with a `ping` action after 3 to 4 time units; every `ping` action is followed by a `pong` action in 7 to 20 time units; every `pong` action is followed by a `ping` action in 3 to 4 time units.

In subsequent chapters we ignore the distinction between the MMT-specification and its corresponding hybrid I/O automaton. When possible, we will use MMT-specifications and not give the corresponding standard notation. However, we will refer in proofs to the deadline variables $last(\cdot)$ and $first(\cdot)$. These deadline variables have some useful properties:

Theorem 2.8.1 *If $M = (A, T, b_l, b_u)$ is an MMT-specification and $A' = hybrid(M)$, then in all reachable states s of A' and for all $C_i \in T$ the following hold:*

1. *$s.first(C_i) \leq s.last(C_i)$*
2. *$s.now \leq s.last(C_i)$*
3. *if C_i is enabled in $s \lceil V$ then $0 \leq last(C_i) - now \leq b_u(C_i)$*

The use of deadline variables is key to the assertional proof style. To prove invariant assertions inductively it is often helpful that the entire future behavior of the system is determined by the current state. Deadline variables encode future timing behavior in the current state. For an example see Lemma 3.6.4.

Table 2.2 The PING-PONG MMT-specification.

Actions: Output: `ping` and `pong`
Vars: Internal: $count \in \mathbb{N}$, initially 0

Discrete Transitions:

`ping`:
- Pre: $count$ is even
- Eff: $count := count + 1$

`pong`:
- Pre: $count$ is odd
- Eff: $count := count + 1$

Tasks:
$PING = \{\texttt{ping}\} : [3, 4]$
$PONG = \{\texttt{pong}\} : [7, 20]$

Notation All HIOAs that result from MMT-specifications have the *now* variable. So that we may compose these HIOAs and others that have a similar *now* variable, we adopt a convention for the *now* variable. We reserve the *now* identifier only for real-valued variables that begin at zero and progress linearly with time at slope exactly one — in other words, variables which represent the current time. These variables must be internal or output variables. When two automata are composed that both have *now* variables, we implicitly rename the variables to some other unique names but refer to both of these variables as if they were named *now*.

Table 2.3 The *hybrid*(PING-PONG) automaton.

Actions: Output: `ping` and `pong`

Vars: Internal: $count \in \mathbb{N}$, initially 0

$now \in \mathbb{R}^{\geq 0}$

$\mathit{first}(PING) \in \mathbb{R}^{\geq 0} \cup \{\infty\}$, initially 3

$\mathit{last}(PING) \in \mathbb{R}^{\geq 0} \cup \{\infty\}$, initially 4

$\mathit{first}(PONG) \in \mathbb{R}^{\geq 0} \cup \{\infty\}$, initially 0

$\mathit{last}(PONG) \in \mathbb{R}^{\geq 0} \cup \{\infty\}$, initially ∞

Discrete Transitions:

`ping`:

Pre: $count$ is even

$\mathit{first}(PING) \leq now$

Eff: $count := count + 1$

$\mathit{first}(PING) := 0$

$\mathit{last}(PING) := \infty$

$\mathit{first}(PONG) := now + 7$

$\mathit{last}(PONG) := now + 20$

`pong`:

Pre: $count$ is odd

$\mathit{first}(PONG) \leq now$

Eff: $count := count + 1$

$\mathit{first}(PING) := now + 3$

$\mathit{last}(PING) := now + 4$

$\mathit{first}(PONG) := 0$

$\mathit{last}(PONG) := \infty$

Trajectories:

$w.\mathit{first}(PING)$, $w.\mathit{last}(PING)$, $w.\mathit{first}(PONG)$, and $w.\mathit{last}(PONG)$ are all constant functions

for all $t \in dom(w)$

$w(t).now = w(0).now + t$

$w(t).now \leq w(t).\mathit{last}(PING)$

$w(t).now \leq w(t).\mathit{last}(PONG)$

Chapter 3

Deceleration Case 1: No Delay and No Feedback

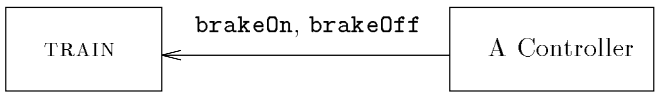
In the deceleration problem we model a computer-controlled train moving along a track. The task of the train's controller is to slow the train within a given distance. In this chapter we consider a very simple model of the train and the controller. The train has two modes, braking and not braking. The controller can instantly effect a change in the mode of the train (relaxed in Chapters 4 and 6). The controller receives no information from the train (relaxed in Chapters 5 and 6). The braking strength of the train varies nondeterministically within known bounds. We model both the train and the controller as hybrid I/O automata. Figure 3-1 illustrates the components and their communication.

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

3.1 Parameters

All the parameters of the specification are constants denoted by c with some dots above it and a subscript. Dots above the constant identify the type of the constant: position (no dots), velocity (one dot), or acceleration (two dots). The dots are a purely syntactic device used to express the *type* of the constant; they do not represent an operation of differentiation on some function. The subscript identifies the particular

Figure 3-1 Overview of Basic Deceleration Model



constant. Initial values of the train's position, velocity and acceleration are $c_{\rm s}, \dot{c}_{\rm s}, \ddot{c}_{\rm s}$. The goal of the deceleration maneuver is to slow the train to a velocity in the interval $[\dot{c}_{\rm minf}, \dot{c}_{\rm maxf}]$ at position $c_{\rm f}$. When the train is not braking its acceleration is exactly zero. When the train is braking its acceleration varies nondeterministically between $[\ddot{c}_{\rm min}, \ddot{c}_{\rm max}]$, both negative. The range is intended to model inherent uncertainty in brake performance. We impose the following constraints on the parameters:

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

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

3.2 The TRAIN Automaton

We model the train as a single HIOA called TRAIN which appears in Table 3.1. The notation used in the table is explained in Section 2.7. The train's physical state is modeled using three variables: $x, \dot{x}, \ddot{x}$. As with the constants, the dots on $\dot{x}$ and $\ddot{x}$ are a syntactic device; the fact there there is a differential relationship between the evolution of these variables is a consequence of the definition of the trajectory set for TRAIN. The train accepts commands to turn the brake on or off through discrete actions `brakeOn` and `brakeOff`. It stores the state of the brake in variable b. While braking the train applies an acceleration that is nondeterministic at every point but is constrained to be an integrable function with range in the interval $[\ddot{c}_{\mathrm{min}}, \ddot{c}_{\mathrm{max}}]$. While not braking the train has exactly zero acceleration. The variable *now* represents the current time; when using assertions to reason about the timing behavior of systems, it is convenient to have an explicit state variable which records the current time.

Table 3.1 The TRAIN automaton.

Actions: Input: `brakeOn` and `brakeOff`

Vars: Output:
- $x \in \mathbb{R}$, initially $x = c_{\mathrm{s}}$
- $\dot{x} \in \mathbb{R}$, initially $\dot{x} = \dot{c}_{\mathrm{s}}$
- $\ddot{x} \in \mathbb{R}$, initially $\ddot{x} = \ddot{c}_{\mathrm{s}}$
- b, a boolean, initially `false`
- $now \in \mathbb{R}^{\geq 0}$, initially 0

Discrete Transitions:

`brakeOn`:
  Eff: $b :=$ `true`
  $\ddot{x} :\in [\ddot{c}_{\mathrm{min}}, \ddot{c}_{\mathrm{max}}]$

`brakeOff`:
  Eff: $b :=$ `false`
  $\ddot{x} := 0$

Trajectories:

if $w(0).b =$ `true` then
  $w.\ddot{x}$ is an integrable function with range $[\ddot{c}_{\mathrm{min}}, \ddot{c}_{\mathrm{max}}]$
else
  $w.\ddot{x} = 0$
for all $t \in I$ the following hold:
  $w(t).b = w(0).b$
  $w(t).now = w(0).now + t$
  $w(t).\dot{x} = w(0).\dot{x} + \int_0^t w(s).\ddot{x}\, ds$
  $w(t).x = w(0).x + \int_0^t w(s).\dot{x}\, ds$

3.3 Properties of TRAIN

The following two lemmas and three corollaries all relate the initial state and final states of a trajectory. They establish standard facts of mechanics which we prove here for completeness. In a treatment of a system with more complex dynamics we expect that the lemmas of this section could be replaced with similar results based on whatever methods from continuous mathematics were appropriate for the specific application. We do not claim that the dynamics of TRAIN are complex or that the mathematics used in the proofs in this section is sophisticated.

In the next two lemmas we characterize the train's behavior when not braking and when braking, respectively. Below and throughout this work, if s and s' are states and x is a variable, we often write x for $s.x$ and x' for $s'.x$ when s and s' are understood.

Lemma 3.3.1 *Let w be a closed trajectory of* TRAIN *whose initial and final states are s and s', respectively, and let $\Delta = now' - now$. If $b =$* `false` *then the following hold:*

1. $\ddot{x}' = \ddot{x} = 0$
2. $\dot{x}' = \dot{x}$
3. $x' = x + \dot{x}\Delta$

Proof: By the definitions of $\dot{x}$ and x in TRAIN and integration. ■

Lemma 3.3.2 *Let w be a closed trajectory of* TRAIN *whose initial and final states are s and s', respectively, and let $\Delta = now' - now$. If $b =$* `true` *then the following hold:*

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

Proof: We prove only the right hand side of the two inequalities; the other side is symmetric. Let z be a trajectory of TRAIN with the domain I the same as w; and let $z(t).\ddot{x} = \ddot{c}_{\max}$ for all $t \in I$ and $z(0).\dot{x} = w(0).\dot{x}$ and $z(0).x = w(0).\dot{x}$. Notice that $w(t).\ddot{x} \leq z(t).\ddot{x}$ for all $t \in I$. Because definite integrals preserve inequalities, we know that for all $t \in I$, $w(t).\dot{x} \leq z(t).\dot{x}$ and $w(t).x \leq z(t).x$. Furthermore, by integration, we know that $z(t).\dot{x} = w(0).x + \ddot{c}_{\max}\Delta$. This establishes the first inequality. Also by integration, we know that $z(t).x = w(0).x + w(0).\dot{x}\Delta + \frac{1}{2}\ddot{c}_{\max}\Delta^2$. This establishes the second inequality. ■

The following corollaries further describe the train's behavior during braking. The first bounds change in time by change in velocity. The second bounds change in position by change in the square of velocity.

Corollary 3.3.3 *Let w be a closed trajectory of* TRAIN *whose initial and final states are s and s', respectively and let $\Delta = now' - now$. If $b =$* `true` *then the following holds:*

$$\frac{\dot{x}' - \dot{x}}{\ddot{c}_{\min}} \leq \Delta \leq \frac{\dot{x}' - \dot{x}}{\ddot{c}_{\max}}$$

Proof: We use Lemma 3.3.2. The steps for only one side are shown:

$$\begin{array}{rcll}
\dot{x}' & \leq & \dot{x} + \ddot{c}_{\max}\Delta & \text{by Lemma 3.3.2} \\
\dot{x}' - \dot{x} & \leq & \ddot{c}_{\max}\Delta & \text{subtract} \\
\ddot{c}_{\max} & \leq & 0 & \text{assumption} \\
\frac{\dot{x}'-\dot{x}}{\ddot{c}_{\max}} & \geq & \Delta & \text{division}
\end{array}$$

■

Corollary 3.3.4 *Let w be a closed trajectory of* TRAIN *whose initial and final states are s and s', respectively and let $\Delta = now' - now$. If $b =$ `true` and $0 \leq \dot{x}'$ then the following holds:*

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

Proof: Again, we show only the right hand side of the inequality. Let $\Delta = now' - now$. Let z be a trajectory as in the proof of Lemma 3.3.2 and let f denote the final state of z. To make the following algebra easier to read, we let $\dot{u}' = f.\dot{x}$ and $u' = f.x$. As usual, $x = s.x, \dot{x} = s.\dot{x}, x' = s'.x$, and $\dot{x}' = s'.\dot{x}$.

$$\begin{array}{rcll}
\dot{u}' & = & \dot{x} + \ddot{c}_{\max}\Delta & \text{integration} \\
u' & = & x + \dot{x}\Delta + \frac{1}{2}\ddot{c}_{\max}\Delta^2 & \text{integration} \\
\Delta & = & \frac{\dot{u}'-\dot{x}}{\ddot{c}_{\max}} & \text{solve for } \Delta \\
u' & = & x + \frac{\dot{x}\dot{u}'-\dot{x}^2}{\ddot{c}_{\max}} + \frac{1}{2}\ddot{c}_{\max}\frac{(\dot{u}')^2-2\dot{x}\dot{u}'+\dot{x}^2}{\ddot{c}_{\max}^2} & \text{substitution} \\
u' & = & x + \frac{1}{2\ddot{c}_{\max}}(2\dot{x}\dot{u}' - 2\dot{x}^2 + (\dot{u}')^2 - 2\dot{x}\dot{u}' + \dot{x}^2) & \text{distribute} \\
u' & = & x + \frac{(\dot{u}')^2-\dot{x}^2}{2\ddot{c}_{\max}} & \text{cancel} \\
x' & \leq & u' & \text{as in Lemma 3.3.2} \\
x' & \leq & x + \frac{(\dot{u}')^2-\dot{x}^2}{2\ddot{c}_{\max}} & \text{transitivity} \\
0 & \leq & \dot{x}' & \text{antecedent} \\
\dot{x}' & \leq & \dot{u}' & \text{as in Lemma 3.3.2} \\
\dot{u}' & < & \dot{x} & (\ddot{c}_{\max} < 0) \\
x' & \leq & x + \frac{(\dot{x}')^2-\dot{x}^2}{2\ddot{c}_{\max}} & \text{substitution} \\
x' - x & \leq & \frac{(\dot{x}')^2-\dot{x}^2}{2\ddot{c}_{\max}} & \text{subtraction}
\end{array}$$

■

3.4 Definition of Controller Correctness

We define a *brake-controller* to be a hybrid I/O automaton with no external variables, no input actions, and output actions `brakeOn`, and `brakeOff`. A *correct* brake-controller is one that when composed with TRAIN, yields a HIOA whose hybrid traces satisfy the following formal axioms:

Timeliness There exists a constant $t \in \mathbb{R}^{\geq 0}$ such that for all hybrid traces if there exists a state of the trace in which $now = t$, then there is a state of the trace in which $x = c_\mathrm{f}$.

Safety In all states of all hybrid traces the following holds:
$x = c_\mathrm{f} \Longrightarrow \dot{c}_{\mathrm{minf}} \leq \dot{x} \leq \dot{c}_{\mathrm{maxf}}$.

These can be stated informally as: (Timeliness) there is a length of time after which we can be sure that the train has reached c_f; and (Safety) when it gets there, it has achieved an appropriate speed. The formal definitions of hybrid traces and related concepts appear in Chapter 2. Note that in (3.4) the state where $x = c_\mathrm{f}$ can occur during time passage, i.e. within a trajectory. For convenience we call the first property the "timeliness" property and the second property the "safety" property.

A controller which stops time before the system reaches c_f is a correct controller according to the above definition. In general, one would like to avoid such vacuous correctness results. This issue is beyond the scope of our investigation, but it is treated in some depth in [1, 4, 5]. None of the of the example controllers presented in this case study stop time.

The following theorem says that the timeliness and safety properties are preserved by the implementation relation (see Section 2.4); in other words, an implementation of a correct brake-controller is itself a correct brake-controller. This theorem is not used in this chapter but rather in Chapter 4.

Theorem 3.4.1 *Let B be a correct brake-controller and let $A \leq B$. Then A is also a correct brake-controller.*

Proof: By Theorem 2.6.2, $A||\textsc{train} \leq B||\textsc{train}$. Timeliness: Let t be the constant which satisfies the timeliness property for B. We show that it also satisfies the timeliness property for A. Let α be a trace of $A||\textsc{train}$; then α is also a trace of $B||\textsc{train}$ and the property holds on α by the correctness of B. Safety: Similarly. ∎

3.5 Example Controller: ONE-SHOT

In this section we give an example of a correct *brake-controller* called ONE-SHOT. There is a broad spectrum of correct controllers from which to choose an example — from fully deterministic controllers to highly non-deterministic controllers. A fully deterministic controller would have exactly one infinite execution (ignoring e transitions). We have chosen to present a controller that is highly non-deterministic: ONE-SHOT exhibits all the possible timings of exactly one `brakeOn` action followed by exactly one `brakeOff` action which a correct controller might exhibit. In other words, ONE-SHOT exhibits all the correct braking strategies which involve exactly one application of the brake. We can imagine controllers with more non-determinism which exhibit not only behaviors with single brake applications but also behaviors with multiple brake applications. We chose ONE-SHOT as an example for three reasons. First, it is easily expressed using an MMT-specification. Second, it has enough interesting behavior that the proofs of this section illustrate non-trivial proof techniques. Third and last, in Chapter 4 we use a simulation proof to show that the composition of a similar controller and a delay buffer is an implementation of this controller. The correctness of the delayed controller then follows from the correctness of ONE-SHOT.

First we define some convenient constants:

$$\begin{aligned} A &= \frac{1}{\dot{c}_{\mathrm{s}}}\left(c_{\mathrm{f}} - c_{\mathrm{s}} - \frac{\dot{c}_{\mathrm{maxf}}^2 - \dot{c}_{\mathrm{s}}^2}{2\ddot{c}_{\mathrm{max}}}\right) \\ B &= \frac{\dot{c}_{\mathrm{maxf}} - \dot{c}_{\mathrm{s}}}{\ddot{c}_{\mathrm{max}}} \\ C &= \frac{\dot{c}_{\mathrm{minf}} - \dot{c}_{\mathrm{s}}}{\ddot{c}_{\mathrm{min}}} \end{aligned}$$

The first, A, is the longest amount of time a correct controller can wait before invoking the brake. The others, B and C, are lower and upper bounds, respectively, on the amount of time a correct controller should apply the brake if it only brakes once. These constants are used as the time bounds on the tasks of ONE-SHOT.

Table 3.2 The ONE-SHOT automaton (MMT-specification)

Actions: Output: `brakeOn` and `brakeOff`
Vars: Internal: $\mathit{phase} \in \{$`idle`, `braking`, `done`$\}$, initially `idle`

Discrete Transitions:

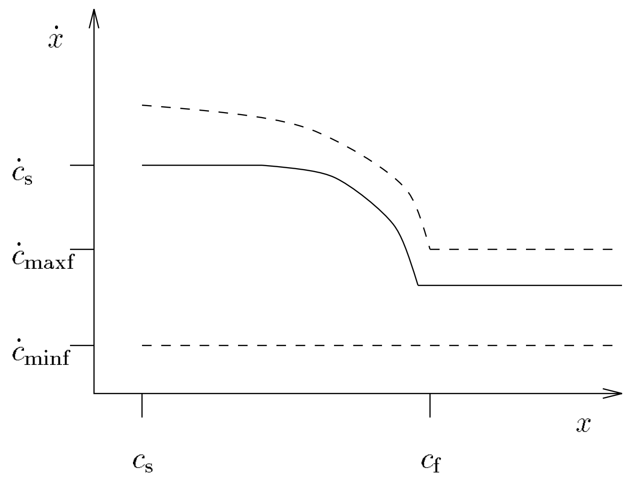
`brakeOn`:
  Pre: $\mathit{phase} =$ `idle`
  Eff: $\mathit{phase} :=$ `braking`

`brakeOff`:
  Pre: $\mathit{phase} =$ `braking`
  Eff: $\mathit{phase} :=$ `done`

Tasks: $\mathit{ON} = \{$`brakeOn`$\} : [0, A]$
  $\mathit{OFF} = \{$`brakeOff`$\} : [B, C]$

The formal description of ONE-SHOT appears in Table 3.2. The notation used in the table, called MMT-specification, is explained in Section 2.8. The controller is called "one-shot" because it applies the brake only once. The automaton's executions consist of three phases `idle`, `braking`, and `done`. It waits between zero and A time

Figure 3-2 Example Execution of ONE-SHOT-SYS



units (`idle` phase), then it applies the brake for at least B and at most C time units (`braking` phase), and then removes the brake (`done`phase). The *ON* task governs the transitions from `idle` to `braking` and the *OFF* task governs the transitions from `braking` to `done`.

3.6 Correctness of ONE-SHOT

In this section we prove the correctness of the ONE-SHOT controller. Recall that the composition of TRAIN and ONE-SHOT is called ONE-SHOT-SYS. We will present lemmas and corollaries that establish the timeliness and safety properties for the hybrid executions of ONE-SHOT-SYS. Before giving the proof, we provide some motivation and an overview.

Figure 3-2 depicts a possible execution of ONE-SHOT-SYS. The vertical axis is velocity and the horizontal axis is position. Since the vehicle is always moving forward, the graph can be read as if time progresses from left to right. The solid line represents the actual behavior of the train in this example execution. The initial flat segment corresponds to the `idle` phase; the downward curve, the `braking` phase; and the final flat segment, the `done` phase. The shape of the downward curve in this execution is meant to reflect a constant deceleration, but this is the exception rather than the rule. The train's deceleration can vary nondeterministically during braking as long as it remains integrable. As achieved deceleration varies between $\ddot{c}_{\mathrm{min}}$ and $\ddot{c}_{\mathrm{max}}$ the curve becomes more or less steep, respectively.

The dotted lines represent upper and lower bounds that we will prove. The lower bound will yield the timeliness property. The meaning of the lower bound is obvious: we will show that the controller never allows the speed to fall below the minimum final velocity. The upper bound (combined with the lower bound) will yield the safety property. The meaning of the upper bound is less obvious: from each point on the upper bound, if the controller initiated braking and the train achieved only the weakest possible braking ($\ddot{c}_{\max}$) the train would slow to exactly $\dot{c}_{\mathrm{maxf}}$ at the final position. Points below this curve are safe because immediately braking for sufficiently long will slow the train to strictly less than $\dot{c}_{\mathrm{maxf}}$ before the final position. Points above this line are unsafe because even with immediate braking, the train may achieve only the weakest possible braking — in that case the train will remain strictly above the required $\dot{c}_{\mathrm{maxf}}$ velocity at the final position.

Now we proceed to the details of the proof. In the following two sections, we prove a variety of properties, almost all of which are invariant assertions. We make extensive use of the deadline variables such as *last*(*ON*) which are implicit in the MMT-specification of ONE-SHOT. These variables allow assertions to encode claims about timing behavior. The proofs offer an argument for the clarity and simplicity of the assertional proof style. Almost all of the proofs involve only very local reasoning about steps of the system. The only proof which is not based on an assertion style, that of Lemma 3.6.8, relies on Theorem 2.3.1.

Section 3.6.1 establishes the timeliness property; Section 3.6.2 establishes the safety property. Together they yield the correctness of the controller which is summarized in Theorem 3.6.14.

3.6.1 Timeliness

In this section we prove the timeliness property, namely that there is a bound t on the time it takes to reach c_{f}. Our method is to prove that at all times there is a positive lower bound on velocity, specifically $\dot{c}_{\mathrm{minf}}$. We do this by characterizing velocity for each of the three phases: `idle` in Lemma 3.6.3, `braking` in Lemma 3.6.4, and `done` in Lemma 3.6.5. Some of the results are more general than necessary for the timeliness property because they will be used in the next section for proving the safety property.

The following two technical lemmas will be used to eliminate certain cases in later inductive arguments. The first says that there is only one `idle` phase and it occurs at the beginning of the execution. The second says that there are some dependencies among the values of the variables b, $\ddot{x}$, and *phase*.

Lemma 3.6.1 *In all reachable states of* ONE-SHOT, *if* (*phase* = `idle`) *then the following hold:*

1. *first*(*ON*) $= 0$
2. *last*(*ON*) $= A$

Proof: Trivial induction. ■

Lemma 3.6.2 *In all reachable states of* ONE-SHOT-SYS *the following hold:*

1. $b \Longrightarrow \ddot{x} \in [\ddot{c}_{\mathrm{min}}, \ddot{c}_{\mathrm{max}}]$

2. $\neg b \Longrightarrow \ddot{x} = 0$

3. $b \Longleftrightarrow (phase = \texttt{braking})$

Proof: Trivial induction. ■

The following lemma characterizes the velocity and position of the train during the controller's idle phase.

Lemma 3.6.3 *In all reachable states of* ONE-SHOT-SYS, *if* $phase = \texttt{idle}$ *the following hold:*

1. $\dot{x} = \dot{c}_{\mathrm{s}}$

2. $x = c_{\mathrm{s}} + (now)\dot{c}_{\mathrm{s}}$

Proof: By induction. The interesting case is trajectories where we note that $\ddot{x} = 0$ and Lemma 3.3.1 applies. Some trivial algebra yields the desired result. ■

The following lemma characterizes the velocity of the train during braking. It is interesting because it involves assertion-style reasoning about the controller's deadline variables. While the controller is in the `braking` phase, $last(OFF) - now$ is the greatest amount of time the train will continue braking. This time must be bounded in order to avoid slowing down below the minimum final speed, $\dot{c}_{\mathrm{minf}}$. A similar result holds for $first(OFF)$ and the upper bound on velocity.

Lemma 3.6.4 *In all reachable states of* ONE-SHOT-SYS, *if* $phase = \texttt{braking}$ *the following hold:*

1. $last(OFF) - now \leq \frac{\dot{c}_{\mathrm{minf}} - \dot{x}}{\ddot{c}_{\mathrm{min}}}$

2. $first(OFF) - now \geq \frac{\dot{c}_{\mathrm{maxf}} - \dot{x}}{\ddot{c}_{\mathrm{max}}}$

Proof: By induction. The two interesting cases are the ON task that sets $phase = \texttt{braking}$ and trajectories while $phase = \texttt{braking}$. For the ON task the pre-state has $phase = \texttt{idle}$ and Lemma 3.6.3 and the definitions of B and C yield the desired results as follows (only (2) is shown):

$$\begin{array}{rcll} B & = & \frac{\dot{c}_{\mathrm{maxf}} - \dot{c}_{\mathrm{s}}}{\ddot{c}_{\mathrm{max}}} & \text{by definition} \\ \dot{x} & = & \dot{c}_{\mathrm{s}} = \dot{x}' & \text{by Lemma 3.6.3} \\ first(OFF)' & = & now' + B & \text{ONE-SHOT definition} \\ first(OFF)' - now' & = & \frac{\dot{c}_{\mathrm{maxf}} - \dot{x}}{\ddot{c}_{\mathrm{max}}} & \text{substitute \& subtract} \end{array}$$

For trajectories, we use Lemma 3.6.2 and the equation from Corollary 3.3.3. Subtraction and expansion of $\Delta = now' - now$ yields the desired results as follows (only

(2) is shown):

$$\begin{array}{rcll}
now' - now & \leq & \frac{\dot{x}' - \dot{x}}{\ddot{c}_{\max}} & \text{by Corollary 3.3.3.} \\
first(OFF) - now & \geq & \frac{\dot{c}_{\mathrm{maxf}} - \dot{x}}{\ddot{c}_{\max}} & \text{inductive hypothesis} \\
first(OFF) - now' & \geq & \frac{\dot{c}_{\mathrm{maxf}} - \dot{x}'}{\ddot{c}_{\max}} & \text{substitute and cancel}
\end{array}$$

■

The following corollary uses basic properties of deadline variables and the preceding lemma to prove that as we exit the `braking` phase and thereafter, we are in the target velocity range.

Corollary 3.6.5 *In all reachable states of* ONE-SHOT-SYS, *if* $phase =$ `done` *the following holds:*

$$\dot{c}_{\mathrm{maxf}} \geq \dot{x} \geq \dot{c}_{\mathrm{minf}}$$

Proof: By induction. The interesting cases are the *OFF* action and trajectories in the `done` phase. For the *OFF* action we know that in the pre-state $phase =$ `braking` so Lemma 3.6.4 applies. Furthermore $first(OFF) \leq now \leq last(OFF)$ by a property of MMT automata. From this we can conclude that $\dot{c}_{\mathrm{maxf}} \geq \dot{x} \geq \dot{c}_{\mathrm{minf}}$ (details for one side shown below). For trajectories, we know that $\ddot{x} = 0$ so $\dot{x} = \dot{x}'$, by Lemma 3.6.2 and Lemma 3.3.1.

$$\begin{array}{rcll}
first(OFF) - now & \geq & \frac{\dot{c}_{\mathrm{maxf}} - \dot{x}}{\ddot{c}_{\max}} & \text{from Lemma 3.6.4} \\
first(OFF) & \leq & now & \text{from Theorem 2.8.1} \\
first(OFF) - now & \leq & 0 & \text{subtraction} \\
0 & \geq & \frac{\dot{c}_{\mathrm{maxf}} - \dot{x}}{\ddot{c}_{\max}} & \text{transitivity} \\
0 & > & \ddot{c}_{\max} & \text{assumption} \\
0 & \leq & \dot{c}_{\mathrm{maxf}} - \dot{x} & \text{multiply} \\
\dot{x} & \leq & \dot{c}_{\mathrm{maxf}} & \text{subtract}
\end{array}$$

■

The following lemma and associated corollary combines the above phase-by-phase results to yield the global result and the time bound.

Lemma 3.6.6 *In all reachable states of* ONE-SHOT-SYS *the following holds:*

$$\dot{x} \geq \dot{c}_{\mathrm{minf}}$$

Proof: We consider cases of *phase*. When $phase =$ `idle` Lemma 3.6.3 gives $\dot{x} = \dot{c}_{\mathrm{s}}$ and by assumption $\dot{c}_{\mathrm{s}} > \dot{c}_{\mathrm{maxf}} \geq \dot{c}_{\mathrm{minf}}$. When $phase =$ `braking`, Lemma 2.8.1 gives $now \leq last(OFF)$ and Lemma 3.6.4 gives the desired result. Finally when $phase =$ `done`, Corollary 3.6.5 applies. ■

Corollary 3.6.7 *In all reachable states of* ONE-SHOT-SYS *the following holds:*

$$x \geq c_{\mathrm{s}} + \dot{c}_{\mathrm{minf}}(now)$$

Proof: Lemma 3.6.6 establishes that in all reachable states (including those in trajectories) $\dot{x} \geq \dot{c}_{\mathrm{minf}}$. At all times $x - c_{\mathrm{s}}$ is the integral of $\dot{x}$. It is a property of definite integrals that lower bounds are preserved. Therefore $x - c_{\mathrm{s}} \geq \int_0^{now} \dot{c}_{\mathrm{minf}}\, dt = \dot{c}_{\mathrm{minf}}(now)$. ■

The following lemma establishes the timeliness property.

Lemma 3.6.8 *Let α be a trace of* ONE-SHOT-SYS*. If there exists a state s of α in which $s.now = \frac{c_{\mathrm{f}} - c_{\mathrm{s}}}{\dot{c}_{\mathrm{minf}}}$, then there is a state s' of α in which $s'.x = s'.c_{\mathrm{f}}$.*

Proof: By Corollary 3.6.7 we know that in state s, $s.x \geq c_{\mathrm{f}}$. We observe that no discrete action modifies x and that for all trajectories w of the system, $w.x$ is a continuous function. Therefore x is a continuous variable of ONE-SHOT-SYS (see end of Section 2.3). Let α' be an execution of ONE-SHOT-SYS whose trace is α. Let $f = \alpha' \downarrow \{x\}$. By Theorem 2.3.1, f is a continuous function. We know $f(s.now) \geq c_{\mathrm{f}}$ and that $f(0) = c_{\mathrm{s}} < c_{\mathrm{f}}$. By the intermediate value theorem, it follows that for some t where $0 \leq t \leq s.now$, $f(t) = c_{\mathrm{f}}$. We conclude that a state where $x = c_{\mathrm{f}}$ is achieved in α' and hence in α. ■

3.6.2 Safety

In this section we prove the safety property, namely that the following formula is an invariant of the system:

$$(x = c_{\mathrm{f}} \Longrightarrow \dot{c}_{\mathrm{minf}} \leq \dot{x} \leq \dot{c}_{\mathrm{maxf}})$$

We have already shown that at all times $\dot{c}_{\mathrm{minf}} \leq \dot{x}$, therefore we need only establish the other half of the inequality. To prove this invariant we prove a stronger invariant:

$$x \leq c_{\mathrm{f}} \Longrightarrow c_{\mathrm{f}} - x \geq \frac{\dot{c}_{\mathrm{maxf}}^2 - \dot{x}^2}{2\ddot{c}_{\mathrm{max}}}$$

Intuitively, this invariant says that before reaching the final position there must be enough distance left to brake, even at the weakest braking. It has as a special case the safety property (note that $\ddot{c}_{\mathrm{max}}$ is negative). This is a common technique for proving an invariant: not all invariants can be proven inductively but there is usually a strengthening of the invariant which can. Once again, we prove the invariant for each *phase*(3.6.9, 3.6.10, 3.6.11) and combine the results (3.6.12). The safety property is proved in corollary 3.6.13.

Lemma 3.6.9 *In all reachable states of* ONE-SHOT-SYS*, if $phase =$ `idle` then $c_{\mathrm{f}} - x \geq \frac{\dot{c}_{\mathrm{maxf}}^2 - \dot{x}^2}{2\ddot{c}_{\mathrm{max}}}$.*

Proof: By Lemmas 2.8.1 and 3.6.1 we know $now \leq A$. Using the equations for $\dot{x}$ and x from Lemma 3.6.3 we substitute and simplify, yielding the desired result (see definition of A).

now	$\leq$	$\frac{1}{\dot{c}_s}\left(c_f - c_s - \frac{\dot{c}_{maxf}^2 - \dot{c}_s^2}{2\ddot{c}_{min}}\right)$	from $now \leq A$
$c_s + (now)\dot{c}_s$	$\leq$	$c_f - \frac{\dot{c}_{maxf}^2 - \dot{c}_s^2}{2\ddot{c}_{min}}$	multiply by $\dot{c}_s$ and add c_s
x	$=$	$c_s + (now)\dot{c}_s$	from Lemma 3.6.3
x	$\leq$	$c_f - \frac{\dot{c}_{maxf}^2 - \dot{c}_s^2}{2\ddot{c}_{min}}$	$=$ and $\leq$ transitive
$c_f - x$	$\geq$	$\frac{\dot{c}_{maxf}^2 - \dot{c}_s^2}{2\ddot{c}_{min}}$	subtract c_f and reverse sign

■

Lemma 3.6.10 *In all reachable states of* ONE-SHOT-SYS, *if* $phase =$ `braking` *then* $c_f - x \geq \frac{\dot{c}_{maxf}^2 - \dot{x}^2}{2\ddot{c}_{max}}$

Proof: By induction. The interesting cases are the *ON* task and trajectories while $phase =$ `braking`. In the *ON* task case Lemma 3.6.9 applies to the pre-state; since none of the state variables mentioned in the formula change during the *ON* task the formula still holds. In the trajectory case, we substitute from Lemma 3.3.4 into the inductive hypothesis and simplify.

$c_f - x$	$\geq$	$\frac{\dot{c}_{maxf}^2 - \dot{x}^2}{2\ddot{c}_{max}}$	inductive hypothesis
$x' - x$	$\leq$	$\frac{\dot{x}'^2 - \dot{x}^2}{2\ddot{c}_{max}}$	from Lemma 3.3.4
$c_f - x - x' + x$	$\geq$	$\frac{\dot{c}_{maxf}^2 - \dot{x}^2 - \dot{x}'^2 + \dot{x}^2}{2\ddot{c}_{max}}$	subtract
$c_f - x'$	$\geq$	$\frac{\dot{c}_{maxf}^2 - \dot{x}'^2}{2\ddot{c}_{max}}$	cancel

■

Lemma 3.6.11 *In all reachable states of* ONE-SHOT-SYS, *if* $x \leq c_f$ *and* $phase =$ `done` *then*

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

Proof: Directly using Lemma 3.6.5. The left hand side is bounded below by zero because $x \leq c_f$. The right hand side is bounded above by zero because $\dot{x} \leq \dot{c}_{maxf}$. ■

Corollary 3.6.12 *In all reachable states of* ONE-SHOT-SYS, *if* $x \leq c_f$ *then* $c_f - x \geq \frac{\dot{c}_{maxf}^2 - \dot{x}^2}{2\ddot{c}_{max}}$

Proof: Directly using Corollaries 3.6.9, 3.6.10, and 3.6.11. ■

Corollary 3.6.13 *In all reachable states of* ONE-SHOT-SYS*:*

$$c_{\mathrm{f}} = x \Longrightarrow \dot{c}_{\mathrm{maxf}} \geq \dot{x} \geq \dot{c}_{\mathrm{minf}}$$

Proof: Directly using 3.6.12 and 3.6.6. ■

We conclude this chapter with a theorem which summarizes the correctness result for the ONE-SHOT controller.

Theorem 3.6.14 *The following are true of* ONE-SHOT-SYS*:*

Timeliness *For all hybrid traces α of* ONE-SHOT-SYS*, if in some state of α $now = \frac{x-c_{\mathrm{s}}}{\dot{c}_{\mathrm{minf}}}$, then for some state in α $x = c_{\mathrm{f}}$.*

Safety *In all states of all hybrid traces of* ONE-SHOT-SYS*, the following holds:* $x = c_{\mathrm{f}} \Longrightarrow \dot{c}_{\mathrm{minf}} \leq \dot{x} \leq \dot{c}_{\mathrm{maxf}}$.

In other words, ONE-SHOT *is a correct brake-controller.*

Proof: We establish the timeliness property for hybrid executions of ONE-SHOT-SYS in Lemma 3.6.8; we establish the safety property for hybrid executions of ONE-SHOT-SYS in Corollary 3.6.13. The properties extend to the hybrid traces of ONE-SHOT-SYS because each hybrid trace is the projection of some hybrid execution. Controller correctness is defined in Section 3.4. ■

Chapter 4

Deceleration Case 2: Delay and No Feedback

In this chapter we extend the model of the train by nondeterministically delaying the braking commands. Rather than modify the train automaton itself, we introduce a new automaton called BUFFER that will serve as a buffer between the train and a controller. Figure 4-1 illustrates the components and their communication.

In the following sections we present BUFFER, modify the correctness criteria to account for the BUFFER, give an example controller called DEL-ONE-SHOT, and prove that it is correct. The proof uses a simulation mapping to show that the composition of DEL-ONE-SHOT and BUFFER implements ONE-SHOT; the correctness of DEL-ONE-SHOT then follows (in part) from Theorem 3.4.1.

4.1 The BUFFER Automaton

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

The BUFFER automaton appears in Table 4.1. It is largely self explanatory. The variable *request* stores a command while it is being buffered. The history variable *violation* becomes true when a new command from the controller arrives before the previous one has exited the buffer, that is when the buffer overflows. We use *violation*

Figure 4-1 Overview of Delay Deceleration Model

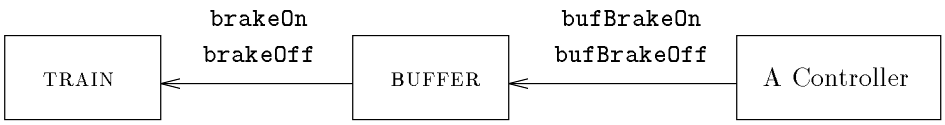


Table 4.1 The BUFFER automaton.

Actions: Inputs: `bufBrakeOn` and `bufBrakeOff`
Outputs: `brakeOn` and `brakeOff`
Vars: Internal: $request \in \{\texttt{on}, \texttt{off}, \texttt{none}\}$, initially `none`
violation, boolean, initially `false`
Discrete Transitions:
`bufBrakeOn`:
Eff: Cases of *request*,
`on` : no effect
`off` : $violation := \texttt{true}$
`none` : $request := \texttt{on}$
`bufBrakeOff`:
Eff: Cases of *request*,
`on` : $violation := \texttt{true}$
`off` : no effect
`none`: $request := \texttt{off}$
`brakeOn`:
Pre: $request = \texttt{on}$
Eff: $request := \texttt{none}$
`brakeOff`:
Pre: $request = \texttt{off}$
Eff: $request := \texttt{none}$

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

to flag this error condition.

4.2 Definition of Controller Correctness, Revisited

We modify the definition of a correct controller to account for the buffer. Let ϕ be an operator on automata which hides the actions `bufBrakeOn` and `bufBrakeOff` (see Section 2.6). A correct *buffered-brake-controller* is a HIOA C with no external variables and with output actions `bufBrakeOn` and `bufBrakeOff` such that the composition $\phi(C||\text{BUFFER})||\text{TRAIN}$ is a correct *brake-controller* as defined in Section 3.4. The use of the hiding operator ϕ in the correctness definition is a technical convenience.

4.3 Parameters, Revisited

Not only do we need to place restrictions on the value of the new parameters (δ^-, δ^+), but we also need to revise the constraints among the original parameters in light of these new ones. Intuitively, the controller is subject to more uncertainty and therefore needs less stringent requirements. The further constraints can be viewed as forcing the target velocity range, $[\dot{c}_{\mathrm{minf}}, \dot{c}_{\mathrm{maxf}}]$ to be wider and hence the controller's task easier. These are the additional constraints:

1. $0 \leq \delta^- \leq \delta^+$
2. $\dot{c}_{\mathrm{s}} \geq \dot{c}_{\mathrm{maxf}} + \ddot{c}_{\mathrm{max}}\delta^+$
3. $\dot{c}_{\mathrm{maxf}} \geq \dot{c}_{\mathrm{minf}} + \ddot{c}_{\mathrm{min}}\delta^+$
4. $\frac{\dot{c}_{\mathrm{maxf}} - \dot{c}_{\mathrm{s}}}{\ddot{c}_{\mathrm{max}}} + \delta^+ - \delta^- \leq \frac{\dot{c}_{\mathrm{minf}} - \dot{c}_{\mathrm{s}}}{\ddot{c}_{\mathrm{min}}} - \delta^+ + \delta^-$

The first constraint ensures that the delay interval is well-defined. The next two are necessary to ensure that the buffer does not overflow. The last constraint replaces constraint number six in Section 3.1; the new version accounts not only for the nondeterminism of the braking strength but also for the buffer. The other five original constraints remain as well but are not shown here. Note that these constraints in this chapter are more restrictive than the constraints from Chapter 3.

4.4 Example Controller: DEL-ONE-SHOT

Here we give an example of a valid buffered-brake-controller called DEL-ONE-SHOT. This automaton is identical to ONE-SHOT of Section 3.4 except in the names of its actions and the duration of its phases. The output actions `brakeOn`, `brakeOff` are replaced by `bufBrakeOn`, `bufBrakeOff`. The time bounds A, B, C are replaced by A', B', C'. These new bounds are:

$$\begin{aligned} A' &= \max(0, A - \delta^+) \\ B' &= B + \delta^+ - \delta^- \\ C' &= C - \delta^+ + \delta^- \end{aligned}$$

We also name the following compositions of automata:

$$\text{DEL-ONE-SHOT-AND-BUF} = \phi(\text{BUFFER}||\text{DEL-ONE-SHOT})$$

$$\text{DEL-ONE-SHOT-SYS} = \text{TRAIN}||\text{DEL-ONE-SHOT-AND-BUF}$$

4.5 Correctness of DEL-ONE-SHOT

The proof of correctness of the controller requires proofs of the timeliness and safety properties. First, we prove that the buffer never overflows in Section 4.5.1. In Section 4.5.2 we prove timeliness and safety using a simulation mapping to the unbuffered case of Chapter 3. The timeliness and safety results of the unbuffered case extend via the simulation to this case.

4.5.1 Non-Violation

Non-violation is proved directly.

Lemma 4.5.1 *In all reachable states of* DEL-ONE-SHOT-AND-BUF *the following holds:*

$$violation = \mathtt{false}.$$

Proof: Violation occurs when $request \neq \mathtt{none}$ and a `bufBrakeOn` or `bufBrakeOff` action takes place. Since these actions are controlled by the ON and OFF tasks it is sufficient to show that $first(ON)$ and $first(OFF)$ are greater than now whenever $request \neq \mathtt{none}$. The following invariant of DEL-ONE-SHOT-SYS is sufficient:

$$request \neq \mathtt{none} \Longrightarrow last(BUFF) \leq \min(first(ON), first(OFF))$$

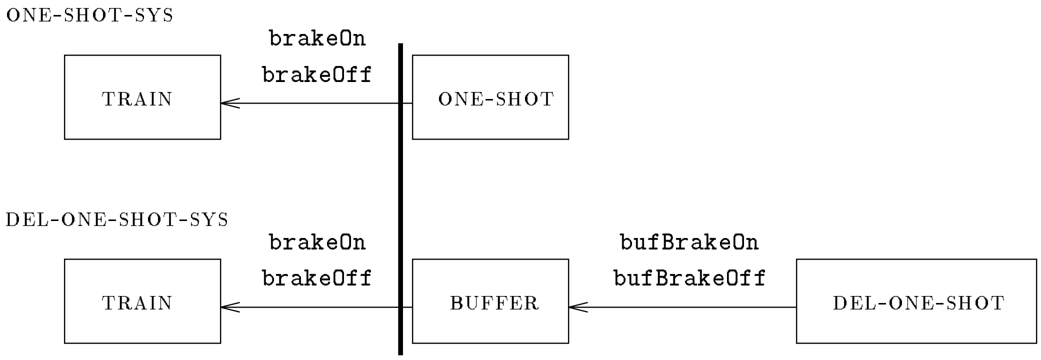
This follows from a simple inductive argument that uses the new constraints on the target velocities and the definition of B'. ■

4.5.2 Timeliness and Safety

In this section we prove the timeliness and safety properties for DEL-ONE-SHOT-SYS via a simulation mapping. The simulation maps states of DEL-ONE-SHOT-AND-BUF to states of the original controller, ONE-SHOT. Note that the use of the hiding operator ϕ in the definition of DEL-ONE-SHOT-AND-BUF makes the two automata comparable (Section 2.5). We use the simulation and Theorem 2.5.1 to show that DEL-ONE-SHOT-AND-BUF implements ONE-SHOT. Then, the timeliness and safety properties of DEL-ONE-SHOT follow from Theorem 2.6.2.

The intuition that suggests this type of proof is as follows: ONE-SHOT exhibits all possible behaviors that engage the brake exactly once and that satisfy the timeliness and safety properties. Therefore, the automaton ONE-SHOT is itself a form of specification for those behaviors — that is, every correct brake-controller which only engages the brake once is an implementation of ONE-SHOT. Since the example controller of this chapter, DEL-ONE-SHOT, only brakes once, we expect that it satisfies the timeliness and safety properties if and only if the composition of DEL-ONE-SHOT and BUFFER implements ONE-SHOT. One direction of the "if and only if" comes from

Figure 4-2 Comparison of ONE-SHOT-SYS and DEL-ONE-SHOT-SYS.



Theorem 3.4.1 and is the proof method we use. The other direction is based on our claim that ONE-SHOT exhibits all possible behaviors that engage the brake exactly once.

Notice that the safety and timeliness properties only mention variables in TRAIN. In light of this, it may appear counter-intuitive that the simulation mapping excludes the train. Consider Figure 4-2, which shows the automata and inter-automaton communication of ONE-SHOT-SYS and DEL-ONE-SHOT-SYS together. The dark vertical line represents a common interface in both systems, namely the interface to TRAIN. A consequence of our simulation mapping is that the external behavior of DEL-ONE-SHOT-AND-BUF is a subset of the external behavior of ONE-SHOT. Their external behavior is precisely the behavior across the dark line and this is all the input that TRAIN receives; therefore TRAIN's behavior in the buffered case is a subset of its behavior in the unbuffered case. Therefore, the timeliness and safety properties, which involve only variables of TRAIN, extend from the unbuffered case to the buffered case.

In the following three subsections we give some supporting lemmas, the simulation mapping, and then the final correctness result in Theorem 4.5.6.

Supporting Lemmas

The following lemma helps reduce the number of cases that need to be considered in the simulation proof.

Lemma 4.5.2 *In all reachable states of* DEL-ONE-SHOT-AND-BUF *exactly one of the following is true:*

1. $phase = \texttt{idle} \wedge request = \texttt{none}$
2. $phase = \texttt{braking} \wedge request = \texttt{on}$
3. $phase = \texttt{braking} \wedge request = \texttt{none}$

4. $phase = \texttt{done} \wedge request = \texttt{off}$

5. $phase = \texttt{done} \wedge request = \texttt{none}$

Furthermore, all transitions lead from a state in one category to a state in the same or immediately subsequent category.

Proof: Simple induction, uses Lemma 4.5.1. ■

The following two technical lemmas help make the simulation proof more readable. Both lemmas concern the time bounds on the idle phase.

Lemma 4.5.3 *In all reachable states of* DEL-ONE-SHOT*, the following holds:*

$$phase = \texttt{idle} \Longrightarrow first(ON) = 0 \wedge last(OFF) = A'$$

Proof: Exactly analogous to Lemma 3.6.1. ■

Lemma 4.5.4 *In all reachable states of* DEL-ONE-SHOT-AND-BUF *the following holds:*

$$(phase = \texttt{braking} \wedge request \neq \texttt{none}) \Longrightarrow last(BUFF) \leq A' + \delta^+ = A$$

Proof: Simple induction, uses Lemma 4.5.2. ■

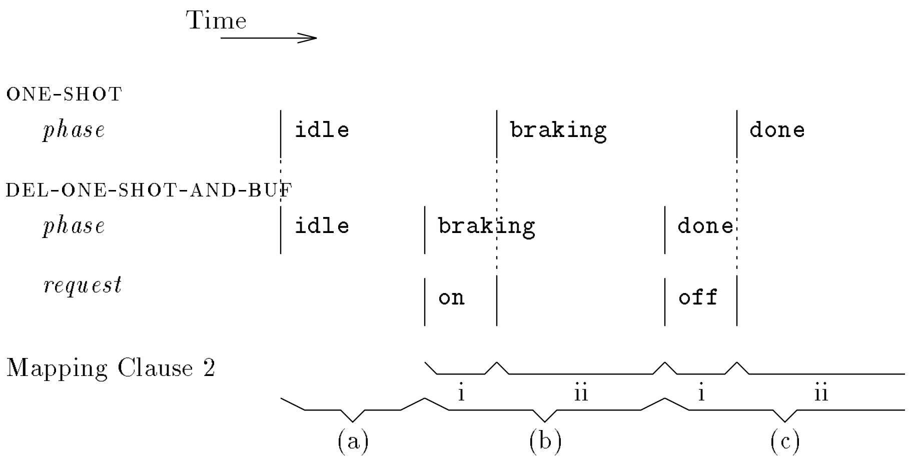
Simulation

In this section we present a simulation relation R from DEL-ONE-SHOT-AND-BUF to ONE-SHOT. The key insight is that since external behavior must be preserved, the timing of external actions must coincide, specifically `brakeOn` and `brakeOff`.

Let s denote a state in the implementation (DEL-ONE-SHOT-AND-BUF), and u denote a state in the specification (ONE-SHOT); the states are related via R (denoted sRu) when the following two conditions hold:

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

Figure 4-3 Overview of Simulation Mapping



i. `off`, then $u.phase =$ `braking` and
$u.first(OFF) \leq s.first(BUFF)$ and
$u.last(OFF) \geq s.last(BUFF)$

ii. `none`, then $u.phase =$ `done`

Intuitively, the simulation is mapping the "virtual" phases of the implementation, DEL-ONE-SHOT-AND-BUF, to the actual phases of the specification, ONE-SHOT. This is illustrated in Figure 4-3. The figure depicts an execution of ONE-SHOT above a corresponding execution of DEL-ONE-SHOT. A virtual phase of DEL-ONE-SHOTis the portion of its execution that corresponds to an actual phase of ONE-SHOT. For example the virtual idle phase consists of the period between the first and second dotted line. The second and third dotted lines represent the times when `brakeOn` and `brakeOff` actions occur, respectively. The figure also shows how mapping clause 2 applies to different portions of the execution.

The proof that the relation R is in fact a simulation mapping appears below. The form of simulation proofs is that of an exhaustive case analysis. To those familiar with the style of simulation proofs, this one is straightforward and unremarkable.

Lemma 4.5.5 *The above relation R is a simulation mapping from* DEL-ONE-SHOT-AND-BUF *to* ONE-SHOT.

Proof: Let s follow from s' in one discrete transition labeled by action π or in one trajectory and let sRu. We must find u' such that $s'Ru'$ and there exists an execution fragment from u to u' with the same trace as π. We break by cases depending on the type of step and its label:

1. If s leads to s' via a trajectory then we must show that there is an equivalent trajectory enabled from u. Since the barriers to time progress are the $last(\cdot)$ variables, it is sufficient to show that they are all greater in the specification. More exactly:

$$\begin{aligned} &min\{u.last(ON), u.last(OFF)\} \\ &\qquad\qquad \geq min\{s.last(ON), s.last(OFF), s.last(BUFF)\} \end{aligned}$$

Cases by $u.phase$:

   (a) $u.phase = \texttt{idle}$
   The OFF task is disabled in u so $u.last(OFF) = \infty$ and we are concerned only with $u.last(ON)$. From the relation R we can break into the following two cases:

      i. $s.phase = \texttt{idle}$ – then $s.last(OFF) = \infty$ and $s.last(BUFF) = \infty$ (by automaton definition and Lemma 4.5.2). By Lemmas 3.6.1 and 4.5.3 $u.last(ON) = A$ and $s.last(ON) = A'$ and by definition $A \geq A'$.

      ii. $s.phase = \texttt{braking} \wedge s.request \neq \texttt{none}$ – Follows from Lemmas 3.6.1 and 4.5.4.

   (b) $u.phase = \texttt{braking}$
   The ON task is disabled in u so $u.last(ON) = \infty$ and we are concerned only with $u.last(OFF)$. From the relation R we can break into the following two cases:

      i. $s.phase = \texttt{braking} \wedge s.request = \texttt{none}$ – then $s.last(ON) = \infty$ and $s.last(BUFF) = \infty$. By clause 2(b)ii of the relation $u.last(OFF) = s.last(OFF) + \delta^+$.

      ii. $s.phase = \texttt{done} \wedge s.request \neq \texttt{none}$ – then $s.last(ON) = s.last(OFF) = \infty$. By clause 2(c)i of the relation $u.last(OFF) = s.last(BUFF)$.

   (c) $u.phase = \texttt{done}$
   Trivial. Both tasks OFF and ON are disabled in u, so $u.last(OFF) = u.last(ON) = \infty$.

2. If π is `bufBrakeOn` then let $u' = u$ and the execution fragment be empty. We must show that $s'Ru'$. Note that $s.phase = \texttt{idle}$ by the definition of the DEL-ONE-SHOT automaton. Also note that $s.request = \texttt{none}$ by Lemma 4.5.1 (non-violation). The results follows by clause 2a of the relation.

3. If π is `bufBrakeOff` then it is similar to the previous case. We let $u' = u$ and the execution fragment is empty. It follows from clause 2(c)i that $s'Ru'$.

4. If π is `brakeOn` then let u' be the unique state that follows u via the `brakeOn` action and let the execution fragment contain only that action. We must show

that `brakeOn` is enabled in u and that $s'Ru'$. Note that $s.request =$ `on` by the definition of the BUFFER automaton. By Lemma 4.5.2 we know that $s.phase =$ `braking`. Therefore by clause $2a$ of the relation we know that $u.phase =$ `idle`. Since $u.first(ON) = 0$ by Lemma 3.6.1, `brakeOn` is enabled in u. It remains to show that u' satisfies the relation. Since s' satisfies the antecedent of clause 2(b)ii, u' must satisfy its consequent. By the definitions of B, B', C, C' it does.

5. If π is `brakeOff` then we proceed much as in the above case. Let u' be the unique state that follows u via the `brakeOff` action and let the execution fragment contain only that action. First, $s.request =$ `off` by the definition of the BUFFER automaton. By Lemma 4.5.2, $s.phase =$ `done`. By clause $2(c)i$ of the relation we know that $u.phase =$ `braking` and that $[u.first(OFF), u.last(OFF)] \supseteq [s.first(BUFF), s.last(BUFF)]$ and `brakeOff` is enabled in s, therefore it is enabled in u. Finally $s'Ru'$ by clause 2(c)ii.

These are all the cases of π. ■

Using the Simulation

In this section we use the above simulation to prove that DEL-ONE-SHOT is a correct buffered-brake-controller.

Theorem 4.5.6 *Automaton* DEL-ONE-SHOT *is a correct buffered-brake-controller.*

We must show that DEL-ONE-SHOT-AND-BUF is a correct brake-controller.
By Lemma 4.5.5 and Theorem 2.5.1:

$$\text{DEL-ONE-SHOT-AND-BUF} \leq \text{ONE-SHOT}$$

By Theorem 3.4.1 and Theorem 3.6.14 DEL-ONE-SHOT-AND-BUF is a correct brake-controller.

Chapter 5

Deceleration Case 3: Feedback and No Delay

In this chapter we describe a more complex model of the deceleration problem where the train provides the controller with sensor feedback at periodic intervals. We define a new train automaton called SENSOR-TRAIN. We also define correctness conditions, give an example controller and prove that it is correct. Figure 5-1 illustrates the components and their communication.

5.1 The SENSOR-TRAIN Automaton

The SENSOR-TRAIN automaton appears in Table 5.1. It accepts `accel`(a) messages which are requests to accelerate at a rate $a \in [\ddot{c}_{\text{min}} + \ddot{c}_{\text{err}}, \ddot{c}_{\text{max}}]$. If a is the requested acceleration then the achieved acceleration of the train is in the interval $[a - \ddot{c}_{\text{err}}, a]$. This is similar to the behavior of TRAIN from Section 3.2 in that the acceleration is non-deterministically chosen from an interval. It differs in that the controller can choose one of the endpoints of the fixed length interval and hence adjust the interval up or down. The train provides sensor information periodically; it sends a `status` message giving the current values of its variables *acc*, $\dot{x}$, and x every δ_{s} time units. The variable *acc* stores the most recent acceleration request. The variable *next* is a deadline variable which stores the time of the next `status` action.

Figure 5-1 Overview of Feedback Deceleration Model

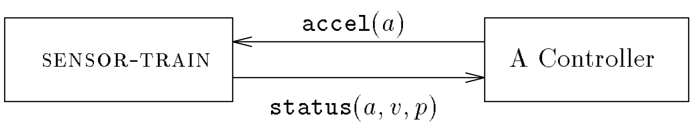


Table 5.1 The SENSOR-TRAIN automaton.

Actions: Inputs: $\texttt{accel}(a)$ for $a \in [\ddot{c}_{\text{min}} + \ddot{c}_{\text{err}}, \ddot{c}_{\text{max}}]$
Outputs: $\texttt{status}(a, v, p)$ for $a, v, p \in \mathbb{R}$
Vars: Outputs: $x \in \mathbb{R}$, initially $x = c_{\text{s}}$
$\dot{x} \in \mathbb{R}$, initially $\dot{x} = \dot{c}_{\text{s}}$
$\ddot{x} \in \mathbb{R}$, initially $\ddot{x} = \ddot{c}_{\text{s}}$
$acc \in [\ddot{c}_{\text{min}} + \ddot{c}_{\text{err}}, \ddot{c}_{\text{max}}]$, initially $\ddot{c}_{\text{s}}$
$next \in \mathbb{R}^{\geq 0}$, initially 0
$now \in \mathbb{R}^{\geq 0}$, initially 0

Discrete Transitions:
$\texttt{accel}(a)$:
Eff: $acc := a$
$\ddot{x} :\in [a - \ddot{c}_{\text{err}}, a]$
$\texttt{status}(a, v, p)$:
Pre: $a = acc$, $v = \dot{x}$, $p = x$ and $now = next$
Eff: $next := now + \delta_{\text{s}}$

Trajectories:
$w.acc$ and $w.next$ are constant functions
$w.\ddot{x}$ is an integrable function with range $[w(0).acc - \ddot{c}_{\text{err}}, w(0).acc]$
For all $t \in I$ the following hold:
$w(t).now = w(0).now + t$
$w(t).now \leq next$
$w(t).\dot{x} = w(0).\dot{x} + \int_0^t w(s).\ddot{x}\, ds$
$w(t).x = w(0).x + \int_0^t w(s).\dot{x}\, ds$

5.2 Properties of SENSOR-TRAIN

The following two properties of SENSOR-TRAIN are similar to the properties of TRAIN proved in Lemmas 3.3.2 and 3.3.4. The first bounds change in velocity by change in time. The second bounds change in position by change in velocity.

Lemma 5.2.1 *For all closed trajectories w of* SENSOR-TRAIN *where s is the initial and s' is the final state of w the following holds:*

$$acc(now' - now) \geq \dot{x}' - \dot{x} \geq (acc - \ddot{c}_{\text{err}})(now' - now)$$

Proof: As in the first part of Lemma 3.3.2, except that acc and $(acc - \ddot{c}_{\text{err}})$ replace $\ddot{c}_{\text{max}}$ and $\ddot{c}_{\text{min}}$ respectively. ■

Lemma 5.2.2 *For all closed trajectories w of* SENSOR-TRAIN *where s is the initial and s' is the final state of w, if $acc \leq 0$ and $0 < \dot{x}'$ then the following holds:*

$$\frac{(\dot{x}')^2 - \dot{x}^2}{2acc} \geq x' - x \geq \frac{(\dot{x}')^2 - \dot{x}^2}{2(acc - \ddot{c}_{\text{err}})}$$

Proof: Similar to Lemma 3.3.4, except that acc and $(acc - \ddot{c}_{\text{err}})$ replace $\ddot{c}_{\max}$ and $\ddot{c}_{\min}$ respectively. ∎

The following property is like the $now \leq last(\cdot)$ property for MMT automata, Theorem 2.8.1.

Lemma 5.2.3 *In all reachable states of* SENSOR-TRAIN *the following holds:*

$$0 \leq next - now \leq \delta_{\text{s}}$$

Proof: Simple induction. ∎

5.3 Definition of Controller Correctness, Revisited

We define a correct *controller-under-feedback* to be a hybrid I/O automaton with no external variables and with output actions $\texttt{accel}(a)$ for $a \in [\ddot{c}_{\min} + \ddot{c}_{\text{err}}, \ddot{c}_{\max}]$ that when composed with SENSOR-TRAIN yields an automaton whose hybrid traces satisfy the timeliness and safety properties from Section 3.4. These are restated here for convenience:

Timeliness There exists a constant $t \in \mathbb{R}^{\geq 0}$ such that for all hybrid traces if there exists a state of the trace in which $now = t$, then there is a state of the trace in which $x = c_{\text{f}}$.

Safety In all states of all hybrid traces the following holds:
$x = c_{\text{f}} \Longrightarrow \dot{c}_{\text{minf}} \leq \dot{x} \leq \dot{c}_{\text{maxf}}$.

5.4 Parameters, Revisited

In order to guarantee that a valid controller exists, we impose the following constraints on the parameters:

1. $c_{\text{s}} < c_{\text{f}}$
2. $\dot{c}_{\text{s}} > \dot{c}_{\text{maxf}} \geq \dot{c}_{\text{minf}} > 0$
3. $\ddot{c}_{\text{err}} > 0$
4. $\delta_{\text{s}} > 0$

5. $\ddot{c}_{\rm min} < \ddot{c}_{\rm min} + \ddot{c}_{\rm err} < 0 \leq \ddot{c}_{\rm max} - \ddot{c}_{\rm err} < \ddot{c}_{\rm max}$
6. $c_{\rm f} - c_{\rm s} \geq \frac{\dot{c}^2_{\rm maxf} - \dot{c}^2_{\rm s}}{2(\ddot{c}_{\rm min} + \ddot{c}_{\rm err})}$
7. $\dot{c}_{\rm maxf} - \dot{c}_{\rm minf} \geq -\ddot{c}_{\rm min}\delta_{\rm s}$

Note that these constraints supersede the original constraints given in Chapter 3. Informally the constraints say the following: (1) the final position is past the initial position; (2) the task is to decelerate the train to a well-defined interval but not to reverse the train; (3) the uncertainty in acceleration is non-zero; (4) the interval between sensor observations is non-zero; (5) certain commands to the train can guarantee periods of strictly negative or non-negative acceleration; (6) there is enough distance to brake, given the weakest braking that can occur after a request for the strongest braking; (7) the target interval of velocities is wide enough. Constraint 7 is only one of a number of constraints that make the target velocity interval wide enough for there to be *some* correct controller. We chose this form of constraint 7 because it is necessary for the correctness of the example controller of this chapter.

Recall that in the description of SENSOR-TRAIN the initial values of both *acc* and $\ddot{x}$ are set to $\ddot{c}_{\rm s}$. In order to avoid a tedious treatment of certain initial conditions, we assume that the train is initially at a convenient acceleration. Let $\ddot{c}_{\rm s}$ be the acceleration needed to reach $\dot{c}_{\rm maxf}$ at exactly $c_{\rm f}$, as follows:

$$\ddot{c}_{\rm s} = \frac{\dot{c}^2_{\rm maxf} - \dot{c}^2_{\rm s}}{2(c_{\rm f} - c_{\rm s})}$$

Notice that $\ddot{c}_{\rm s}$ is negative.

5.5 Example Controller: ZIG-ZAG

Controlling the train in the presence of sensory feedback appears to require a substantially different algorithm from that in the non-feedback case. Here we give an example valid *controller-under-feedback* called ZIG-ZAG. The system composed of SENSOR-TRAIN and ZIG-ZAG is called ZIG-ZAG-SYS. We describe ZIG-ZAG in Table 5.2.

We explain informally the behavior of ZIG-ZAG. The controller takes no action unless it receives a `status`(a, v, p) message in which $v \leq \dot{c}_{\rm maxf}$; this is guaranteed to occur eventually and before the final position because of our choice of the initial negative acceleration $\ddot{c}_{\rm s}$. This is an arbitrary choice in the design of ZIG-ZAG — there are other correct controllers that adjust the acceleration earlier. Once the controller is informed that the velocity of the train in below $\dot{c}_{\rm maxf}$, it immediately send an `accel`(a) message where a is the acceleration which will accelerate the train from its current velocity to $\dot{c}_{\rm maxf}$ in $\delta_{\rm s}$ time (if that acceleration is higher than $\ddot{c}_{\rm max}$, the largest allowed value of a, then it uses $\ddot{c}_{\rm max}$.) . If the train doesn't achieve the requested acceleration then the velocity in $\delta_{\rm s}$ time will be less than $\dot{c}_{\rm maxf}$. Constraint 7 on the parameters

Table 5.2 The ZIG-ZAG automaton.

Actions: Inputs: $\texttt{status}(a, v, p)$ for $a, v, p \in \mathbb{R}$
Outputs: $\texttt{accel}(a)$ for $a \in [\ddot{c}_{\mathrm{min}} + \ddot{c}_{\mathrm{err}}, \ddot{c}_{\mathrm{max}}]$
Vars: Internal: $send \in [\ddot{c}_{\mathrm{min}} + \ddot{c}_{\mathrm{err}}, \ddot{c}_{\mathrm{max}}] \cup \{\texttt{none}\}$, initially $\texttt{none}$

Discrete Transitions:

$\texttt{status}(a, v, p)$:
Eff:
if $v \leq \dot{c}_{\mathrm{maxf}}$ then
$send := \min\left(\ddot{c}_{\mathrm{max}}, \frac{\dot{c}_{\mathrm{maxf}} - v}{\delta_{\mathrm{s}}}\right)$

$\texttt{accel}(a)$:
Pre: $send = a$
Eff: $send := \texttt{none}$

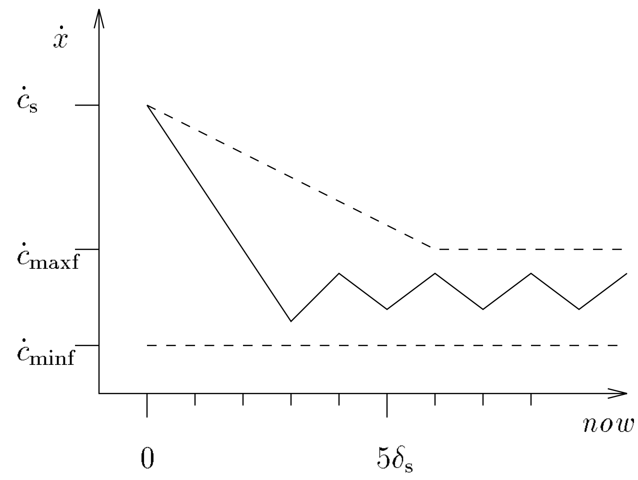
Trajectories:

$w.send$ is a constant function
if w is not a trivial trajectory then
$w(0).send = \texttt{none}$
for all $t \in I$ the following holds:
$w(t).now = w(0).now + t$

from Section 5.4 is sufficient to ensure that the interval $[\dot{c}_{\mathrm{maxf}}, \dot{c}_{\mathrm{minf}}]$ is wide enough that this strategy doesn't cause the velocity to dip below $\dot{c}_{\mathrm{minf}}$. In the definition of the trajectory set, the first "if" statement ensures that time progresses only if the controller has nothing to send.

The controller is called ZIG-ZAG because of the shape of the curve in $\dot{x} \times now$ space of the worst-case behavior of ZIG-ZAG-SYS (recall that ZIG-ZAG-SYS is the composition of SENSOR-TRAIN and ZIG-ZAG). Figure 5-2 depicts a possible behavior for the system; it assumes constant acceleration. The train begins at time zero with velocity $\dot{c}_{\mathrm{s}}$ and acceleration $\ddot{c}_{\mathrm{s}}$. If it achieved $\ddot{c}_{\mathrm{s}}$ acceleration it would reach the goal velocity of $\dot{c}_{\mathrm{maxf}}$ at exactly c_{f} (the upper dotted line). However, for the first three δ_{s} periods it only achieves $\ddot{c}_{\mathrm{s}} - \ddot{c}_{\mathrm{err}}$ acceleration (the solid line). At that point the controller sees that $\dot{x} \leq \dot{c}_{\mathrm{maxf}}$ and changes the acceleration (first bend in solid line). Every δ_{s} time units the controller continues to adjust acceleration so that the highest it will reach is $\dot{c}_{\mathrm{maxf}}$.

Figure 5-2 Possible behavior of ZIG-ZAG-SYS.



5.6 Correctness of ZIG-ZAG

The structure of the proof is very similar to that of the simple case examined in Chapter 3: first, we show the timeliness property via a global lower bound on velocity; second, we show the safety property via a more complex invariant that has as a subcase the invariant used in Chapter 3.

5.6.1 Timeliness

In this section we prove the timeliness property. The first lemma is a technical lemma that says that whenever the controller is going to send a new acceleration, there is δ_s time until the next `status` message. This is obvious because the `status` messages are sent at δ_s intervals and the controller responds to them immediately.

Lemma 5.6.1 *In all reachable states of* ZIG-ZAG-SYS *the following holds:*

$$send \neq \texttt{none} \Longrightarrow next = now + \delta_s$$

Proof: Trivial induction. ■

The next lemma, Lemma 5.6.2, is the major new result needed to prove the timeliness property. As in Chapter 3, we would like to prove the timeliness property with the invariant $\dot{x} \geq \dot{c}_{minf}$. However, this invariant cannot be proved directly with an inductive argument. Once again, we strengthen the invariant to yield an invariant assertion that can be proved inductively; the weaker invariant follows as a corollary.

The stronger invariant appears in Lemma 5.6.2. It is an invariant that describes a lower bound on velocity at the current time *and* for the near future — the current sensory interval. This property uses a set of implications with mutually exclusive and exhaustive antecedents. Each implication corresponds to one of the periodic logical phases of the system: $send = \texttt{none}$, when the ZIG-ZAG is waiting for the next `status` message; and $send \neq \texttt{none}$, when ZIG-ZAG has just received a `status` message and is about to send a new `accel` command. The invariant makes a different claim for each of these phases. On the one hand, the invariant says that if $send = \texttt{none}$ then the current velocity is above $\dot{c}_{\text{minf}}$ and the velocity at the time of the next status message will be also. The worst-case velocity at the time of the next status message is calculated using the current lower bound on acceleration, $acc - \ddot{c}_{\text{err}}$, and the time left until the next status message, $next - now$. This type of calculation appears again in more complex forms in subsequent sections and chapters. On the other hand, the invariant says that if $send \neq \texttt{none}$ then the current velocity is above $\dot{c}_{\text{minf}}$ and the velocity at the time of the next status message will be also. In this case, the worst-cast velocity at the time of the next status message is calculated using the acceleration that the controller is about to send the train, namely the variable *send* itself. This type of invariant appears again later in more complex forms.

Lemma 5.6.2 *In all reachable states of* ZIG-ZAG-SYS, *the following hold:*

1. $send = \texttt{none} \Longrightarrow \dot{x} \geq \dot{c}_{\text{minf}} \wedge \dot{x} + (acc - \ddot{c}_{\text{err}})(next - now) \geq \dot{c}_{\text{minf}}$
2. $send \neq \texttt{none} \Longrightarrow \dot{x} \geq \dot{c}_{\text{minf}} \wedge \dot{x} + (send - \ddot{c}_{\text{err}})\delta_{\text{s}} \geq \dot{c}_{\text{minf}}$

Proof: By induction. Notice that the antecedents of the two implications are mutually exclusive and exhaustive; we will refer to them as Rule 1 and 2. We say that a rule *applies* when it's antecedent is true and that it *holds* when it applies and its consequent is true (or when it doesn't apply).

Basis: In the initial state $\dot{x} > \dot{c}_{\text{maxf}}$ so Rule 1 applies. It holds because of our assumptions on the parameters, the definition of $\ddot{c}_{\text{s}}$, and the definition of the initial states of the automata.

Induction: Suppose the property is true in state s; we must show that it is true in s' which follows from s in one discrete transition labeled by action π or in one trajectory. For the sake of brevity, we denote variables in the post-state by adding primes, e.g. we write now' instead of $s'.now$. We brake by cases on the type of step and its label: `accel`, `status`, or trajectory.

1. $\pi = \texttt{accel}$: notice that $send \neq \texttt{none}$ by the action's precondition, so Rule 2 applies in s and by the inductive hypothesis it holds. The only variables which change are *send* and *acc*; the action sets $acc' = send$ and $send' = \texttt{none}$. Therefore Rule 1 must apply in s'. We must show that it holds. Clearly, $\dot{x}' = \dot{x} \geq \dot{c}_{\text{minf}}$ by the inductive hypothesis. By Lemma 5.6.1 $next - now = \delta_{\text{s}}$ and because none of these variables change $next' - now' = \delta_{\text{s}}$. By substituting $next' - now' = \delta_{\text{s}}$, $acc' = send$ and $send' = \texttt{none}$ into the inequality in Rule 2 we get :

$\dot{x} + (acc' - \ddot{c}_{\mathrm{err}})(next' - now') = \dot{x} + (send - \ddot{c}_{\mathrm{err}})\delta_{\mathrm{s}} \geq \dot{c}_{\mathrm{minf}}$
This shows that Rule 1 holds in s'.

2. π = `status`: notice that $next = now$ by the action's precondition, so $next \neq now + \delta_{\mathrm{s}}$ and by the contra-positive of Lemma 5.6.1 $send$ = `none`; therefore, Rule 1 applies in s and by the inductive hypothesis it holds. The only variables which change are $send$ and $next$. We break by cases of $send'$:

   (a) $send'$ = `none`: Rule 1 applies in s'; must show that it holds. According to the automata definitions $\dot{x}' = \dot{x} = v \geq \dot{c}_{\mathrm{maxf}}$, $next' - now' = \delta_{\mathrm{s}}$, and $acc' - \ddot{c}_{\mathrm{err}} \geq \ddot{c}_{\mathrm{min}}$. By assumption on the parameters: $\dot{c}_{\mathrm{maxf}} - \dot{c}_{\mathrm{minf}} > -\ddot{c}_{\mathrm{min}}\delta_{\mathrm{s}}$. From these, we reach the desired conclusion with some algebra:

$$
\begin{array}{rcll}
\dot{c}_{\mathrm{maxf}} - \dot{c}_{\mathrm{minf}} & \geq & -\ddot{c}_{\mathrm{min}}\delta_{\mathrm{s}} & \text{parameter assumption} \\
\dot{c}_{\mathrm{maxf}} + \ddot{c}_{\mathrm{min}}\delta_{\mathrm{s}} & \geq & \dot{c}_{\mathrm{minf}} & \text{subtract} \\
\dot{x}' & \geq & \dot{c}_{\mathrm{maxf}} & \text{automaton definition} \\
\dot{x}' + \ddot{c}_{\mathrm{min}}\delta_{\mathrm{s}} & \geq & \dot{c}_{\mathrm{minf}} & \text{substitute} \\
\delta_{\mathrm{s}} & > & 0 & \text{parameter assumption} \\
acc' - \ddot{c}_{\mathrm{err}} & \geq & \ddot{c}_{\mathrm{min}}. & \text{automaton definition} \\
\dot{x}' + (acc' - \ddot{c}_{\mathrm{err}})\delta_{\mathrm{s}} & \geq & \dot{c}_{\mathrm{minf}} & \text{substitution} \\
\delta_{\mathrm{s}} & = & next' - now' & \text{automaton definition} \\
\dot{x}' + (acc' - \ddot{c}_{\mathrm{err}})(next' - now') & \geq & \dot{c}_{\mathrm{minf}} & \text{substitution}
\end{array}
$$

   Thus Rule 1 holds in s'.

   (b) $send' \neq$ `none`: Rule 2 applies in s'; must show that it holds. Above we showed that $next = now$ and Rule 1 holds in state s from which we know that $\dot{x} \geq \dot{c}_{\mathrm{minf}}$. This is half of Rule 2; it remains to show the other half. According to the automata definitions: $send' = \min(\ddot{c}_{\mathrm{max}}, \frac{\dot{c}_{\mathrm{maxf}} - \dot{x}}{\delta_{\mathrm{s}}})$. By assumption on the parameters $\ddot{c}_{\mathrm{max}} - \ddot{c}_{\mathrm{err}} \geq 0$, therefore if $send' = \ddot{c}_{\mathrm{max}}$ Rule 2 applies trivially. Assume that $send' = \frac{\dot{c}_{\mathrm{maxf}} - \dot{x}}{\delta_{\mathrm{s}}} < \ddot{c}_{\mathrm{max}}$. Some algebra yields the desired result:

$$
\begin{array}{rcll}
\ddot{c}_{\mathrm{min}} + \ddot{c}_{\mathrm{err}} & < & 0 & \text{parameter assumption} \\
\delta_{\mathrm{s}} & > & 0 & \text{parameter assumption} \\
-\ddot{c}_{\mathrm{min}}\delta_{\mathrm{s}} & > & \ddot{c}_{\mathrm{err}}\delta_{\mathrm{s}} & \text{subtract \& multiply} \\
\dot{c}_{\mathrm{maxf}} - \dot{c}_{\mathrm{minf}} & \geq & -\ddot{c}_{\mathrm{min}}\delta_{\mathrm{s}} & \text{parameter assumption} \\
\dot{c}_{\mathrm{maxf}} - \dot{c}_{\mathrm{minf}} & \geq & \ddot{c}_{\mathrm{err}}\delta_{\mathrm{s}} & \text{transitivity} \\
\dot{c}_{\mathrm{maxf}} - \ddot{c}_{\mathrm{err}}\delta_{\mathrm{s}} & \geq & \dot{c}_{\mathrm{minf}} & \text{subtract} \\
\dot{x}' + \dot{c}_{\mathrm{maxf}} - \dot{x}' - \ddot{c}_{\mathrm{err}}\delta_{\mathrm{s}} & \geq & \dot{c}_{\mathrm{minf}} & \text{anti-cancel} \\
\dot{x}' + \left(\frac{\dot{c}_{\mathrm{maxf}} - \dot{x}}{\delta_{\mathrm{s}}} - \ddot{c}_{\mathrm{err}}\right)\delta_{\mathrm{s}} & \geq & \dot{c}_{\mathrm{minf}} & \text{anti-distribute} \\
send' & = & \frac{\dot{c}_{\mathrm{maxf}} - \dot{x}}{\delta_{\mathrm{s}}} & \text{assumption} \\
\dot{x}' + (send' - \ddot{c}_{\mathrm{err}})\delta_{\mathrm{s}} & \geq & \dot{c}_{\mathrm{minf}} & \text{substitute}
\end{array}
$$

   Thus Rule 2 holds in s'.

3. The step is a trajectory: then $send = send'$ = `none` according to the trajectories of the controller. Thus, Rule 1 holds in s, applies in s' and must be shown

to hold in s'. This case uses Lemma 5.2.1, the inductive hypothesis and some simple algebra.
Notice that $acc = acc'$, so let $X = (acc - \ddot{c}_{\mathrm{err}}) = (acc' - \ddot{c}_{\mathrm{err}})$:

$$\begin{array}{rcll} \dot{x} + X(next - now) & \geq & \dot{c}_{\mathrm{minf}} & \text{inductive hypothesis} \\ \dot{x}' - \dot{x} & \geq & X(now' - now) & \text{by Lemma 5.2.1} \\ \dot{x}' - \dot{x} - X(now' - now) & \geq & 0 & \text{subtract} \\ \dot{x} + \dot{x}' - \dot{x} + X(next - now) & & & \\ -X(now' - now) & \geq & \dot{c}_{\mathrm{minf}} & \text{add} \\ \dot{x}' + X(next - now') & \geq & \dot{c}_{\mathrm{minf}} & \text{cancel} \end{array}$$

For the $\dot{x} \geq \dot{c}_{\mathrm{minf}}$ requirement: by Lemma 5.2.3 $next - now \geq 0$, thus if $X \geq 0$ then $\dot{x}' \geq \dot{x} \geq \dot{c}_{\mathrm{minf}}$; otherwise, $\dot{x}' \geq \dot{x}' + X(next - now') \geq \dot{c}_{\mathrm{minf}}$ (by Lemma 5.2.3). Thus Rule 1 holds in s'.

■

Corollary 5.6.3 *In all reachable states of* ZIG-ZAG-SYS *the following holds:*

$$\dot{x} \geq \dot{c}_{\mathrm{minf}}$$

Proof: Directly from 5.6.2. The antecedents form an exhaustive set of cases, and in all cases the property is true. ■

This leads to the timeliness property as Lemma 3.6.6 did in Chapter 3. The corollaries which yield the timeliness property are exactly analogous and are not restated here. The final result is stated in Theorem 5.6.8.

5.6.2 Safety

The following technical lemma is says that under certain conditions a certain inequality is maintained during trajectories. Informally, the inequality tests whether there remains enough distance to brake the train to below $\dot{c}_{\mathrm{maxf}}$. This inequality appeared extensively in the proof of the safety property in Section 3.6.2.

Lemma 5.6.4 *Let w be a closed trajectory of* ZIG-ZAG-SYS *where s is the initial state and s' is the final state of w. If $acc = \ddot{c}_{\mathrm{s}}$, $x \leq c_{\mathrm{f}}$, and $x' \leq c'_{\mathrm{f}}$ then*

$$c_{\mathrm{f}} - x \geq \frac{\dot{c}^2_{\mathrm{maxf}} - \dot{x}^2}{2\ddot{c}_{\mathrm{s}}} \Longrightarrow c_{\mathrm{f}} - x' \geq \frac{\dot{c}^2_{\mathrm{maxf}} - (\dot{x}')^2}{2\ddot{c}_{\mathrm{s}}}$$

Proof: The proof is similar to those in Section 3.6.2.

$$
\begin{array}{rcll}
acc & = & \ddot{c}_{\mathrm{s}} \leq 0 & \text{assumption} \\
c_{\mathrm{f}} - x & \geq & \frac{\dot{c}^2_{\mathrm{maxf}} - \dot{x}^2}{2\ddot{c}_{\mathrm{s}}} & \text{assumption} \\
x' - x & \leq & \frac{(\dot{x}')^2 - \dot{x}^2}{2acc} & \text{by Lemma 5.2.2} \\
x - x' & \geq & \frac{\dot{x}^2 - (\dot{x}')^2}{2acc} & \text{multiply} \\
c_{\mathrm{f}} - x + x - x' & \geq & \frac{\dot{c}^2_{\mathrm{maxf}} - \dot{x}^2 + \dot{x}^2 - (\dot{x}')^2}{2acc} & \text{add} \\
c_{\mathrm{f}} - x' & \geq & \frac{\dot{c}^2_{\mathrm{maxf}} - (\dot{x}')^2}{2\ddot{c}_{\mathrm{s}}} & \text{cancel}
\end{array}
$$

■

The following lemma is the major result needed to prove the safety property. It is similar to two other results: (1) Corollary 3.6.12 and its supporting lemmas, which used a similar equation to bound "distance remaining"; and, (2) Lemma 5.6.2 of this section, which provides a set of implication with an exhaustive set of antecedents. Each of the clauses can be associated with a portion of the solid line in the graph in Figure 5-2. The first clause applies to the initial downward segment; it says that before passing the $\dot{c}_{\mathrm{maxf}}$ threshold the following hold: the acceleration $\ddot{c}_{\mathrm{s}}$ is in effect; the controller is not sending any commands; and there is enough distance left to brake at the current acceleration. The second and third clauses guarantee that once the velocity has dipped below $\dot{c}_{\mathrm{maxf}}$, it will never rise above $\dot{c}_{\mathrm{maxf}}$. These clauses guarantee an upper bound in a manner analogous to the clauses of Lemma 5.6.2 which guaranteed a lower bound.

Lemma 5.6.5 *In all reachable states of* ZIG-ZAG-SYS *the following hold:*

1. $\dot{x} > \dot{c}_{\mathrm{maxf}} \Longrightarrow acc = \ddot{c}_{\mathrm{s}} \wedge send = \mathtt{none} \wedge \left((x \leq c_{\mathrm{f}}) \Longrightarrow c_{\mathrm{f}} - x \geq \frac{\dot{c}^2_{\mathrm{maxf}} - \dot{x}^2}{2\ddot{c}_{\mathrm{s}}}\right)$
2. $\dot{x} \leq \dot{c}_{\mathrm{maxf}} \wedge send = \mathtt{none} \Longrightarrow (\dot{x} + acc(next - now)) \leq \dot{c}_{\mathrm{maxf}}$
3. $\dot{x} \leq \dot{c}_{\mathrm{maxf}} \wedge send \neq \mathtt{none} \Longrightarrow (\dot{x} + send(\delta_{\mathrm{s}})) \leq \dot{c}_{\mathrm{maxf}}$

Proof: This is an inductive proof very similar to the proof of Lemma 5.6.2 above. As in that lemma, the property is the conjunction of a set of implications whose antecedents are mutually exclusive and exhaustive. We use similar terminology here, calling them Rules 1, 2, and 3. Notice that Rules 2 and 3 are analogous to Rules 1 and 2 of the previous lemma except that they guarantee an upper bound instead of a lower bound. We omit portions of this proof which are directly analogous.

Basis: In the initial state Rule 1 applies and is satisfied trivially. Induction: Suppose the property is true in state s; we must show that it is true in s' which follows from s in one step — either a discrete step labeled by action π or a trajectory. For the sake of brevity, we denote variables in the post-state by adding primes, e.g. we write now' instead of $s'.now$. We brake by cases on the type of step and the label π: `accel`, `status`, or trajectory.

1. $\pi =$ `accel`: Either $\dot{x} \leq \dot{c}_{\mathrm{maxf}}$ or not.

(a) $\dot{x} \leq \dot{c}_{\mathrm{maxf}}$: This case is exactly analogous to the $\pi =$ `accel` case of the proof of Lemma 5.6.2. Here, Rule 3 holds in state s and Rule 2 is shown to hold in state s'. We abbreviate the proof by noting that $acc' = send$ and $next' - now' = \delta_{\mathrm{s}}$.

(b) $\dot{x} > \dot{c}_{\mathrm{maxf}}$: by the inductive hypothesis Rule 1 holds in s and therefore $send =$ `none`; however in that case, this action was not enabled in s. Therefore $\dot{x} > \dot{c}_{\mathrm{maxf}}$ is impossible for the `accel` action case.

2. $\pi =$ `status`: Either $\dot{x} \leq \dot{c}_{\mathrm{maxf}}$ or not.

   (a) $\dot{x} \leq \dot{c}_{\mathrm{maxf}}$: This case is exactly analogous to the $\pi =$ `status` case of the proof of Lemma 5.6.2. Here, Rule 2 holds in state s and Rule 3 can be shown to hold in state s'. We omit the proof.

   (b) $\dot{x} > \dot{c}_{\mathrm{maxf}}$: Thus, Rule 1 holds in states s. By the automata definitions only variable $next$ changes as a result of this action (because $\dot{x} > \dot{c}_{\mathrm{maxf}}$). Since $next$ does not appear in Rule 1, it must continue to hold in state s'.

3. The step is a trajectory: Either $\dot{x} \leq \dot{c}_{\mathrm{maxf}}$

   (a) $\dot{x} \leq \dot{c}_{\mathrm{maxf}}$: This case is exactly analogous to the trajectory in the proof of Lemma 5.6.2. Here, Rule 2 holds in state s and can be shown to also hold in state s'. We omit the proof.

   (b) $\dot{x} > \dot{c}_{\mathrm{maxf}}$: Thus, Rule 1 holds in states s. By the definition of automata, we know that only the variables now, $\ddot{x}$, $\dot{x}$, and x are modified by this action. Therefore, we know that $acc' = acc = \ddot{c}_{\mathrm{s}}$ and $send' = send =$ `none`. There are two cases, either Rule 1 holds in s' or Rule 2 does.

      i. $\dot{x}' > \dot{c}_{\mathrm{maxf}}$: Rule 1 applies in s' and we must show that it holds. This is guaranteed by Lemma 5.6.4.

      ii. $\dot{x}' \leq \dot{c}_{\mathrm{maxf}}$: Rule 2 applies in s'. Note that $acc' = \ddot{c}_{\mathrm{s}}$ is negative, while $(next - now')$ is always positive by Lemma 5.2.3. Since $\dot{x}' \leq \dot{c}_{\mathrm{maxf}}$, we know $\dot{x}' + acc'(next - now') \leq \dot{c}_{\mathrm{maxf}}$. Therefore Rule 2 holds in s'.

∎

The following corollaries correspond directly to Corollaries 3.6.12 and 3.6.13.

Corollary 5.6.6 *In all reachable states of* ZIG-ZAG-SYS *the following holds:*

$$(x \leq c_{\mathrm{f}}) \Longrightarrow c_{\mathrm{f}} - x \geq \frac{\dot{c}_{\mathrm{maxf}}^2 - \dot{x}^2}{2\ddot{c}_{\mathrm{s}}}$$

Proof: Directly from 5.6.5. If the first implication applies, then it appears in the consequent. If the second implication or third applies, then $\dot{c}_{\mathrm{maxf}}^2 - \dot{x}^2$ is positive, hence, the fraction is negative and the inequality holds. These cases are exhaustive. ∎

The following corollary establishes the safety property.

Corollary 5.6.7 *In all reachable states of* ZIG-ZAG-SYS *the following holds:*

$$c_{\mathrm{f}} = x \Longrightarrow \dot{c}_{\mathrm{maxf}} \geq \dot{x} \geq \dot{c}_{\mathrm{minf}}$$

Proof: Directly from 5.6.6 and 5.6.3. ■

We summarize the correctness results in the following theorem.

Theorem 5.6.8 *Automaton* ZIG-ZAG *is a correct controller-under-feedback.*

Proof: We must show that the hybrid traces of ZIG-ZAG-SYS satisfy the timeliness and safety properties (see Section 5.3). As mentioned at the end of Section 5.6.1, the timeliness property follows from Corollary 5.6.3 just as it did from Lemma 3.6.6 in Chapter 3. We have omitted the intermediate results. Corollary 5.6.7 is exactly the safety property. ■

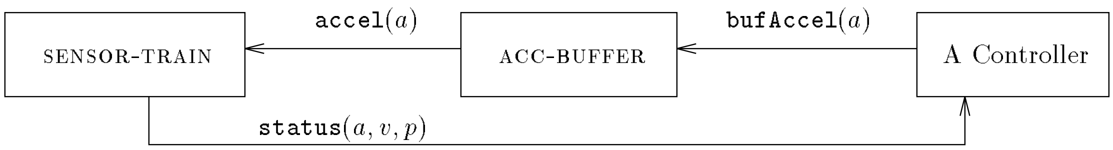
Chapter 6

Deceleration Case 4: Delay and Feedback

In this chapter we combine periodic sensor feedback and command delay. As in Chapter 4, we introduce delay via a buffer called ACC-BUFFER. We make no modification to the SENSOR-TRAIN automaton. We define a notion of a correct controller for this buffered system. We give an example of a correct controller called DEL-ZIG-ZAG that involves only minor modifications to the ZIG-ZAG controller of Chapter 5. Figure 6-1 illustrates the components and their communication.

In Chapter 4, we use a simulation based argument to prove that the composition of DEL-ONE-SHOT and BUFFER implements ONE-SHOT, the highly nondeterministic controller of Chapter 3. One might expect a similar development in this chapter — namely that we use a simulation proof to show that the composition of DEL-ZIG-ZAG and ACC-BUFFER implements ZIG-ZAG, the controller of Chapter 5. This is not the case; we prove the correctness of DEL-ZIG-ZAG directly. In fact, no simulation proof is possible because the composition of any controller and ACC-BUFFER can not implement ZIG-ZAG. Informally this is clear because ACC-BUFFER will introduce a delay between the time when the train gives the controller sensor input and when the train receives the related command. No such delay occurs for ZIG-ZAG — it responds to each sensor input without delay. There remains the question of whether some other choice of example controllers could have enabled the use of a simulation proof in this chapter in a manner analogous to Chapter 4. We address that issue in Chapter 7.

Figure 6-1 Overview of Feedback with Delay Deceleration Model



6.1 The ACC-BUFFER Automaton

The buffer, called ACC-BUFFER, has much the same structure as that of Chapter 4. It appears in Table 6.1 as an MMT-specification.

Table 6.1 The ACC-BUFFER automaton.

Actions: Inputs: `bufAccel`(a) for $a \in [\ddot{c}_{\min} + \ddot{c}_{\text{err}}, \ddot{c}_{\max}]$
Outputs: `accel`(a) for $a \in [\ddot{c}_{\min} + \ddot{c}_{\text{err}}, \ddot{c}_{\max}]$
Vars: Internal: $request \in [\ddot{c}_{\min} + \ddot{c}_{\text{err}}, \ddot{c}_{\max}] \cup \{$`none`$\}$, initially `none`
violation, boolean, initially `false`

Discrete Transitions:

`bufAccel`(a):
Eff: if $request =$ `none` then
$request := a$
else
$violation :=$ `true`

`accel`(a):
Pre: $request = a$
Eff: $request :=$ `none`

Tasks: $BUFF = \{$`accel`$(a)\} : [\delta^-, \delta^+]$

The variable *request* stores a command while it is being buffered. The major difference between ACC-BUFFER and BUFFER of Chapter 4 is the type of the command being buffered. The variable *violation* is true when a new command from the controller arrives before the previous one has exited the buffer, that is when the buffer overflows. We use the history variable *violation* to flag this error condition.

6.2 Definition of Controller Correctness, Revisited

A valid *controller-under-feedback-and-delay* is an HIOA with no external variables and with output actions `bufAccel`(a) for $a \in [\ddot{c}_{\min} + \ddot{c}_{\text{err}}, \ddot{c}_{\max}]$ that when composed with ACC-BUFFER yeilds a correct *controller-under-feedback* as defined in Section 5.3.

In Section 4.2 we use a hiding operator ϕ in the definition of correctness for a *buffered-brake-controller*. We do not need such a hiding operator here because we are not comparing hybrid traces as one does in a simulation proof.

6.3 Parameters, Revisited

In order to guarantee that a valid controller, exists we impose the following constraints on the parameters:

1. $c_s < c_f$
2. $\dot{c}_s > \dot{c}_{maxf} \geq \dot{c}_{minf} > 0$
3. $\ddot{c}_{err} > 0$
4. $\delta_s > \delta^+ \geq \delta^- \geq 0$
5. $\ddot{c}_{min} < \ddot{c}_{min} + \ddot{c}_{err} < 0 \leq \ddot{c}_{max} - \ddot{c}_{err} < \ddot{c}_{max}$
6. $c_f - c_s \geq \frac{\dot{c}^2_{maxf} - \dot{c}^2_s}{2(\ddot{c}_{min} + \ddot{c}_{err})}$
7. $\dot{c}_{maxf} - \dot{c}_{minf} \geq -\ddot{c}_{min}(\delta_s + \delta^+)$
8. $\dot{c}_{maxf} - \dot{c}_{minf} \geq \ddot{c}_{err}(\delta^- + \delta_s) + (\ddot{c}_{max} - \ddot{c}_{min})(\delta^+ - \delta^-)$

Constraints 1, 2, 3, 5, and 6 are identical to the same numbered constraints from Section 5.4; they are restated here for convenience. Constraint 4 requires that the delay interval be well-defined and not zero and that it be shorter than the frequency of sensor feedback. Constraints 7 and 8 both ensure that the target velocity interval is wide enough. As in Chapter 5, other choices for constraints 7 and 8 are reasonable, but as stated the constraints are necessary for the correctness of the example controller of this chapter.

For convenience, we continue to assume as in Chapter 5 that the initial values of acc and $\ddot{x}$ are set to $\ddot{c}_s$, where:

$$\ddot{c}_s = \frac{\dot{c}^2_{maxf} - \dot{c}^2_s}{2(c_f - c_s)}$$

Notice that $\ddot{c}_s$ is negative.

6.4 Example Controller: DEL-ZIG-ZAG

We do not define a completely new controller for this chapter. Rather, we modify the ZIG-ZAG controller of Chapter 5. We define DEL-ZIG-ZAG to be identical to ZIG-ZAG except that we rename its output actions $\texttt{accel}(a)$ to $\texttt{bufAccel}(a)$ and redefine the tranisitions labeled with the $\texttt{status}(a, v, p)$ input actions, as follows:

$\texttt{status}(a, v, p)$:
  Eff: if $v \leq \dot{c}_{maxf}$ then
    if $\dot{c}_{maxf} < v + a(\delta_s + \delta^+)$ then
      $send := \frac{\dot{c}_{maxf} - v - a\delta^+}{\delta_s}$

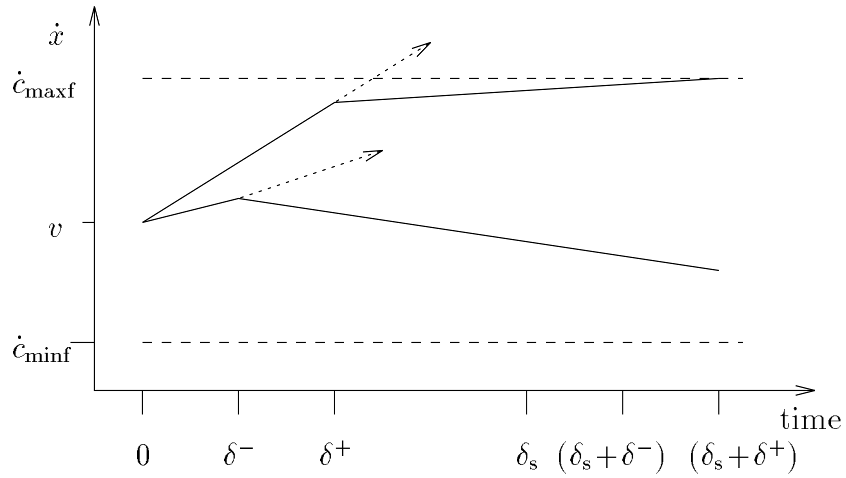
else

$send := \frac{\dot{c}_{\rm maxf} - v - a\delta^-}{\delta_s + \delta^+ - \delta^-}$

The composition of SENSOR-TRAIN, ACC-BUFFER, and DEL-ZIG-ZAG is called DEL-ZIG-ZAG-SYS.

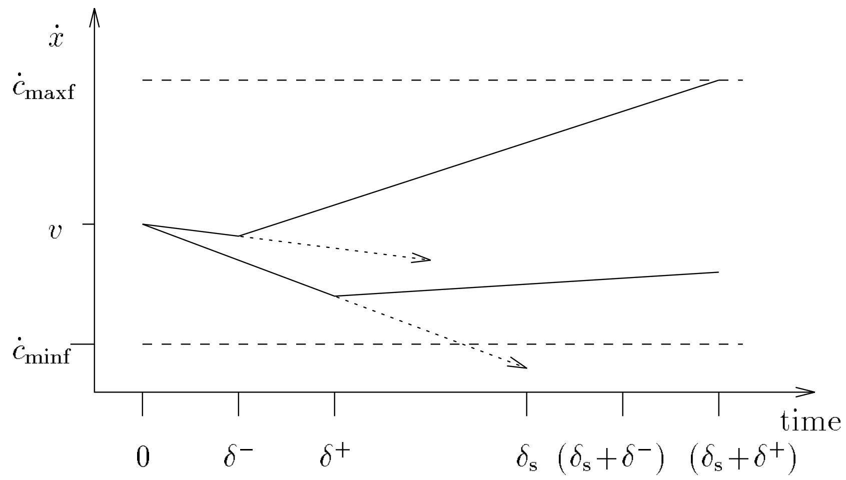
For each `status` message, DEL-ZIG-ZAG only takes action if $v \leq \dot{c}_{\rm maxf}$; this is similar to ZIG-ZAG and allows for an initial braking period at the initial (negative) acceleration $\ddot{c}_{\rm s}$. Once the velocity drops below $\dot{c}_{\rm maxf}$, the action the controller takes depends on whether an adjustment upward or downward is needed in the acceleration to keep the velocity below $\dot{c}_{\rm maxf}$. The two cases are depicted in Figure 6-2 and Figure 6-3. The figures show velocity versus time graphs of possible behaviors of DEL-ZIG-ZAG-SYS. Time zero in both figures is the time of some $\texttt{status}(a, v, p)$ message in which $v \leq \dot{c}_{\rm maxf}$. The horizontal dashed lines are the velocity bounds. The solid lines form a "bent wedge"; this wedge represents upper and lower bounds on the possible behavior of DEL-ZIG-ZAG-SYS. The origin of the wedge is at time zero when $\dot{x} = v$. The portion of the wedge before the bend bounds the evolution of $\dot{x}$ while the current acceleration is in effect. The bend in the wedge represents the change in acceleration when the buffer outputs the controller's command. The portion of the wedge after the bend bounds the evolution of $\dot{x}$ after the controllers requested acceleration takes effect. The angles of the first part of the wedge are determined by a and $a - \ddot{c}_{\rm err}$; the angles of the second part of the wedge are determined by $send$ and $send - \ddot{c}_{\rm err}$. The dotted lines represent the bounds on behavior if a remained in effect.

Figure 6-2 Adjustment downward by DEL-ZIG-ZAG.



Let us focus on Figure 6-2 first. Notice the difference between the *time* of the upper and lower bends in Figure 6-2: the lower side of the wedge bends at time δ^- and the upper side at time δ^+. This is because it is an adjustment *downward*, that is $send < a$. The upper bound on $\dot{x}$ happens when the buffer delays $send$ as long as possible; similarly, the lower bound occurs when the buffer delays *send* as little as possible. The test in the above pseudo-code "if $\dot{c}_{\mathrm{maxf}} < v + \ldots$" is true when if the current acceleration (dotted line) is allowed to remain in effect then $\dot{x}$ will exceed $\dot{c}_{\mathrm{maxf}}$ before the next guaranteed change of acceleration at time $\delta_{\mathrm{s}} + \delta^{+}$. The first branch of the "if" statement results in an adjustment downward in the acceleration as depicted in Figure 6-2. It is adjusted so that the top of the wedge is exactly $\dot{c}_{\mathrm{maxf}}$ at time $\delta_{\mathrm{s}} + \delta^{+}$. Constraints 7 and 8 on the parameters (see Section 6.3) ensure that this choice for *send* does not result in the bottom of the wedge passing below $\dot{c}_{\mathrm{minf}}$.

Figure 6-3 Adjustment upward by DEL-ZIG-ZAG.



The upward adjustment depicted in Figure 6-3 is analogous to the downward adjustment but reversed. The upper side of the wedge results from the buffer delivering the upward adjustment as soon as possible; the lower side of the wedge results from the buffer delivering the upward adjustment as late as possible. As before, the "else" branch of the "if" statement results in the top of the wedge being at exactly $\dot{c}_{\mathrm{maxf}}$ at time $\delta_{\mathrm{s}} + \delta^{+}$; however, the calculation is a bit more complex because the bend in the upper side of the wedge occurs earlier, at time δ^{-}. Once again Constraints 7 and 8 on the parameters ensure that this choice for *send* does not result in the bottom of the wedge passing below $\dot{c}_{\mathrm{minf}}$.

6.5 Correctness of DEL-ZIG-ZAG

The proof of correctness of the controller requires proofs of the timeliness and safety properties. The structure of the proofs is similar to that of Chapter 5. We first prove a "non-violation" property and then we prove each correctness property in a separate subsection.

We have presented the buffer using an MMT-specification. Since there is only one MMT task in the buffer and no other MMT-specifications to consider, we abbreviate *first*($BUFF$) and *last*($BUFF$) as *first* and *last*.

6.5.1 Non-Violation

In this section we prove that the history variable *violation* remains `false` in all executions of DEL-ZIG-ZAG-SYS. It follows as a corollary of the following lemma.

As in the proof of non-violation in Section refsec:DelayVio, it is sufficient to prove the invariant that either *send* or *request* is always `none`. As before we must strengthen this invariant so that it may be proved by induction. The lemma proves this stronger form that is the conjunction of two implications. Informally, it uses deadline variables to say that (1) DEL-ZIG-ZAG only sends commands immediately after `status` messages and (2) ACC-BUFFER will relay requested commands before the next `status` message. It depends primarily on constraint 4 on the parameters: $\delta_s > \delta^+$.

Lemma 6.5.1 *In all reachable states of* DEL-ZIG-ZAG-SYS *the following hold:*

1. $send \neq \texttt{none} \Longrightarrow (next = now + \delta_s) \wedge request = \texttt{none}$
2. $request \neq \texttt{none} \Longrightarrow (last + \delta_s = next + \delta^+) \wedge send = \texttt{none}$

Proof: Proof by induction. The property to be proved consists of the conjunction of two implications; we call them Rule 1 and Rule 2 in the style of the proof of Lemma 5.6.2. Note that only one of the Rules can apply and hold in a given state. Basis: in the initial state neither rule applies. Induction: Let state s lead to state s' via a single step — either a discrete step labeled by action π or a trajectory. We proceed by cases on the type of step and π: `accel`, `bufAccel`, `status`, or trajectory.

1. $\pi = \texttt{accel}$: Rule 2 applies and holds in s. The transition sets $request' = \texttt{none}$ and does not affect *send*. Therefore, neither rule applies in the s'.

2. $\pi = \texttt{bufAccel}$: Rule 1 applies and holds in s'. The transition sets $request' \neq \texttt{none}$ and $send' = \texttt{none}$. It does not affect *now* or *next* and it sets $last' = now + \delta^+$. By the inductive hypothesis, $next = next' = now + \delta_s$, so Rule 2 applies and holds in s'.

3. $\pi = \texttt{status}$: We claim that neither rule applies in s. The precondition for this action is $now = next$; so clearly Rule 1 cannot apply in s. For the purpose of contradiction suppose Rule 2 applied in s, then $last + \delta_s = next + \delta^+$. However, by assumption on the parameters $\delta_s > \delta^+$, so $last < next$ and therefore $last < next = now$. But this contradicts Theorem 2.8.1. Thus neither rule applies in s, i.e. $send = \texttt{none}$ and $request = \texttt{none}$. The transition does not affect *request* so $request' = \texttt{none}$ and it sets $next' = now' + \delta_s$. Thus Rule 1 holds in s' (whether or not it applies).

4. The step is a trajectory: does not affect any of the mentioned variables except *now*. The *now* variable only appears in Rule 1 and that rule only applies when time passage is forbidden.

These cases are exhaustive and thus the property holds. ■

The non-violation property for DEL-ZIG-ZAG-SYS is established in the following corollary.

Corollary 6.5.2 *In all reachable states of* DEL-ZIG-ZAG-SYS *violation* $=$ `false`.

Proof: Violation occurs when *request* $\neq$ `none` and a `bufAccel` action takes place. This action is only enabled when *send* $\neq$ `none`; however, by Lemma 6.5.1 *request* $=$ `none` in that case. Therefore the property holds. ■

6.5.2 Timeliness

The structure of the proof is similar to that in Chapter 5. As in that chapter, the major result we require is an invariant that implies the lower bound invariant on velocity. The following lemma establishes such a result by strengthening the lower bound on velocity. It is analogous to Lemma 5.6.2; it is more complex because of extra cases and the uncertainty introduced by the buffer. We have changed the notation slightly to accommodate the more complex formulas. The invariant consists of four clauses: 1, 2a, 2bi, and 2bii. We explain their informal meaning in terms of the wedges of Figures 6-2 and 6-3. Each clause tests that at a certain point in the execution, the lower arm of the wedge remains above $\dot{c}_{\mathrm{minf}}$. Clause 1 applies when the controller has chosen a command (stored in *send*) but has not yet passed it to the buffer. Clause 2a applies when neither the controller nor the buffer are holding an unsent command. Clause 2bi applies when the buffer holds a command which has *not* been held long enough to relay. Clause 2bii applies when the buffer holds a command which has been held long enough to relay.

Lemma 6.5.3 *Let $\overline{z}$ denote $z - \ddot{c}_{\mathrm{err}}$. In all reachable states of* DEL-ZIG-ZAG-SYS *the following hold:*

1. $send \neq \texttt{none} \Longrightarrow request = \texttt{none} \wedge \dot{x} \geq \dot{c}_{\mathrm{minf}} \wedge$
   $\dot{x} + \overline{acc}(\delta^-) + \min(\overline{acc}, \overline{send})(\delta^+ - \delta^-) + \overline{send}(\delta_{\mathrm{s}}) \geq \dot{c}_{\mathrm{minf}}$

2. $send = \texttt{none} \Longrightarrow$

   (a) $request = \texttt{none} \Longrightarrow \dot{x} \geq \dot{c}_{\mathrm{minf}} \wedge \dot{x} + \overline{acc}(next - now + \delta^+) \geq \dot{c}_{\mathrm{minf}}$

   (b) $request \neq \texttt{none} \Longrightarrow$

i. $now < first \Longrightarrow \dot{x} \geq \dot{c}_{\mathrm{minf}} \wedge \dot{x} + \overline{acc}(first - now)$
$+ \min(\overline{acc}, \overline{request})(\delta^+ - \delta^-) + \overline{request}(\delta_{\mathrm{s}}) \geq \dot{c}_{\mathrm{minf}}$

ii. $now \geq first \Longrightarrow \dot{x} \geq \dot{c}_{\mathrm{minf}} \wedge$
$\dot{x} + \min(\overline{acc}, \overline{request})(last - now) + \overline{request}(\delta_{\mathrm{s}}) \geq \dot{c}_{\mathrm{minf}}$

Proof: Proof by induction. As in the proofs of similar lemmas from the previous chapter we refer to the parts of the above invariant as "rules". Basis case: In the initial state Rule 2a applies. We show that it holds as follows: Note that $\overline{acc} = \overline{\ddot{c}_{\mathrm{s}}}$ and $next - now = 0$ and $\dot{x} = \dot{c}_{\mathrm{s}} > \dot{c}_{\mathrm{maxf}} \geq \dot{c}_{\mathrm{minf}}$. Thus, it is sufficient to show that $\dot{c}_{\mathrm{maxf}} + \overline{\ddot{c}_{\mathrm{s}}}\delta^+ \geq \dot{c}_{\mathrm{minf}}$. This follows from the fact that $\overline{\ddot{c}_{\mathrm{s}}} \geq \ddot{c}_{\mathrm{min}}$ and parameter constraint 7. Inductive case: Suppose the property is true in state s; we must show that it is true in s' which follows from s in one step — either a discrete transition labeled by π or a trajectory. For the sake of brevity, we denote variables in the post-state by adding primes, e.g. we write now' instead of $s'.now$. We brake by cases on the type of step and on π: `accel`, `bufAccel`, `status`, or trajectory.

1. $\pi =$ `accel`: We know $request \neq$ `none`, so by Lemma 6.5.1 $send =$ `none`. Furthermore, $now \geq first$, by this actions precondition, so Rule 2bii applies is s and holds by the inductive hypothesis. As for the post-state — $request' =$ `none` and $send' =$ `none`, so Rule 2a applies in s'. We show that it holds by noting that $\overline{acc'} = \overline{request}$ and no other relevant variables have changed. Substitution and Lemma 6.5.1 yield the desired result, as follows:

$$\begin{array}{rcll}
last - now & \geq & 0 & \text{by Theorem 2.8.1} \\
\overline{req} & \geq & \min(\overline{acc}, \overline{req}) & \text{definition of min} \\
\dot{x} + \min(\overline{acc}, \overline{req})(last - now) + \overline{req}\delta_{\mathrm{s}} & \geq & \dot{c}_{\mathrm{minf}} & \text{inductive hypothesis} \\
\dot{x} + \overline{req}(last - now) + \overline{req}\delta_{\mathrm{s}} & \geq & \dot{c}_{\mathrm{minf}} & \text{substitute} \\
\dot{x} + \overline{req}(last - now + \delta_{\mathrm{s}}) & \geq & \dot{c}_{\mathrm{minf}} & \text{group} \\
last + \delta_{\mathrm{s}} & = & next + \delta^+ & \text{by 6.5.1} \\
\dot{x} + \overline{req}(next - now + \delta^+) & \geq & \dot{c}_{\mathrm{minf}} & \text{substitute} \\
\overline{acc'} & = & \overline{req} & \text{automaton definition} \\
\dot{x}' + \overline{acc'}(next' - now' + \delta^+) & \geq & \dot{c}_{\mathrm{minf}} & \text{substitute}
\end{array}$$

2. $\pi =$ `bufAccel`: We know $send \neq$ `none` so Rule 1 applies in s. It holds by the inductive hypothesis. Also, $request' \neq$ `none`, $send' =$ `none`, and $now' < first'$, so Rule 2bi applies in s'. We must show that it holds. This is trivial because $request' = send$ and $first = now + \delta^-$.

3. $\pi =$ `status`: As in the same case in the proof of Lemma 6.5.1, we know that $send =$ `none` and $request =$ `none`; thus, Rule 2a applies in s and it holds by the inductive hypothesis. We break by cases:

   (a) $send' =$ `none`, so Rule 2a applies in s'. It holds because none of the variables in its consequent are affected by the transition.

(b) $send' \neq$ `none`, so Rule 1 applies in s'. Note that $a = acc$, so we write acc instead; similarly for v and $\dot{x}$. Also note that $now = next$ by the actions precondition, and $\dot{x} \leq \dot{c}_{\mathrm{maxf}}$ by the actions effect. Finally, note that $send$ and $next$ are the only variables modified on this transition. We break by cases on the branch of the conditional taken in the effect clause in DEL-ZIG-ZAG.

i. $\dot{c}_{\mathrm{maxf}} < \dot{x} + acc(\delta_{\mathrm{s}} + \delta^{+})$ — In this case, we first resolve the "min" operator by showing that $send' < acc$. As follows:

$$\begin{array}{rcll}
send' & = & \frac{\dot{c}_{\mathrm{maxf}} - \dot{x} - (acc)\delta^{+}}{\delta_{\mathrm{s}}} & \text{automaton definition} \\
\dot{x} + (acc)\delta^{+} + send'\delta_{\mathrm{s}} & = & \dot{c}_{\mathrm{maxf}} & \text{simplify} \\
\dot{c}_{\mathrm{maxf}} & < & \dot{x} + acc(\delta_{\mathrm{s}} + \delta^{+}) & \text{case} \\
\dot{x} + (acc)\delta^{+} + send'\delta_{\mathrm{s}} & < & \dot{x} + acc(\delta_{\mathrm{s}} + \delta^{+}) & \text{substitute} \\
send'\delta_{\mathrm{s}} & < & acc\delta_{\mathrm{s}} & \text{cancel} \\
\delta_{\mathrm{s}} & \geq & 0 & \text{parameter assumption} \\
send' & < & acc & \text{divide}
\end{array}$$

Now we must show that $\dot{x} + \overline{acc}\delta^{-} + \overline{send'}(\delta^{+} - \delta^{-} + \delta_{\mathrm{s}}) \geq \dot{c}_{\mathrm{minf}}$. First, notice that $send' < acc$ implies that $0 \leq acc - \overline{send'}$. Also, acc is bounded above by $\ddot{c}_{\mathrm{max}}$ and $\overline{send'}$ below by $\ddot{c}_{\mathrm{min}}$. This justifies the first inequality that appears below:

$$\begin{array}{rcll}
\ddot{c}_{\mathrm{max}} - \ddot{c}_{\mathrm{min}} \geq acc - \overline{send'} & \geq & 0 & \text{above} \\
\dot{c}_{\mathrm{maxf}} - \ddot{c}_{\mathrm{err}}(\delta^{-} + \delta_{\mathrm{s}}) & & & \\
-(\ddot{c}_{\mathrm{max}} - \ddot{c}_{\mathrm{min}})(\delta^{+} - \delta^{-}) & \geq & \dot{c}_{\mathrm{minf}} & \text{parameter assumption} \\
\delta^{+} \geq \delta^{-} & \geq & 0 & \text{parameter assumption} \\
\dot{c}_{\mathrm{maxf}} - \ddot{c}_{\mathrm{err}}(\delta^{-} + \delta_{\mathrm{s}}) & & & \\
-(acc - \overline{send'})(\delta^{+} - \delta^{-}) & \geq & \dot{c}_{\mathrm{minf}} & \text{substitute} \\
\dot{x} + (acc)\delta^{+} + send'\delta_{\mathrm{s}} & = & \dot{c}_{\mathrm{maxf}} & \text{as above} \\
\dot{x} + (acc)\delta^{+} + send'\delta_{\mathrm{s}} - \ddot{c}_{\mathrm{err}}(\delta^{-} + \delta_{\mathrm{s}}) & & & \\
-(acc - \overline{send'})(\delta^{+} - \delta^{-}) & \geq & \dot{c}_{\mathrm{minf}} & \text{substitute} \\
\dot{x} + \overline{acc}\delta^{-} + \overline{send'}(\delta^{+} - \delta^{-} + \delta_{\mathrm{s}}) & \geq & \dot{c}_{\mathrm{minf}} & \text{simplify}
\end{array}$$

ii. $\dot{c}_{\mathrm{maxf}} \geq \dot{x} + acc(\delta_{\mathrm{s}} + \delta^{+})$ — As in the previous case, we first resolve the "min" operator by showing that $send' \geq acc$. As follows:

$$
\begin{array}{rcll}
\mathit{send}' & = & \frac{\dot{c}_{\mathrm{maxf}} - \dot{x} - (acc)\delta^-}{\delta_s + \delta^+ - \delta^-} & \text{automaton definition} \\
\dot{x} + (acc)\delta^- & & & \\
+\mathit{send}'(\delta_s + \delta^+ - \delta^-) & = & \dot{c}_{\mathrm{maxf}} & \text{simplify} \\
\dot{c}_{\mathrm{maxf}} & \geq & \dot{x} + acc(\delta_s + \delta^+) & \text{case} \\
\dot{x} + (acc)\delta^- & & & \\
+\mathit{send}'(\delta_s + \delta^+ - \delta^-) & < & \dot{x} + acc(\delta_s + \delta^+) & \text{substitute} \\
\mathit{send}'(\delta_s + \delta^+ - \delta^-) & < & acc(\delta_s + \delta^+ - \delta^-) & \text{cancel} \\
(\delta_s + \delta^+ - \delta^-) & \geq & 0 & \text{parameter assumption} \\
\mathit{send}' & < & acc & \text{divide}
\end{array}
$$

Now we must show that $\dot{x} + \overline{acc}\delta^+ + \overline{\mathit{send}'}\delta_s \geq \dot{c}_{\mathrm{minf}}$. By similar reasoning to that used in the analogous case above we get the first inequality:

$$
\begin{array}{rcll}
\ddot{c}_{\max} - \ddot{c}_{\min} \geq \mathit{send}' - \overline{acc} & \geq & 0 & \text{above} \\
\dot{c}_{\mathrm{maxf}} - \ddot{c}_{\mathrm{err}}(\delta^- + \delta_s) & & & \\
-(\ddot{c}_{\max} - \ddot{c}_{\min})(\delta^+ - \delta^-) & \geq & \dot{c}_{\mathrm{minf}} & \text{parameter assumption} \\
\delta^+ \geq \delta^- & \geq & 0 & \text{parameter assumption} \\
\dot{c}_{\mathrm{maxf}} - \ddot{c}_{\mathrm{err}}(\delta^- + \delta_s) & & & \\
-(\mathit{send}' - \overline{acc})(\delta^+ - \delta^-) & \geq & \dot{c}_{\mathrm{minf}} & \text{substitute} \\
\dot{x} + (acc)\delta^- + \mathit{send}'(\delta_s + \delta^+ - \delta^-) & = & \dot{c}_{\mathrm{maxf}} & \text{as above} \\
\dot{x} + (acc)\delta^- + \mathit{send}'(\delta_s + \delta^+ - \delta^-) & & & \\
-\ddot{c}_{\mathrm{err}}(\delta^- + \delta_s) - (\mathit{send}' - \overline{acc})(\delta^+ - \delta^-) & \geq & \dot{c}_{\mathrm{minf}} & \text{substitute} \\
\dot{x} + \overline{acc}\delta^+ + \overline{\mathit{send}'}\delta_s & \geq & \dot{c}_{\mathrm{minf}} & \text{simplify}
\end{array}
$$

4. The step is a trajectory: We know that $\mathit{send} = \mathit{send}' = \texttt{none}$ so Rule 2 applies in s and s'. This case is straightforward. It uses a similar argument to that of the trajectory case in the proof of Lemma 5.6.2. We outline the subcases that must be considered but give no details of their proofs:

   (a) $\mathit{request} = \mathit{request}' = \texttt{none}$, so Rule 2a applies in s and s'.

   (b) $\mathit{request} = \mathit{request}' \neq \texttt{none}$, so Rule 2b applies in s and s'.

      i. $\mathit{now} < \mathit{first}$, so Rule 2bi applies in s. We proceed by cases:

         A. $\mathit{now}' < \mathit{first}'$, so Rule 2bi applies in s'.

         B. $\mathit{now}' \geq \mathit{first}'$, so Rule 2bii applies in s'.

      ii. $\mathit{now} \geq \mathit{first}$, so Rule 2bii applies in s and s'.

■

The following corollary establishes the lower bound on velocity as an invariant of DEL-ZIG-ZAG-SYS.

Corollary 6.5.4 *In all reachable state of* DEL-ZIG-ZAG-SYS *the following holds:*

$\dot{x} \geq \dot{c}_{\mathrm{minf}}$

Proof: Directly from 6.5.3. The antecedents form an exhaustive set of cases, and in all cases the property is true. ∎

Corollary 6.5.4 leads to the timeliness property just as Lemma 3.6.6 did in Chapter 3. The corollaries that yield the timeliness property are exactly analogous and are not restated here. The final result is summarized in Theorem 6.5.6 at the end of this chapter.

6.5.3 Safety

In this section, we give only the major result, Lemma 6.5.5; it leads to the safety property for DEL-ZIG-ZAG just as Lemma 5.6.5 for ZIG-ZAG. We do not give the intermediate corollaries and lemmas that yield the safety property because they are precisely analogous to those of Section 5.6.2.

Lemma 6.5.5 is similar to both Lemma 5.6.5 and Lemma 6.5.3. It is a strengthening of the desired invariant and its form is the conjunction of a set of implications. The form of the first clause borrows from the first clause of Lemma 5.6.5. The form of the remaining clauses is analogous to Lemma 6.5.3; however, these clauses check that the upper arm of the wedge is lower than $\dot{c}_{\mathrm{maxf}}$ whereas the analogous clauses in Lemma 6.5.3 check the lower arm of the wedge against $\dot{c}_{\mathrm{minf}}$.

Lemma 6.5.5 *In all reachable states of* DEL-ZIG-ZAG-SYS *the following hold:*

1. $\dot{x} > \dot{c}_{\mathrm{maxf}} \Longrightarrow acc = \ddot{c}_{\mathrm{s}} \wedge send = \mathtt{none} \wedge \left((x \leq c_{\mathrm{f}}) \Longrightarrow c_{\mathrm{f}} - x \geq \frac{\dot{c}^2_{\mathrm{maxf}} - \dot{x}^2}{2\ddot{c}_{\mathrm{s}}}\right)$
2. $\dot{x} \leq \dot{c}_{\mathrm{maxf}} \Longrightarrow$
   (a) $send \neq \mathtt{none} \Longrightarrow request = \mathtt{none} \wedge$
   $\dot{x} + acc(\delta^-) + \max(acc, send)(\delta^+ - \delta^-) + send(\delta_{\mathrm{s}}) \leq \dot{c}_{\mathrm{maxf}}$
   (b) $send = \mathtt{none} \Longrightarrow$
      i. $request = \mathtt{none} \Longrightarrow \dot{x} + acc(next - now + \delta^+) \leq \dot{c}_{\mathrm{maxf}}$
      ii. $request \neq \mathtt{none} \Longrightarrow$
         A. $now < first \Longrightarrow$
         $\dot{x} + acc(first - now) + \max(acc, request)(\delta^+ - \delta^-) + request(\delta_{\mathrm{s}}) \leq \dot{c}_{\mathrm{maxf}}$
         B. $now \geq first \Longrightarrow$
         $\dot{x} + \max(acc, request)(last - now) + request(\delta_{\mathrm{s}}) \leq \dot{c}_{\mathrm{maxf}}$

Proof: The invariant in this lemma is very similar to that of Lemma 6.5.3 and so is its proof. ∎

We summarize the correctness results in the following theorem.

Theorem 6.5.6 *Automaton* DEL-ZIG-ZAG *is a correct controller-under-feedback-and-delay.*

Proof: We must show that the composition of DEL-ZIG-ZAG and ACC-BUFFER is a correct controller-under-feedback as defined in Section 5.3. This in turn requires that the hybrid traces of DEL-ZIG-ZAG-SYS satisfy the timeliness and safety properties of Section 3.4. As mentioned at the end of Section 6.5.2, the timeliness property follows from Corollary 6.5.4 just as it did from Lemma 3.6.6 in Chapter 3. We have omitted the intermediate results. Similarly, the safety property follows from Lemma 6.5.5 as it did from Lemma 5.6.5 in Chapter 5. We have omitted the intermediate results. ■

Chapter 7

Conclusion

Summary

We have presented a case study in the application of hybrid I/O automaton techniques to automated transit systems. The purpose of the case study is to test the applicability of HIOA techniques to the area of automated transit; in particular, we are concerned that HIOA techniques express hybrid systems faithfully and that they allow clear and scalable proofs of significant properties of these systems.

We focused on the deceleration maneuver in which a train's controller slows the train to a target velocity range within a given distance. We examined four versions of the deceleration maneuver, each with a different model of the communication between controller and train: plain, delay, feedback, and feedback with delay. In the plain case of Chapter 3, the controller receives no sensor information from the train and controls the brake through on and off commands which take effect immediately. The delay case of Chapter 4 is like the plain case except that the brake commands are delayed. In the feedback case of Chapter 5, the controller receives periodic sensor information from the train; the controller can instantly command the train to achieve specific positive and negative accelerations subject to some performance error. The feedback with delay case of Chapter 6 is like the feedback case except that the acceleration commands are delayed. For each case we give a model of the non-controller portion of the system, define correctness of a controller, give an example of a correct controller, and prove that it is correct.

We model the train and the controller as HIOAs communicating through discrete actions. For the cases with delay, we interpose a third automaton which serves as a buffer, delaying messages from the controller to the train. The buffers and some of the example controllers are defined using the MMT-specifications of Section 2.8. The other automata are defined using the standard notation of Section 2.7.

The main correctness conditions for controllers are the timeliness and safety properties, defined in Section 3.4. The timeliness property says that the train always progresses to the destination location within a fixed time. The safety property says that when the train arrives at the destination it has achieved a velocity in the target range. These properties mention only the variables of the train. Since the train outputs these variables, we cast these properties as hybrid trace properties of the composition of the train and the controller (and a buffer if applicable).

We use two major proof methods: invariant assertions and simulations. The use of invariant assertions is ubiquitous in this case study. The use of invariant assertions usually involves strengthening a proposed invariant assertion until it can be proved by induction on the steps of a hybrid execution. These inductive proofs have a stylized form that separates reasoning about discrete behavior (actions) from continuous behavior (trajectories). Timing information such as the current time and deadlines for events are explicitly modeled in the state as variables (e.g. now, $last(OFF)$). These variables facilitate proofs of timing behavior using invariant assertions. MMT-specifications implicitly add many such timing variables in a standard manner which makes the automata definitions and related proofs more readable.

We use one simulation in this case study: in Chapter 4 a simulation shows that the composition of the buffer and controller of that chapter is an implementation of the controller of Chapter 3. Using the subtitutivity result of Theorem 2.6.2, the timeliness and safety properties follow because they are preserved by hybrid trace inclusion.

This case study contains full proofs of the correctness of the various controllers. However, some of the proofs are only sketched, when similar formal proofs appear in other chapters.

Evaluation

The hybrid I/O automaton model and its related tools provide a framework in which a modest hybrid system can be described naturally and verified formally. Trajectories appear essential to a faithful treatment of physical systems. They permit differential relationships between physical variables to be expressed directly. We also found shared variables useful. If the variables of a system are exposed then some properties can be expressed as hybrid trace properties. This allows certain properties like the timeliness and safety properties to be cast as hybrid trace properties which in the timed I/O automaton model would necessarily have been properties of timed executions.

The proofs in this case study are clear and scalable from the plain case to the feedback with delay case. We believe clarity and scalability are the result of our reliance on invariant assertions throughout. This technique enhances clarity because invariant assertions have a close relationship to intuitive, informal claims. The proofs of invariant assertions are usually by induction in a stylized manner which allows for easy navigation and checking. The assertional technique is scalable to more complex systems because often the invariant itself holds on the more complex system. Even if it does not, often the invariant of the simple system appears embedded in an invariant of the more complex system. For example, the invariant in Lemma 3.6.10 appears in clause 1 of the invariant in Lemma 5.6.5. When substitution like this occurs the proof of the original invariant can often be reused with minor modification. For example, compare the proofs of Lemmas 3.6.10 and 5.6.4. We believe this kind of reuse is characteristic of invariant assertion based methods. There remains the challenge of finding invariants that maximize reuse.

We have a more guarded evaluation of simulations because of their more limited use in this case study. The simulation proof in Chapter 4 is clear and concise. However, we acknowledge that its use is limited in two respects. First, it involves only the computer portion of the system. As a result, the components and the simulation itself could have been expressed using timed I/O automaton methods. Our contribution is in showing how this well understood method of proof for computer systems can be woven into the treatment of a hybrid system.

Second, we acknowledge that the case study does not demonstrate that simulations scale from the delay case to the feedback with delay case. As mentioned in Chapter 6, no simulation is possible from a controller for the feedback with delay case to ZIG-ZAG, the example controller of Chapter 5. Because ZIG-ZAG always responds instantly to its sensor input, no controller with delayed responses can implement it. This begs the question of whether a simulation based proof in the feedback with delay case is possible given some other choice of controller for the feedback case. The answer is yes. However, we chose not to present such a controller because it would be overly complex without illustrating any new techniques or insights. The complexity of such a controller arises from its need to be highly non-deterministic both in *when* it sends multiple acceleration commands and *which* acceleration command it sends. This differs from the simple non-determinism of ONE-SHOT of Chapter 3 that merely varies the timing of two brake commands and not their content.

Further Work

This case study took shape during the early stages of the development of the HIOA model and does not exercise all the model's features. In particular, further case studies involving HIOA's could investigate more fully the use of shared variables. In this work we modeled the physical part of the system, the train, as a single automaton. We believe that the shared variables of HIOAs are the key to a more modular treatment of physical systems. Some modest progress in this direction appears in [15] where sensors and actuators are modeled as separate automata which share variables with the physical system. Nevertheless, we anticipate further progress in using this facet of the HIOA model.

We look forward to further examination of the utility of simulation proofs for hybrid systems. An effort toward this begins in [14] but much remains to be done. We chose to avoid a highly abstract example controller in Chapter 5 because for this example the increased non-determinism would lead to complexity that would obscure the description. The utility of simulation proofs depends on the lucidity of more abstract specifications; we hope that our experience in this case study is the exception rather than the rule for hybrid systems.

Much work remains for the M.I.T. Theory of Distributed Systems research group in our long-term project applying these techniques to automated transit systems. Current research involves further case studies in ground based transportation systems. We are modeling multi-vehicle maneuvers arising in the California PATH project [8, 9, 10]. The high-level and preliminary treatment of safety systems in [15] will be extended to examine the implementations of those systems in the Raytheon Personal Rapid Transit project. We hope to develop a machine parsable language for hybrid system specifications and to develop tools for computer aided proof checking and verification. We are examining methods for integrating into our methods the techniques of relevant disciplines such as mechanical engineering and control theory. Our long term goal is to help design the industrial strength formal tools that will have an impact on the design and development of real transportation systems.

Bibliography

[1] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In *DIMACS Workshop on Verification and Control of Hybrid Systems*, October 1995. To appear in R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science, Springer-Verlag. Also, to appear as MIT/LCS/TM-544.

[2] Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part II: Timing-based systems. Technical Memo MIT/LCS/TM-487.c, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, April 1995.

[3] Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part II: Timing-based systems. *Information and Computation*. To appear. Available now as [2].

[4] R. Gawlick, R. Segala, J. Søgaard-Andersen, and N. Lynch. Liveness in timed and untimed systems. Technical Report MIT/LCS/TR-587, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 02139, December 1993.

[5] Rainer Gawlick, Roberto Segala, Jørgen Søgaard-Andersen, and Nancy Lynch. Liveness in timed and untimed systems. In Serge Abiteboul and Eli Shamir, editors, *Proceedings of the 21st International Colloquim, ICALP94*, volume 820 of *Lecture Notes in Computer Science*, pages 166–177, Jerusalem, Israel, July 1994. Springer-Verlag. Full version in [4].

[6] 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.

[7] 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.

[8] Datta N. Godbole, John Lygeros, and Shankar Sastry. Hierarchical hybrid control: A case study. Prelminary report for the california path program, Institute of Transportations Studies, University of California, August 1994.

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

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

[11] Nancy Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. In *DIMACS Workshop on Verification and Control of Hybrid Systems*, October 1995. To appear in R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science, Springer-Verlag. Also, to appear as MIT/LCS/TM-545.

[12] Constance Heitmeyer and Nancy Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. In *Proceedings of the IEEE Real-Time Systems Symposium.*, pages 120–131, San Juan, Puerto Rico, December 1994. IEEE Computer Society Press.

[13] Constance Heitmeyer and Nancy Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. Technical Memo MIT/LCS/TM-511, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, November 1994.

[14] Nancy Lynch. A three-level analysis of a simple acceleration maneuver, with uncertainties. Manuscript. WWW URL=http://theory.lcs.mit.edu/three-level.html.

[15] H.B. Weinberg, Nancy Lynch, and Norman Delisle. Verification of automated vehicle protection systems. In *DIMACS Workshop on Verification and Control of Hybrid Systems*, October 1995. To appear in R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science, Springer-Verlag.

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

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

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

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

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

[21] Victor Luchangco. Using simulation techniques to prove timing properties. Master's thesis, MIT Electrical Engineering and Computer Science, 1995. In progress.

[22] Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part I: Untimed systems. *Info. Comput.*, to appear.

[23] Keith Marzullo, Fred B. Schneider, and Navin Budhiraja. Derivation of sequential real-time, process control programs. In Andre M. van Tilborg and Gary M. Koob, editors, *Foundations of Real-Time Computing*, pages 39–54. Kluwer Academic Publishers, 1991.

[24] Gunter Leeb and Nancy Lynch. Proving safety properties of the steam boiler controller: Formal methods for industrial applications, a case study, January 1996. Submitted for publication. Presented at the *Methods for Semantics and Specification*, International Conference and Research Center for Computer Science, Schloss, Dagstuhl, Germany, June 1995, as "Using Timed Automata for the Steam Boiler Controller Problem.".

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

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

[27] Nancy A. Lynch and Hagit Attiya. Using mappings to prove timing properties. *Distributed Computing*, 6(2):121–139, 1992.

[28] N. Lynch and M. Tuttle. An introduction to Input/Output automata. *CWI-Quarterly*, 2(3):219–246, September 1989. Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands.

[29] N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *Proceedings of the* 6^{th} *Annual ACM Symposium on Principles of Distributed Computing*, pages 137–151, August 1987.

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