Second European Workshop on Real-time and Hybrid Systems., Grenoble, France June 1995

Proving Correctness of a Vehicle Maneuver: Deceleration

Nancy Lynch * H.B. Weinberg †

March 29, 1995

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.

Among the formal tools that have been developed are the *timed I/O automaton* model of Lynch and Vaandrager. Timed I/O automata are (possibly infinite state) labeled transition systems that can be composed by synchronizing on common actions. Timed I/O automata support proof methods based on invariant assertions and simulations [1, 2], as well as compositional reasoning [3]. Invariants and simulations may include statements about time, including deadlines for future events. These tools have previously been used for verifying several timing-based algorithms [4, 5, 6], and for a small case study on railroad control [7, 8].

In this paper, we describe our recent application of timed I/O automata, invariants and simulations, to reasoning about problems arising in automated transportation systems. This work has led to new development of the formal tools as well as insights about the application systems.

Typical examples of automated transportation systems include the Raytheon Personal Rapid Transit System and the California PATH project [9, 10, 11]. In these 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 [9] 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. In this paper we focus on a single maneuver: the task of decelerating a vehicle to a target speed within a certain distance. A scenario in which such a maneuver would be used is when a vehicle is approaching an area whose maximum allowable velocity is lower than the vehicle's current velocity.

Previous applications of timed I/O automaton methods recommend them for several reasons. First, invariant assertions are usually proved by induction on the length of an execution, in a stylized form that makes them easy to write, check and understand. Second, systems can be described using levels of abstraction, where each level is a separate timed automaton. The levels of the hierarchy are related to one another using a mapping called a simulation, which formally expresses the abstraction relation. Assertions proved on the high level models extend to the lower level models via the simulation mapping. This is

*Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139. Research supported by ARPA grant N00014-92-J-4033, NSF grant 922124-CCR, and ONR-AFOSR grant F49620-94-1-0199.

†Research supported by an NSF Graduate Fellowship

useful because assertions are usually easier to prove on the more abstract models. Third, system models and reasoning methods can be parameterized, allowing reasoning in terms of abstract constraints among the parameters. Fourth 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 paper, the insight gained through this manual process is particularly useful.

Our work required us to extend the timed I/O automaton model slightly, adding some extra flexibility in the types of analog behavior that can be expressed. We call the resulting extension the *hybrid I/O automaton* model. We model our hybrid systems as compositions of hybrid I/O automata, with separate automata for the discrete and analog components. The physical equations that describe the vehicle involve position, velocity, and acceleration. Hybrid I/O automata are able to capture the integration (or differentiation) relation between these physical variables.

This paper develops four cases of the deceleration maneuver. The first case is the simplest. The second case introduces a communication delay between the controller and the vehicle. The third case introduces feedback; 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. Most of the proofs are proofs by induction of an invariant assertion. The proof of the second case, delay, employs a simulation mapping.

We describe related work. 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 Ed Anderson of the Taxi2000 Corp. Prof. Shankar Sastry and his colleagues at Berkeley have studied intelligent highway systems [9, 10, 11] 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 [10]. Another project involving formal modeling of advanced vehicle control systems, using some computer science techniques, was carried out by Schneider and co-workers [12]. Their emphasis was on the use of a particular methodology to *derive* correct solutions.

The development of models and verification methods for timing-based systems is an active research area within computer science. Our timed I/O automaton model is similar, for example, to a model of Alur and Dill [13], to one of Lamport [14] and to one of Henzinger, Manna and Pnueli [15]. Some have extended their models to treat hybrid systems, for example, Manna and Pnueli [16].

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 [17, 18]. Typical work on temporal logic appears in [19]. Typical algorithmic solutions to problems of fault-tolerance, communication, synchronization, distributed agreement, resource allocation, etc., are presented in [20].

2 Model

In this section, we present an informal description of the *hybrid I/O automaton* model. Formal definitions and statements of major theorems of the model appear in Appendix A. The *hybrid I/O automaton* model presented here is an extension of the *timed I/O automaton* model. We assume the reader is familiar with timed I/O automata.

A hybrid I/O automaton is an infinite state machine with two types of transitions: *discrete steps* and *time passage steps*. Discrete steps capture discrete state transition and are no different from those of timed I/O automata. The discrete steps of the automaton are labeled with actions and the actions are partitioned into input, output, and internal actions. When automata are composed, they synchronize on input/output actions. On the other hand, time passage steps, also called *trajectories*, capture continuous evolution through a set of states. A trajectory is a mapping $w : I \rightarrow states$, where I is a left-closed interval of $\mathbb{R}^{\geq 0}$ with left endpoint equal to zero. There are no further restrictions placed on an *individual* trajectory. This is the main difference between the hybrid and timed models: trajectories are primitive in the hybrid model. The only restriction placed on trajectories is that the set of trajectories for an automaton is closed under countable "concatenation" and "splitting". We define these terms formally in the appendix. We state without proof that these properties hold for the trajectory sets of all automata defined in this paper.

A number of definitions and results for timed I/O automata have been extended to hybrid I/O automata, including: executions, traces, admissibility, sampling, simulations, and composition.

We use two notations for specifying hybrid I/O automata: standard and MMT-style. The standard notation corresponds closely to the formal definition of the automata. The states of the automaton are specified as the possible assignments to a set of variables. The discrete transitions of the automaton are specified in the *precondition-effect* style of [21, 22]. The trajectories are specified directly. We discuss standard notation in the context of an example in Section 3.2. The MMT-style notation is a convenient shorthand for a useful subclass of hybrid I/O automata. Converting an MMT-style description to a standard description is a purely syntactic transformation. This transformation and related theorems are described formally in Appendix B. The name "MMT" derives from the names of the authors of [23] where they present a model which corresponds to this subclass. We will discuss MMT-style notation in more detail when it is first used in Section 3.5.

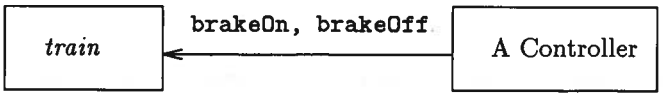

Figure 1: Overview of Basic Deceleration Model

3 Basic Deceleration

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

In the following subsections we describe the parameters of the specification, give a hybrid I/O automaton 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 (none), velocity (one), or acceleration (two). The subscript identifies the particular constant. Initial values of the train's position, velocity and acceleration are $c_s, \dot{c}_s, \ddot{c}_s$. The goal of the deceleration maneuver is to slow the train to a velocity in $[\dot{c}_{minf}, \dot{c}_{maxf}]$ at position c_f. When the train is not braking its acceleration is exactly zero. When the train is braking its acceleration varies non-deterministically between $[\ddot{c}_{min}, \ddot{c}_{max}]$, both negative. The range is intended to model inherent uncertainty in brake performance. We impose the following constraints on the parameters:

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

The first three constraints are self-explanatory. Since braking is stronger when acceleration is more negative, notice that $\ddot{c}_{min}$ is the strongest braking power, and $\ddot{c}_{max}$ the weakest. The fifth constraint ensures that with the weakest possible braking there is still enough distance to reach the highest allowable speed. 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 Train

We model the train as a single hybrid I/O automaton called *train*. The train accepts brake commands. While braking the train applies an acceleration that is non-deterministic at every point but is constrained

[1] We use the terms "train" and "track" but we could also use "car" and "road", or "vehicle" and "guideway".

to be an integrable function with range in the interval $[\ddot{c}_{min}, \ddot{c}_{max}]$. While not braking the train has exactly zero acceleration.

Inputs: `brakeOn`, `brakeOff`, **Outputs**: none
Variables: $x, \dot{x}, \ddot{x} \in \mathbb{R}$, initially $x = c_s, \dot{x} = \dot{c}_s$, and $\ddot{x} = \ddot{c}_s = 0$
$b \in \{$ `true` , `false` $\}$, initially `false`
$now \in \mathbb{R}^{\geq 0}$, initially zero
Discrete Actions:
`brakeOn`:
Effect: $b' =$ `true` , and $\ddot{x}' \in [\ddot{c}_{min}, \ddot{c}_{max}]$
`brakeOff`:
Effect: $b' =$ `false` , and $\ddot{x}' = 0$
Trajectories:
An I-trajectory w is a trajectory when for all $t \in I$ the following hold:
$w(t).now = w(0).now + t$
if $b =$ `true` then
$w(t).\ddot{x}$ is an integrable function with range $[\ddot{c}_{min}, \ddot{c}_{max}]$
else
$w(t).\ddot{x} = 0$
$w(t).\dot{x} = w(0).\dot{x} + \int_0^t w(s).\ddot{x}\, ds$
$w(t).x = w(0).x + \int_0^t w(s).\dot{x}\, ds$

Table 1: The *train* automaton.

The formal description of *train* appears in Table 3.2. Since this is the first example of a standard automaton definition, we explain some of the major features. The description identifies the inputs, outputs, variables, discrete actions, and trajectories of *train*. The automaton has two inputs and no outputs. The variables x,$\dot{x}$, and $\ddot{x}$ model the position, velocity, and acceleration; b determines whether the train is braking or not; *now* represents time. When describing the effects of a discrete transition, a primed value of a variable refers to the value in the post-state. If no value is assigned to the primed variable then it is assumed to remain unchanged by the transition. Thus, the effect of the `brakeOff` action only changes variables $\ddot{x}$ and b. In the specification of the trajectory set we use the term I-trajectory for a function of the correct type: $I \rightarrow states(train)$ where I is a left-closed interval of $\mathbb{R}^{\geq 0}$ with left end-point equal to zero. Notice that the "now" variable increases linearly with slope 1 during time passage.

3.3 Properties of *train*

We prove some properties of *train* that will be needed later. The two lemmas and three corollaries all relate the initial state and final states of a trajectory. 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 3.1 *For all closed trajectories w of* train *where s is the initial and s' is the final state of w and $\Delta = now' - now$, if $b =$* `false` *then the following hold:*
$\ddot{x}' = \ddot{x} = 0$
$\dot{x}' = \dot{x}$
$x' = x + \dot{x}\Delta$

Proof: By the definition of integrator variable and *train*. ■

Lemma 3.2 *For all closed trajectories w of* train *where s is the initial and s' is the final state of w and $\Delta = now' - now$, if $b =$* `true` *then the following hold:*

$$\begin{array}{rrl} \dot{x} + \ddot{c}_{min}\Delta & \leq \dot{x}' \leq & \dot{x} + \ddot{c}_{max}\Delta \\ x + \dot{x}\Delta + \frac{1}{2}\ddot{c}_{min}\Delta^2 & \leq x' \leq & x + \dot{x}\Delta + \frac{1}{2}\ddot{c}_{max}\Delta^2 \end{array}$$

Proof: We prove only one side of the 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 *For all closed trajectories w of* train *where s is the initial and s' is the final state of w and $\Delta = now' - now$, if $b =$* `true` *then:*

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

Proof: We use Lemma 3.2. The steps for only one side are shown:

$\dot{x}'$	$\leq$	$\dot{x} + \ddot{c}_{max}\Delta$	by 3.2
$\dot{x}' - \dot{x}$	$\leq$	$\ddot{c}_{max}\Delta$	subtract
$\ddot{c}_{max}$	$\leq$	0	assumption
$\frac{\dot{x}'-\dot{x}}{\ddot{c}_{max}}$	$\geq$	Δ	division

■

Corollary 3.4 *For all closed trajectories w of* train *where s is the initial and s' is the final state of w the following holds:*

$b \wedge (0 \leq \dot{x}') \Longrightarrow \frac{(\dot{x}')^2-\dot{x}^2}{2\ddot{c}_{min}} \leq x' - x \leq \frac{(\dot{x}')^2-\dot{x}^2}{2\ddot{c}_{max}}$

Proof: Again, we show only one side of the inequalities. Let $\Delta = now' - now$. Let z be a trajectory as in the proof of Lemma 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 $\dot{x}' = s'.\dot{x}$.

$\dot{u}'$	$=$	$\dot{x} + \ddot{c}_{max}\Delta$	integration
u'	$=$	$x + \dot{x}\Delta + \frac{1}{2}\ddot{c}_{max}\Delta^2$	integration
Δ	$=$	$\frac{\dot{u}'-\dot{x}}{\ddot{c}_{max}}$	solve for Δ
u'	$=$	$x + \frac{\dot{x}\dot{u}'-\dot{x}^2}{\ddot{c}_{max}} + \frac{1}{2}\ddot{c}_{max}\frac{(\dot{u}')^2-2\dot{x}\dot{u}'+\dot{x}^2}{\ddot{c}_{max}^2}$	substitution
u'	$=$	$x + \frac{1}{2\ddot{c}_{max}}(2\dot{x}\dot{u}' - 2\dot{x}^2 + (\dot{u}')^2 - 2\dot{x}\dot{u}' + \dot{x}^2)$	distr.
u'	$=$	$x + \frac{(\dot{u}')^2-\dot{x}^2}{2\ddot{c}_{max}}$	cancel
x'	$\leq$	u'	as in 3.2
x'	$\leq$	$x + \frac{(\dot{u}')^2-\dot{x}^2}{2\ddot{c}_{max}}$	transitivity
0	$\leq$	$\dot{x}'$	anteceedent
$\dot{x}'$	$\leq$	$\dot{u}'$	as in 3.2
$\dot{u}'$	$<$	$\dot{x}$	$(\ddot{c}_{max} < 0)$
x'	$\leq$	$x + \frac{(\dot{x}')^2-\dot{x}^2}{2\ddot{c}_{max}}$	substitution
$x' - x$	$\leq$	$\frac{(\dot{x}')^2-\dot{x}^2}{2\ddot{c}_{max}}$	subtraction

■

3.4 Controller

We define a *brake-controller* to be a hybrid I/O automaton with no input actions, and output actions `brakeOn`, and `brakeOff`. A *correct* brake-controller is one that when composed with *train*, yields a set of finite executions which satisfy the following formal axioms:

1. There exists $t \in \mathbb{R}^{\geq 0}$ such that in all finite executions
   if time progresses past t, then for some state s in the execution $s.x = s.c_f$.

2. If s is the final state of some finite execution, then $(s.x = s.c_f \Longrightarrow s.\dot{c}_{minf} \leq s.\dot{x} \leq s.\dot{c}_{maxf})$.

These can be stated informally as: (1) the train eventually reaches reaches c_f; and (2) when it gets there, it has achieved an appropriate speed. The formal definitions of finite executions and related concepts appears in Appendix A. Note in (1) that the state s where $x = c_f$ can occur during time passage, i.e. within a trajectory. Both of these properties are technically safety properties; the first is a bounded-liveness property and the second is an invariant. For convenience we call the first property the "liveness" property and the second property the "safety" property.

3.5 An Example Controller: *one-shot*

Here we give an example of a correct *brake-controller* called *one-shot*. We provide a description of *one-shot* in MMT shorthand. First we define some convenient constants, A, B, C:

$$A = \frac{1}{\dot{c}_s}\left(c_f - c_s - \frac{\dot{c}_{maxf}^2 - \dot{c}_s^2}{2\ddot{c}_{max}}\right), \quad B = \frac{\dot{c}_{maxf} - \dot{c}_s}{\ddot{c}_{max}}, \quad C = \frac{\dot{c}_{minf} - \dot{c}_s}{\ddot{c}_{min}}$$

Now we define *one-shot* as: The formal description of *one-shot* appears in Table 3.5. The controller is called

Inputs: none, **Outputs:** `brakeOn`, `brakeOff`
Variables: *phase* $\in$ { `idle` , `braking` , `done` }, initially `idle`.
Actions:
`brakeOn`:
Precondition: *phase* = `idle`
Effect: *phase*$'$ = `braking`
`brakeOff`:
Precondition: *phase* = `braking`
Effect: *phase*$'$ = `done`
Tasks: ON = { `brakeOn` } : $[0, A]$
OFF = { `brakeOff` } : $[B, C]$

Table 2: The *one-shot* automaton.

"one-shot" because it applies the brake only once. Since this is the first time the MMT-style notation is used, we attempt to explain the specification informally. The automaton's executions consist of three phases `idle`, `braking`, and `done`. It waits between zero and A time 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 controller is called "one-shot" because it applies the brake only once. The ON task governs the transitions from `idle` to `braking` and the OFF task governs the transitions from `braking` to `done`.

What is a task? A task is a set of actions and an associated lower and upper time bound. The OFF task consists of only the `brakeOff` output action and the bounds B and C. Notice that constraint (6) on the parameters in Section 3.1 ensures that $B \leq C$. The task OFF is *enabled* when the precondition of its action is satisfied, i.e. when *phase* = `braking` . The time bounds ensure that the `brakeOff` action: (1)

cannot occur until it has been continuously enabled for at least B time units and (2) must occur before it is continuously enabled for more than C time units.

When an MMT specification is "translated" into a hybrid I/O automaton, state variables called $first(X)$ and $last(X)$ are added for each task X to keep track of the task's time bounds. For example, the OFF task has associated auxiliary variables $first(OFF)$ and $last(OFF)$. These auxiliary variables are manipulated so as to enforce the time bounds on the OFF task. For example, one of the effects of the `brakeOn` action is to update the deadlines for the OFF task as follows: $first'(OFF) = now + B$ and $last'(OFF) = now + C$. A precondition of the `brakeOff` action is that $first(OFF) \leq now$. These auxiliary variables are described in more detail in Section B.

3.6 Correctness of *one-shot-system*

The following lemmas and corollaries describe properties of the composition of *train* and *one-shot* called *one-shot-system*. Almost all the properties are invariant assertions. The first subsection proves the liveness property, the second proves the safety property. Together they establish the correctness of the controller.

3.6.1 Liveness

In this subsection we prove the liveness 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 $\dot{c}_{minf}$. We do this by characterizing velocity for each of the three phases: `idle` in Lemma 3.7, `braking` in Lemma 3.8, and `done` in Lemma 3.9. Some of the results are more general than necessary because they will be used in the safety section.

The following two technical lemmas will be used in the correctness proof.

Lemma 3.5 *In all reachable states of* one-shot *the following holds:*
$(phase =$ `idle` $) \Longrightarrow (first(ON) = 0 \wedge last(ON) = A)$

Proof: By induction. ■

Lemma 3.6 *In all reachable states of* one-shot-system *the following hold:*
1. $(b \Longrightarrow \ddot{x} \in [\ddot{c}_{min}, \ddot{c}_{max}])$ *2.* $(\neg b \Longrightarrow \ddot{x} = 0)$
3. $b \Longleftrightarrow (phase =$ `braking` $)$

Proof: Trivial induction. ■

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

Lemma 3.7 *In all reachable states of* one-shot-system, *if* $phase =$ `idle` *the following hold:*

1. $\dot{x} = \dot{c}_s$
2. $x = c_s + (now)\dot{c}_s$

Proof: By induction. The interesting case is time passage where we note that $\ddot{x} = 0$ and Lemma 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 reasoning about the controller's deadline variables. While in the `braking` phase, $last(OFF) - now$ is the greatest amount of time the system will continue braking. This time must be bounded in order to avoid slowing down below the minimum final speed, $\dot{c}_{minf}$. A similar result holds for $first(OFF)$ and the upper bound on velocity.

Lemma 3.8 *In all reachable states of* one-shot-system, *if phase* = `braking` *the following hold:*

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

Proof: By induction. The two interesting cases are the ON task that sets *phase* = `braking` and a time passage step while *phase* = `braking` . For the ON task the pre-state has *phase* = `idle` and Lemma 3.7 and the definitions of B and C yield the desired results as follows (only (1) is shown):

B	$=$	$\frac{\dot{c}_{maxf} - \dot{c}_s}{\ddot{c}_{max}}$	by definition
$\dot{x}$	$=$	$\dot{c}_s = \dot{x}'$	by 3.7
$first(OFF)'$	$=$	$now' + B$	*one-shot* definition
$first(OFF)' - now'$	$=$	$\frac{\dot{c}_{maxf} - \dot{x}}{\ddot{c}_{max}}$	subst. and subt.

For time passage steps we use Lemma 3.6 and the equation from Corollary 3.3. Subtraction and expansion of $\Delta = now' - now$ yields the desired results as follows (only (1) is shown):

$now' - now$	$\leq$	$\frac{\dot{x}' - \dot{x}}{\ddot{c}_{max}}$	by 3.3.
$first(OFF) - now$	$\geq$	$\frac{\dot{c}_{maxf} - \dot{x}}{\ddot{c}_{max}}$	inductive hyp.
$first(OFF) - now'$	$\geq$	$\frac{\dot{c}_{maxf} - \dot{x}'}{\ddot{c}_{max}}$	subt. and cancel

■

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

Corollary 3.9 *In all reachable states of* one-shot-system, *if phase* = `done` *the following holds:* $\dot{c}_{maxf} \geq \dot{x} \geq \dot{c}_{minf}$

Proof: By induction. The interesting cases are the OFF action and time passage in the `done` phase. For the OFF action we know that in the pre-state *phase* = `braking` so Lemma 3.8 applies. Furthermore $first(OFF) \leq now \leq last(OFF)$ by a property of MMT automata. From this we can conclude that $\dot{c}_{maxf} \geq \dot{x} \geq \dot{c}_{minf}$ (details for one side shown below). For the time passage step, we know that $\ddot{x} = 0$ so $\dot{x} = \dot{x}'$, by Lemma 3.6 and Lemma 3.1.

$first(OFF) - now$	$\geq$	$\frac{\dot{c}_{maxf} - \dot{x}}{\ddot{c}_{max}}$	from 3.8
$first(OFF)$	$\leq$	now	MMT property
$first(OFF) - now$	$\leq$	0	subtraction
0	$\geq$	$\frac{\dot{c}_{maxf} - \dot{x}}{\ddot{c}_{max}}$	trans.
0	$>$	$\ddot{c}_{max}$	assumption
0	$\leq$	$\dot{c}_{maxf} - \dot{x}$	mult.
$\dot{x}$	$\leq$	$\dot{c}_{maxf}$	subtr.

■

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

Lemma 3.10 *In all reachable states of* one-shot-system $\dot{x} \geq \dot{c}_{minf}$.

Proof: We break on cases of *phase*. When *phase* = `idle` Lemma 3.7 gives $\dot{x} = \dot{c}_s$ and by assumption $\dot{c}_s > \dot{c}_{maxf} \geq \dot{c}_{minf}$. When *phase* = `braking` , Lemma B.1 gives $now \leq last(OFF)$ and Lemma 3.8 gives the desired result. Finally when *phase* = `done` corollary 3.9 applies. ■

Corollary 3.11 *In all reachable states of* one-shot-system*:*

$$x \geq c_s + \dot{c}_{minf}(now)$$

Proof: Lemma 3.10 establishes that in all reachable states (including those in trajectories) $\dot{x} \geq \dot{c}_{minf}$. At all times $x - c_s$ is the integral of $\dot{x}$. It is a property of definite integrals that lower bounds are preserved. Therefore $x - c_s \geq \int_0^{now} \dot{c}_{minf}\, dt = \dot{c}_{minf}(now)$. ■

Since $\dot{c}_{minf} > 0$, the above lemma guarantees that for any admissible timed execution of *one-shot-system* for all states where $now \geq \frac{c_f - c_s}{\dot{c}_{minf}}$, we have $x \geq c_f$. This is the t required by the liveness property.

3.6.2 Safety

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

$$(x = c_f \implies \dot{c}_{minf} \leq \dot{x} \leq \dot{c}_{maxf})$$

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

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

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 $\ddot{c}_{max}$ is negative). Once again, we prove the invariant for each phase (3.12, 3.13, 3.14) and combine the results (3.15). The safety property is proved in corollary 3.16.

Lemma 3.12 *In all reachable states of* one-shot-system*, if phase =* `idle` *then* $c_f - x \geq \frac{\dot{c}_{maxf}^2 - \dot{x}^2}{2\ddot{c}_{max}}$

Proof: By Lemmas B.1 and 3.5 we know $now \leq A$. Using the equations for $\dot{x}$ and x from Lemma 3.7 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 $now \leq A$
$c_s + (now)\dot{c}_s$	$\leq$	$c_f - \frac{\dot{c}_{maxf}^2 - \dot{c}_s^2}{2\ddot{c}_{min}}$	mult. $\dot{c}_s$ and add c_s
x	$=$	$c_s + (now)\dot{c}_s$	from 3.7
x	$\leq$	$c_f - \frac{\dot{c}_{maxf}^2 - \dot{c}_s^2}{2\ddot{c}_{min}}$	$=$ and $\leq$ transitive
$c_f - x$	$\geq$	$\frac{\dot{c}_{maxf}^2 - \dot{c}_s^2}{2\ddot{c}_{min}}$	subt. c_f and reverse sign

■

Lemma 3.13 *In all reachable states of* one-shot-system*, 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 time passage while *phase =* `braking` . In the ON task case Lemma 3.12 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 time passage case we substitute from Lemma 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 hyp
$x' - x$	$\leq$	$\frac{\dot{x}'^2 - \dot{x}^2}{2\ddot{c}_{max}}$	from 3.4
$c_f - x - x' + x$	$\geq$	$\frac{\dot{c}_{maxf}^2 - \dot{x}^2 - \dot{x}'^2 + \dot{x}^2}{2\ddot{c}_{max}}$	subt.
$c_f - x'$	$\geq$	$\frac{\dot{c}_{maxf}^2 - \dot{x}'^2}{2\ddot{c}_{max}}$	cancel

■

Lemma 3.14 *In all reachable states of* one-shot-system, *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 3.9. 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 3.15 *In all reachable states of* one-shot-system, *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 3.12, 3.13, and 3.14. ■

Corollary 3.16 *In all reachable states of* one-shot-system*:*

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

Proof: Directly using 3.15 and 3.10. ■

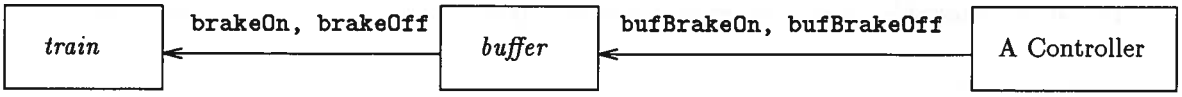

Figure 2: Overview of Delay Deceleration Model

4 Delay

In this section we extend the model of the train by nondeterministically delaying the acceleration commands. Rather than modify the train automaton itself, we introduce a new automaton called *accel-buffer* that will serve as a buffer between the train and a controller. Figure 2 illustrates the components and their communication.

In the following subsections we present the buffer, extend the correctness criteria, give an example controller, and prove that it is correct. The proof of correctness relies on the correctness of the previous controller and a simulation mapping. Note that the specification of the train remains unchanged.

4.1 The Buffer

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 interval $[\delta^-, \delta^+]$ (where $0 \leq \delta^- \leq \delta^+$).

Inputs: `bufBrakeOn`, `bufBrakeOff`, **Outputs:** `brakeOn`, `brakeOff`
Variables: *request* $\in$ { `on` , `off` , `none` }, initially `none`
violation $\in$ { `true` , `false` }, initially `false`
Actions:
`bufBrakeOn`:
Effect: Cases of *request*,
`on` : no effect
`off` : *violation*$'$ = `true`
`none` : *request*$'$ = `on`
`bufBrakeOff`:
Effect: Cases of *request*,
`on` : *violation*$'$ = `true`
`off` : no effect
`none` : *request*$'$ = `off`
`brakeOn`:
Precondition: *request* = `on`
Effect: *request*$'$ = `none`
`brakeOff`:
Precondition: *request* = `off`
Effect: *request*$'$ = `none`
Tasks: $BUFF$ = { `brakeOn`, `brakeOff` } : $[\delta^-, \delta^+]$

Table 3: The *buffer* automaton.

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 variable *violation* is true when the buffer overflows.

4.2 Definition of a valid buffered brake controller

We modify the definition of a valid controller. A valid *buffered-brake-controller* is a timed I/O automaton with output actions `bufBrakeOn` and `bufBrakeOff` that when composed with both *brake-buffer* and *train* yields a timed I/O automaton whose finite executions satisfy the "liveness" and "safety" properties of Section 3.4 and the additional invariant:

$violation =$ `false`

This property guarantees that the controller never overflows the buffer; we call it the "non-violation" property.

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, $[\dot{c}_{minf}, \dot{c}_{maxf}]$ to be wider and hence the controller's task easier. These are the additional constraints:

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

The first constraint ensures that the delay interval is well-defined. The next two are necessary to ensure the non-violation. The last constraint replaces constraint number six in Section 3.1; the new version accounts not only for the non-determinism of the braking strength but also for the buffer. The other five original constraints remain as well but are not shown here.

4.4 Example buffered brake controller: *buffered-one-shot*

Here we give an example of a valid *buffered-brake-controller* called *buffered-one-shot*. This automaton is identical to *one-shot* (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:

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

Buffered-one-shot-system is the name for the composition of *buffered-one-shot*, *buffer*, and *train*. We will also be interested in *buffer-and-one-shot*, the composition of only the buffer and the controller.

4.5 Proof of Correctness for *buffered-one-shot*

The proof of correctness of the controller requires proofs of the non-violation, liveness, and safety properties. In the next Section 4.5.1 we prove non-violation. In Section 4.5.2 we prove liveness and safety using a simulation mapping to the unbuffered case. The liveness and safety results of the unbuffered case extend via the simulation to this case.

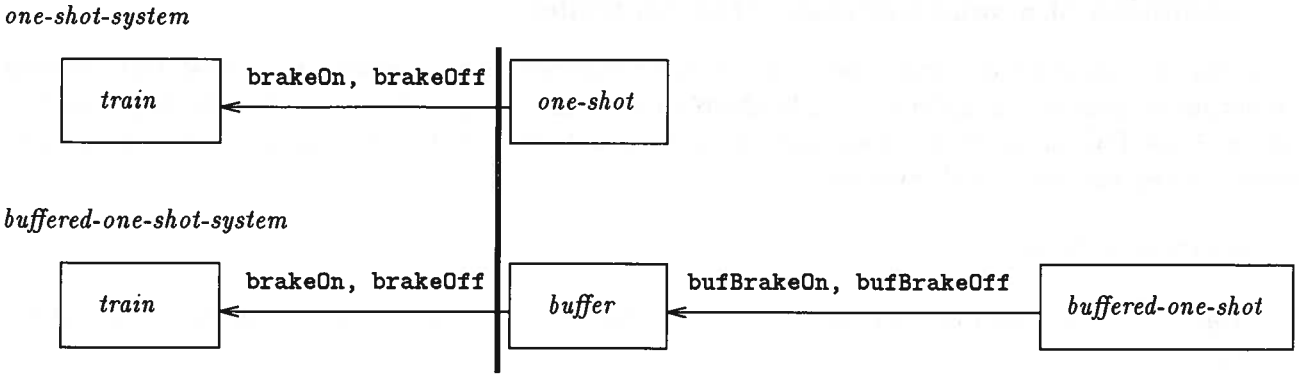

Figure 3: Systems with and without buffering compared. The thick line is a common interface.

4.5.1 Non-Violation

Non-violation is proved directly.

Lemma 4.1 *In all reachable states of* buffered-one-shot-system *violation* = `false` .

Proof: Violation occurs when *request* $\neq$ `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* $\neq$ `none` . The following invariant of the system is sufficient:

$$request \neq \texttt{none} \implies (last(BUFF) \leq first(ON)) \wedge (last(BUFF) \leq first(OFF))$$

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

4.5.2 Liveness and Safety

In this section we prove the liveness and safety properties for *buffered-one-shot-system* via a simulation mapping. The simulation is a mapping between the original controller *one-shot* and *buffer-and-one-shot* (the composition of the buffer and the controller but not the train.) Notice that the safety and liveness properties only mention variables in *train*. In light of this, it may appear counter-intuitive that the simulation mapping excludes the train. We explain this informally here and more formally in at the end of this section. Consider Figure 3 which shows the automata and inter-automata communication of *one-shot-system* and *buffered-one-shot-system* 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 *buffer-and-one-shot* is a subset of the external behavior of *one-shot*. Their external behavior is precisely the interface across the dark line and this is all *train* experiences; therefore *train's* behavior in the buffered case is a subset of its behavior in the unbuffered case. Thus, invariants that only involve 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 a rigorous version of the preceding discussion.

Supporting Lemmas

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

Lemma 4.2 *In all reachable states of* buffer-and-one-shot *exactly one of the following is true:*

1. $phase =$ `idle` $\wedge$ $request =$ `none`
2. $phase =$ `braking` $\wedge$ $request =$ `on`
3. $phase =$ `braking` $\wedge$ $request =$ `none`
4. $phase =$ `done` $\wedge$ $request =$ `off`
5. $phase =$ `done` $\wedge$ $request =$ `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 4.1. ■

The following two technical lemmas make the simulation proof more readable — the same arguments could be embedded in the simulation proof itself. Both lemmas concern the time bounds on the idle phase.

Lemma 4.3 *In all reachable states of* buffered-one-shot, *the following holds:*
$phase =$ `idle` $\Longrightarrow first(ON) = 0 \wedge last(OFF) = A'$.

Proof: Exactly analogous to 3.5. ■

Lemma 4.4 *In all reachable states of* buffered-one-shot-system *the following holds:* ($phase =$ `braking` $\wedge$ $request \neq$ `none`)
$\Longrightarrow last(BUFF) \leq A' + \delta^+ = A$.

Proof: Simple induction, uses 4.2. ■

Simulation

In this section we present a simulation relation R from *buffer-and-one-shot* 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 (buffered, *buffer-and-one-shot*, and u denote a state in the specification (unbuffered, *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 $u.phase =$ `idle`
   (b) `braking` , by cases of $s.request$:
      i. `on` , then $u.phase =$ `idle`
      ii. `none` , then $u.phase =$ `braking` and
      $u.first(OFF) \leq s.first(OFF) + \delta^-$ and
      $u.last(OFF) \geq s.last(OFF) + \delta^+$
   (c) `done` , by cases of $s.request$:
      i. `off` , then $u.phase =$ `braking` and
      $u.first(OFF) \leq s.first(BUFF)$ and
      $u.last(OFF) \geq s.last(BUFF)$
      ii. `none` , then $u.phase =$ `done`

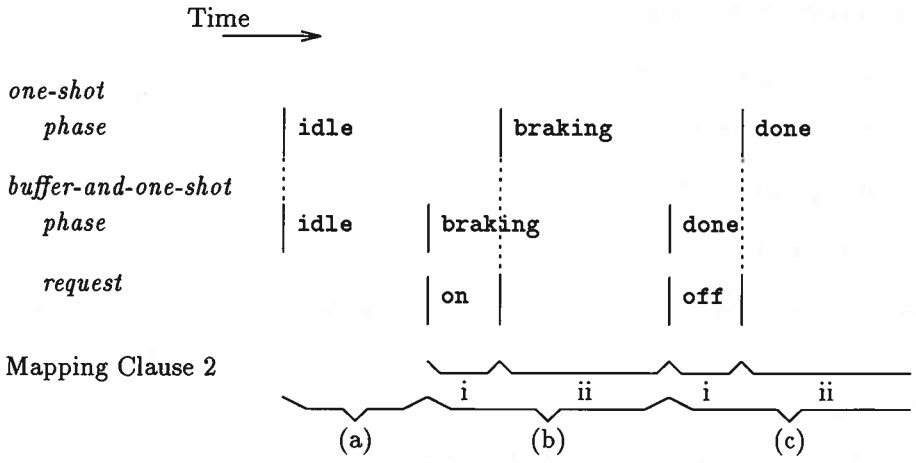

Figure 4: Overview of Simulation Mapping

Intuitively, the simulation is mapping the "virtual" phases of *buffer-and-one-shot* to the actual phases of *one-shot* . This is illustrated in Figure 4. The figure depicts an execution of *one-shot* above a corresponding execution of *buffered-on-shot.* A virtual phase of *buffered-one-shot* is 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 *The above relation R is a simulation mapping.*

Proof: Let s lead to s' via action π in the implementation 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 on the type of π:

1. If π is a time passage step then we must show that there is an equivalent time passage step 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:

$$min\{u.last(ON), u.last(OFF)\} \geq min\{s.last(ON), s.last(OFF), s.last(BUFF)\}$$

   Cases by $u.phase$:

   (a) $u.phase =$ `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. $s.phase =$ `idle` – then $s.last(OFF) = \infty$ and $s.last(BUFF) = \infty$ (by automaton definition and 4.2). By lemmas 3.5 and 4.3 $u.last(ON) = A$ and $s.last(ON) = A'$ and by definition $A \geq A'$.

      ii. $s.phase =$ `braking` $\wedge$ $s.request \neq$ `none` – Follows from lemmas 3.5 and 4.4.

   (b) $u.phase =$ `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. $s.phase =$ `braking` $\wedge\, s.request =$ `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. $s.phase =$ `done` $\wedge\, s.request \neq$ `none` – then $s.last(ON) = s.last(OFF) = \infty$. By clause 2(c)i of the relation $u.last(OFF) = s.last(BUFF)$.

(c) $u.phase =$ `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 $LO.phase =$ `idle` by the definition of the *buffered-one-shot* automaton. Also note that $s.request =$ `none` by Lemma 4.1 (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 4.2 we know that $s.phase =$ `braking` . Therefore by clause 2*a* of the relation we know that $u.phase =$ `idle` . Since $u.first(ON) = 0$ by Lemma 3.5, `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 4.2, $s.phase =$ `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

This section requires familiarity with Appendix A. In order to simplify the exposition of this section, let T denote the *train* automaton, C denote *one-shot* and B denote the composition of *buffer* and *buffered-one-shot*. The above simulation R is a forward simulation from B to C. By Theorem A.1 we conclude that:

$$t\text{-}traces^*(B) \subseteq t\text{-}traces^*(C)$$

In turn, by Theorem A.4 we can conclude that:

$$t\text{-}execs^*(A \times B)|A \subseteq t\text{-}execs^*(A \times C)|A$$

As first presented in Section 3.4, the liveness and safety properties are defined over the finite timed executions of $A \times C$. The similar properties for the buffered case are the identical properties but for the finite timed executions of $A \times B$. Since the properties only mention the variables of A, we need only consider the projection of finite timed executions on A. Therefore, the above inclusion of the projected executions is sufficient.

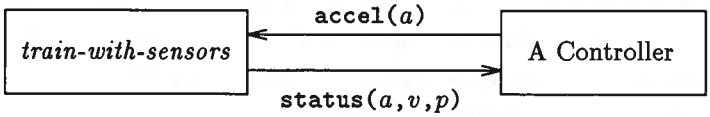

Figure 5: Overview of Feedback Deceleration Model

5 Sensor Feedback

In this section 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 *train-with-sensors*. We also define correctness conditions, give an example controller and prove that it is correct. Figure 5 illustrates the components and their communication.

5.1 Train with Sensors

The *train-with-sensors* automaton appears in Table 5.1. It accepts `accel(`a`)` messages which are requests to accelerate at a rate $a \in [\ddot{c}_{min}+\ddot{c}_{err}, \ddot{c}_{max}]$. If a is the requested acceleration then the achieved acceleration of the train is in the interval $[a - \ddot{c}_{err}, a]$. The train provides sensor information periodically; it sends a `status` message giving the current values of acc, $\dot{x}$, and x every δ_{sense} time units.

Inputs: `accel(`a`)` for $a \in [\ddot{c}_{min} + \ddot{c}_{err}, \ddot{c}_{max}]$
Outputs: `status(`a,v,p`)` for $a, v, p \in \mathbb{R}$
Variables: $x, \dot{x}, \ddot{x} \in \mathbb{R}$, initially $x = c_s$, $\dot{x} = \dot{c}_s$, and $\ddot{x} = \ddot{c}_s$
$acc \in [\ddot{c}_{min} + \ddot{c}_{err}, \ddot{c}_{max}]$, initially $\ddot{c}_s$
$next, now \in \mathbb{R}^{\geq 0}$, both initially zero
Discrete Actions:
`accel(`a`)`:
Effect: $acc' = a$ and $\ddot{x}' \in [a - \ddot{c}_{err}, a]$
`status(`a,v,p`)`:
Precondition: $a = acc$, $v = \dot{x}$, $p = x$, and $now = next$
Effect: $next' = now + \delta_{sense}$
Trajectories:
An I-trajectory w is a trajectory when for all $t \in I$ the following hold:
$w(t).now = w(0).now + t$
$w(t).\ddot{x}$ is an integrable function with range $[acc - \ddot{c}_{err}, acc]$
$w(t).\dot{x} = w(0).\dot{x} + \int_0^t w(s).\ddot{x}\ ds$
$w(t).x = w(0).x + \int_0^t w(s).\dot{x}\ ds$

Table 4: The *train-with-sensors* automaton.

5.2 Properties of *train-with-sensors*

The following two properties of *train-with-sensors* are similar to the *train* properties of Section 3. The first bounds change in velocity by change in time. The second bounds change in position by change in velocity.

Lemma 5.1 *For all closed trajectories w of* train-with-sensors *where s is the initial and s' is the final state of w the following holds:*

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

Proof: As in the first part of Lemma 3.2, except that acc and $(acc - \ddot{c}_{err})$ replace $\ddot{c}_{max}$ and $\ddot{c}_{min}$ respectively. ■

Lemma 5.2 *For all closed trajectories w of* train-with-sensors *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:*

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

Proof: Similar to Lemma 3.4, except that acc and $(acc - \ddot{c}_{err})$ replace $\ddot{c}_{max}$ and $\ddot{c}_{min}$ respectively. ■

The following property is like the $now \leq last(\cdot)$ property for MMT automata, Theorem B.1.

Lemma 5.3 *The following is an invariant of* train-with-sensors*:*

$$0 \leq next - now \leq \delta_{sense}$$

Proof: Simple Induction. ■

5.3 Controller under Feedback

We define a correct *controller-under-feedback* to be a hybrid I/O automaton that when composed with *train-with-sensors* yields an automaton that satisfies the "liveness" and "safety" properties from Section 3.4.

5.4 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. $\delta_{sense} > 0$
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. $\dot{c}_{maxf} - \dot{c}_{minf} \geq -\ddot{c}_{min}\delta_{sense}$

Note that these constraints supersede the original constraints given in section 3.

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

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

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

Inputs: `status(a,v,p)` for $a, v, p \in \mathbb{R}$
Outputs: `accel(a)` for $a \in [\ddot{c}_{min} + \ddot{c}_{err}, \ddot{c}_{max}]$
Variables: $send \in [\ddot{c}_{min} + \ddot{c}_{err}, \ddot{c}_{max}] \cup \{$ none $\}$, initially none
Discrete Actions:
`status(a,v,p)`:
Effect:
if $v \leq \dot{c}_{maxf}$ then
$send' = \min\left(\ddot{c}_{max}, \frac{\dot{c}_{maxf} - v}{\delta_{sense}}\right)$
`accel(a)`:
Precondition: $send = a$
Effect: $send' =$ none
Trajectories:
An I-trajectory w is a trajectory when for all $t \in I$ the following hold:
$w(t).now = w(0).now + t$
$w(0).send = w(t).send =$ none

Table 5: The *fall-and-zig-zag* automaton.

5.5 Example Controller under Feedback

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 *fall-and-zig-zag*. The composed system is called *feedback-system*. We describe *fall-and-zig-zag* in timed I/O automaton format in Table 5.5.

The controller is called *fall-and-zig-zag* because of the the shape of the curve in $\dot{x} \times now$ space of the worst-case behavior of the composed system. Figure 5.5 depicts a possible behavior for the system; it assumes constant acceleration. The train begins at time zero with velocity $\dot{c}_s$ and acceleration $\ddot{c}_s$. If it achieved $\ddot{c}_s$ acceleration it would reach the goal velocity of $\dot{c}_{maxf}$ (the upper dotted line). However, for the first three δ_{sense} periods it only achieves $\ddot{c}_s - \ddot{c}_{err}$ acceleration (the solid line). At that point the controller sees that $\dot{x} \leq \dot{c}_{maxf}$ and changes the acceleration (first bend in solid line). Every δ_{sense} time units the controller continues to adjust acceleration to maintain the trains velocity in the target range $[\dot{c}_{minf}, \dot{c}_{maxf}]$.

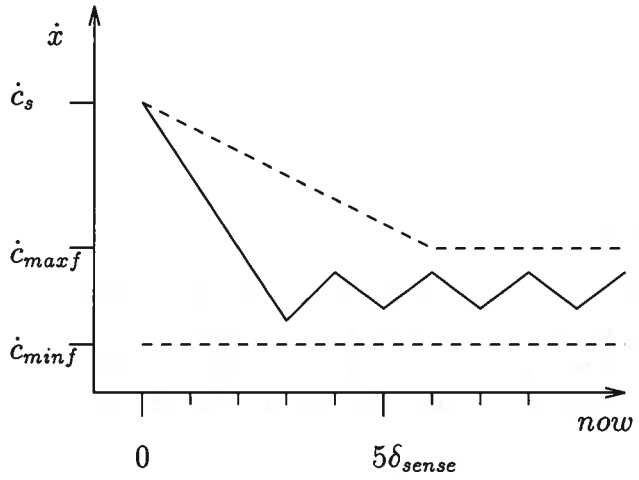

Figure 6: Possible behavior of *feedback-system*.

5.6 Correctness of *feedback-system*

The structure of the proof is very similar to that of the simple case examined in Section 3: first, we show the liveness property via a global lower bound on velocity; second, we show the safety property via a more complex invariant that has as a sub-case the invariant used in Section 3. To make the structure of the overall proof more clear and increase general readability, we have relegated the large proofs of the two main lemmas to their own subsections.

5.6.1 Liveness

In this section we prove the liveness property. The first lemma is a technical lemma needed in the proof of the second one.

Lemma 5.4 *The following is an invariant of* feedback-system*:*

$$send \neq \text{none} \implies next = now + \delta_{sense}$$

Proof: Trivial induction. ■

The next lemma is the major new result needed to prove the liveness property. It is an invariant that describes a lower bound on velocity for the near future, i.e. the current sensory interval. The associated corollary states the resulting global property. This property uses a set of implications with mutually exclusive and exhaustive anteceedents. Each implication corresponds to one of the periodic logical phases of the system. This type of invariant is seen again later in more complex forms.

Lemma 5.5 *The conjunction of the following statements is an invariant of* feedback-system*:*

$$send = \text{none} \implies \dot{x} \geq \dot{c}_{minf} \wedge \dot{x} + (acc - \ddot{c}_{err})(next - now) \geq \dot{c}_{minf}$$
$$send \neq \text{none} \implies \dot{x} \geq \dot{c}_{minf} \wedge \dot{x} + (send - \ddot{c}_{err})\delta_{sense} \geq \dot{c}_{minf}$$

Proof appears in Section 5.6.3.

Corollary 5.6 *The following is an invariant of* feedback-system*:*

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

Proof: Directly from 5.5. The anteceedents form an exhaustive set of cases, and in all cases the property is true. ■

This leads to the liveness property as Lemma 3.10 did in Section 3. The corollaries which yield the liveness property are exactly analogous and are not restated here.

5.6.2 Safety

The following technical lemma is needed for the subsequent lemma.

Lemma 5.7 *For all w trajectories of* feedback-system *where s is the initial state and s′ is the final state of w. If* $acc = \ddot{c}_s$, $x \leq c_f$, *and* $x' \leq c'_f$ *then,*

$$c_f - x \geq \frac{\dot{c}^2_{maxf} - \dot{x}^2}{2\ddot{c}_s} \implies c_f - x' \geq \frac{\dot{c}^2_{maxf} - (\dot{x}')^2}{2\ddot{c}_s}$$

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

$$
\begin{array}{rcll}
acc & = & \ddot{c}_s \leq 0 & \text{assumption} \\
c_f - x & \geq & \frac{\dot{c}_{maxf}^2 - \dot{x}^2}{2\ddot{c}_s} & \text{assumption} \\
x' - x & \leq & \frac{(\dot{x}')^2 - \dot{x}^2}{2acc} & \text{by Lemma 5.2} \\
x - x' & \geq & \frac{\dot{x}^2 - (\dot{x}')^2}{2acc} & \text{multiply} \\
c_f - x + x - x' & \geq & \frac{\dot{c}_{maxf}^2 - \dot{x}^2 + \dot{x}^2 - (\dot{x}')^2}{2acc} & \text{add} \\
c_f - x' & \geq & \frac{\dot{c}_{maxf}^2 - (\dot{x}')^2}{2\ddot{c}_s} & \text{cancel}
\end{array}
$$

■

The following lemma is the major result needed to prove the safety property. It is similar to two other results: (1) Corollary 3.15 and its supporting lemmas, which used a similar equation to bound "distance remaining"; and, (2) Lemma 5.5 of this section, which provides a set of implication with an exhaustive set of anteceedents.

Lemma 5.8 *The conjunction of the following statements is an invariant of* feedback-system*:*

$$
\begin{array}{rcl}
\dot{x} > \dot{c}_{maxf} & \Longrightarrow & acc = \ddot{c}_s \wedge send = \text{ none } \wedge \left((x \leq c_f) \Longrightarrow c_f - x \geq \frac{\dot{c}_{maxf}^2 - \dot{x}^2}{2\ddot{c}_s} \right) \\
\dot{x} \leq \dot{c}_{maxf} \wedge send = \text{ none } & \Longrightarrow & (\dot{x} + acc(next - now)) \leq \dot{c}_{maxf} \\
\dot{x} \leq \dot{c}_{maxf} \wedge send \neq \text{ none } & \Longrightarrow & (\dot{x} + send(\delta_{sense})) \leq \dot{c}_{maxf}
\end{array}
$$

Proof appears in Section 5.6.4.

The following corollaries correspond directly to Corollary 3.15 and Corollary 3.16.

Corollary 5.9 *The following is an invariant of* feedback-system*:*

$$(x \leq c_f) \Longrightarrow c_f - x \geq \frac{\dot{c}_{maxf}^2 - \dot{x}^2}{2\ddot{c}_s}$$

Proof: Directly from 5.8. If the first implication applies, then it appears in the consequent. If the second implication or third applies, then $\dot{c}_{maxf}^2 - \dot{x}^2$ is positive, hence, the fraction is negative and the inequality holds. These cases are exhaustive. ■

Corollary 5.10 *The following is an invariant of* feedback-system*:*

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

Proof: Directly from 5.9 and 5.6. ■

This establishes the safety property.

5.6.3 Proof of Lemma 5.5:

Proof: By induction. Notice that the antecedents of the three 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 anteceedent is true and that it *holds* when it applies and its consequent is true. Basis: in the initial state $\dot{x} > \dot{c}_{maxf}$ so Rule 1 applies. It holds because of our assumptions on the parameters, the definition of $\ddot{c}_s$, 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 step called π. 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 π: `accel`, `status`, or time passage.

1. $\pi =$ `accel` : notice that $send \neq$ `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 $send' =$ `none` . Therefore Rule 1 must apply in s'. We must show that it holds. Clearly, $\dot{x}' = \dot{x} \geq \dot{c}_{minf}$ by the inductive hypothesis. By Lemma 5.4 $next - now = \delta_{sense}$ and because none of these variables change $next' - now' = \delta_{sense}$. By substituting $next' - now' = \delta_{sense}$, $acc' = send$ and $send' =$ `none` into the inequality in Rule 2 we get :
$\dot{x} + (acc' - \ddot{c}_{err})(next' - now') = \dot{x} + (send - \ddot{c}_{err})\delta_{sense} \geq \dot{c}_{minf}$
This shows that Rule 1 holds in s'.

2. $\pi =$ `status` : notice that $next = now$ by the action's precondition, so $next \neq now + \delta_{sense}$ and by the contra-positive of Lemma 5.4 $send =$ `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 $\dot{x}' = \dot{x} = v \geq \dot{c}_{maxf}$, $next' - now' = \delta_{sense}$, and $acc' - \ddot{c}_{err} \geq \ddot{c}_{min}$. By assumption on the parameters: $\dot{c}_{maxf} - \dot{c}_{minf} > -\ddot{c}_{min}\delta_{sense}$. From these, we reach the desired conclusion with some algebra:

$\dot{c}_{maxf} - \dot{c}_{minf}$	$\geq$	$-\ddot{c}_{min}\delta_{sense}$	param.
$\dot{c}_{maxf} + \ddot{c}_{min}\delta_{sense}$	$\geq$	$\dot{c}_{minf}$	subtr.
$\dot{x}'$	$\geq$	$\dot{c}_{maxf}$	auto. def.
$\dot{x}' + \ddot{c}_{min}\delta_{sense}$	$\geq$	$\dot{c}_{minf}$	subst.
δ_{sense}	$>$	0	param.
$acc' - \ddot{c}_{err}$	$\geq$	$\ddot{c}_{min}$.	auto.def.
$\dot{x}' + (acc' - \ddot{c}_{err})\delta_{sense}$	$\geq$	$\dot{c}_{minf}$	subst.
δ_{sense}	$=$	$next' - now'$	auto. def.
$\dot{x}' + (acc' - \ddot{c}_{err})(next' - now')$	$\geq$	$\dot{c}_{minf}$	subst.

   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 $\dot{x} \geq \dot{c}_{minf}$. This is half of Rule 2; it remains to show the other half. According to the automata definitions: $send' = \min(\ddot{c}_{max}, \frac{\dot{c}_{maxf} - \dot{x}}{\delta_{sense}})$. By assumption on the parameters $\ddot{c}_{max} - \ddot{c}_{err} \geq 0$, therefore if $send' = \ddot{c}_{max}$ Rule 2 applies trivially. Assume that $send' = \frac{\dot{c}_{maxf} - \dot{x}}{\delta_{sense}} < \ddot{c}_{max}$. Some algebra yields the desired result:

$\ddot{c}_{min} + \ddot{c}_{err}$	$<$	0	param. assum.
δ_{sense}	$>$	0	param. assum.
$-\ddot{c}_{min}\delta_{sense}$	$>$	$\ddot{c}_{err}\delta_{sense}$	subtr. & mult.
$\dot{c}_{maxf} - \dot{c}_{minf}$	$\geq$	$-\ddot{c}_{min}\delta_{sense}$	param. assum.
$\dot{c}_{maxf} - \dot{c}_{minf}$	$\geq$	$\ddot{c}_{err}\delta_{sense}$	transitivity
$\dot{c}_{maxf} - \ddot{c}_{err}\delta_{sense}$	$\geq$	$\dot{c}_{minf}$	subtract
$\dot{x}' + \dot{c}_{maxf} - \dot{x}' - \ddot{c}_{err}\delta_{sense}$	$\geq$	$\dot{c}_{minf}$	anti-cancel
$\dot{x}' + \left(\frac{\dot{c}_{maxf} - \dot{x}}{\delta_{sense}} - \ddot{c}_{err}\right)\delta_{sense}$	$\geq$	$\dot{c}_{minf}$	anti-distribute
$send'$	$=$	$\frac{\dot{c}_{maxf} - \dot{x}}{\delta_{sense}}$	assumption
$\dot{x}' + (send' - \ddot{c}_{err})\delta_{sense}$	$\geq$	$\dot{c}_{minf}$	substitute

   Thus Rule 2 holds in s'.

3. π is a time passage step, 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 cases uses Lemma 5.1, the inductive hypothesis and some simple algebra.
Notice that $acc = acc'$, so let $X = (acc - \ddot{c}_{err}) = (acc' - \ddot{c}_{err})$:

$$\begin{aligned}
\dot{x} + X(next - now) &\geq \dot{c}_{minf} && \text{ind. hypothesis} \\
\dot{x}' - \dot{x} &\geq X(now' - now) && \text{by 5.1} \\
\dot{x}' - \dot{x} - X(now' - now) &\geq 0 && \text{subtract} \\
\dot{x} + \dot{x}' - \dot{x} + X(next - now) - X(now' - now) &\geq \dot{c}_{minf} && \text{add} \\
\dot{x}' + X(next - now') &\geq \dot{c}_{minf} && \text{cancel}
\end{aligned}$$

For the $\dot{x} \geq \dot{c}_{minf}$ requirement: if $X \geq 0$ then $\dot{x}' \geq \dot{x} \geq \dot{c}_{minf}$; otherwise, $\dot{x}' \geq \dot{x}' + X(next - now') \geq \dot{c}_{minf}$ (by Lemma 5.3). Thus Rule 1 holds in s'.

■

5.6.4 Proof of Lemma 5.8:

Proof: This is an inductive proof very similar to the proof of Lemma 5.5 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 called π. 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 π: time passage, `accel`, or `status`.

1. $\pi =$ `accel` : Either $\dot{x} \leq \dot{c}_{maxf}$ or not.
   (a) $\dot{x} \leq \dot{c}_{maxf}$: This case is exactly analogous to the $\pi =$ `accel` case of the proof of Lemma 5.5. Here, Rule 3 holds in state s and Rule 2 can be shown to hold in state s'. We omit the proof.
   (b) $\dot{x} > \dot{c}_{maxf}$: by the inductive hypothesis Rule 1 holds in s and therefore $send =$ `false` ; however in that case, this action was not enabled in s. Therefore $\dot{x} > \dot{c}_{maxf}$ is impossible for the `accel` action case.
2. $\pi =$ `status` : Either $\dot{x} \leq \dot{c}_{maxf}$ or not.
   (a) $\dot{x} \leq \dot{c}_{maxf}$: This case is exactly analogous to the $\pi =$ `status` case of the proof of Lemma 5.5. Here, Rule 2 holds in state s and Rule 3 can be shown to hold in state s'. We omit the proof.
   (b) $\dot{x} > \dot{c}_{maxf}$: Thus, Rule 1 holds in states s. By the automata definitions only variable $next$ changes as a result of this action (because $\dot{x} > \dot{c}_{maxf}$). Since $next$ does not appear in Rule 1, it must continue to hold in state s'.
3. π is a time passage step: Either $\dot{x} \leq \dot{c}_{maxf}$
   (a) $\dot{x} \leq \dot{c}_{maxf}$: This case is exactly analogous to the time passage case of the proof of Lemma 5.5. Here, Rule 2 holds in state s and can be shown to also hold in state s'. We omit the proof.
   (b) $\dot{x} > \dot{c}_{maxf}$: 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 $acc' = acc = \ddot{c}_s$ and $send' = send =$ `none` . There are two cases, either Rule 1 holds in s' or Rule 2 does.
      i. $\dot{x}' > \dot{c}_{maxf}$: Rule 1 applies in s' and we must show that it holds. This is guaranteed by Lemma 5.7.

ii. $\dot{x}' \leq \dot{c}_{maxf}$: Rule 2 applies in s'. Note that $acc' = \ddot{c}_s$ is negative, while $(next - now')$ is always positive by Lemma 5.3. Since $\dot{x}' \leq \dot{c}_{maxf}$, we know $\dot{x}' + acc'(next - now') \leq \dot{c}_{maxf}$. Therefore Rule 2 holds in s'.

■

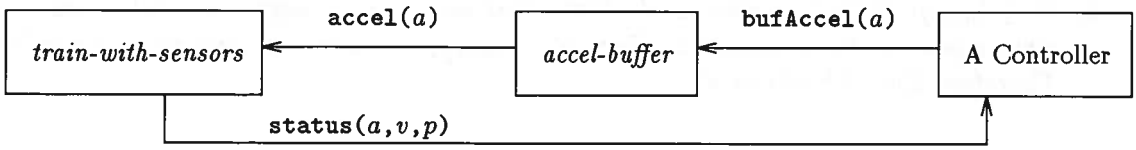

Figure 7: Overview of Feedback with Delay Deceleration Model

6 Feedback and Delay

In this section we combine periodic sensor feedback and command delay. As in Section 4, we introduce delay via a buffer. We make no modification to the *train-with-sensors* automaton and only minor modifications to the *fall-and-zig-zag* automaton. A new buffer called *accel-buffer* is introduced. Figure 7 illustrates the components and their communication.

6.1 Buffer Revisited

The buffer has much the same structure as that of Section 4. The *accel-buffer* is defined in Table 6.1.

Inputs: `bufAccel(`a`)` for $a \in [\ddot{c}_{min} + \ddot{c}_{err}, \ddot{c}_{max}]$
Outputs: `accel(`a`)` for $a \in [\ddot{c}_{min} + \ddot{c}_{err}, \ddot{c}_{max}]$
Variables: $request \in [\ddot{c}_{min} + \ddot{c}_{err}, \ddot{c}_{max}] \cup \{$ `none` $\}$, initially `none`
$violation \in \{$ `true` , `false` $\}$, initially `false`
Actions:
`bufAccel(`a`)`:
Effect: if $request =$ `none` then
$request' = a$
else
$violation' =$ `true`
`accel(`a`)`:
Precondition: $request = a$
Effect: $request' =$ `none`
Tasks: $BUFF = \{$ `accel(`a`)` $\} : [\delta^-, \delta^+]$

Table 6: The *accel-buffer* automaton.

6.2 Controller under Feedback and Delay

A valid *controller-under-feedback-and-delay* is a hybrid I/O automaton that when composed with *train-with-sensors* yields an automaton that satisfies the "non-violation", " liveness", and "safety" properties from Section 4.2.

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. $\delta_{sense} > \delta^+ \geq \delta^- \geq 0$
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. $\dot{c}_{maxf} - \dot{c}_{minf} \geq -\ddot{c}_{min}(\delta_{sense} + \delta^+)$

Most of these constraints are identical to those in Section 5, they are restated here for convenience. Where they differ these constraints supersede those in that section. For convenience, we also continue to assume 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 under Feedback and Delay

We need not completely redefine the *fall-and-zig-zag* controller of Section 5. Instead we define *buffered-fall-and-zig-zag* to be identical to *fall-and-zig-zag* except that we rename its output `accel(`a`)` to `bufAccel(`a`)` and redefine the `status(`a`,`v`,`p`)` action, as follows:

`status(`a`,`v`,`p`)`:
  Effect:
    if $v \leq \dot{c}_{maxf}$ then
      if $\dot{c}_{maxf} < v + a(\delta_{sense} + \delta^+)$ then
        $send' = \frac{\dot{c}_{maxf} - v - a\delta^+}{\delta_{sense}}$
      else
        $send' = \frac{\dot{c}_{maxf} - v - a\delta^-}{\delta_{sense} + \delta^+ - \delta^-}$

The composition of *train-with-sensors*, *accel-buffer*, and *buffered-fall-and-zig-zag* is called *buffered-feedback-system.*

6.5 Proof of Correctness for *buffered-fall-and-zig-zag*

The proof of correctness of the controller requires proofs of the non-violation, liveness, and safety properties. The structure of the proofs is similar to that of Section 5. We prove each property in a separate subsection.

6.5.1 Non-Violation

In this section we prove the non-violation property for *buffered-feedback-system.* It follows as a corollary of two preliminary lemmas. Note, the following lemma is indentical to Lemma 5.4.

Lemma 6.1 *The following is an invariant of* buffered-feedback-system*:*

$$send \neq \text{none} \implies next = now + \delta_{sense}$$

Proof: Trivial induction. ■

Lemma 6.2 *The following is an invariant of* buffered-feedback-system*:*

$$request \neq \text{none} \implies last(BUFF) < next \wedge send = \text{none}$$

Proof: Straightforward induction, uses Lemma 6.1. ■

Corollary 6.3 *In all reachable states of* buffered-one-shot-system *violation* = `false` .

Proof: Violation occurs when *request* $\neq$ `none` and a `bufAccel` action takes place. These actions are only enabled when *send* $\neq$ `none` . Therefore we would like to show that *request* $\neq$ `none` $\vee$ *send* $\neq$ `none` . This is an immediate consequence of Lemma 6.2. ■

6.5.2 Liveness

We do not present all the lemmas results necessary for the liveness result. The structure of the proof is similar to that in Section 5. As in that section, the major result we require is a case-by-case invariant that has as a corollary the lower bound on velocity. The following lemma establishes such a result. It is analogous to Lemma 5.5; it is more complex because of extra cases and uncertainty introduced by the buffer. We have changed the notation slightly to accomodate the more complex conditions.

Lemma 6.4 *Let* $\overline{z}$ *denote* $z - \ddot{c}_{err}$. *The conjunction of the following statements is an invariant of* buffered-feedback-system

1. $send \neq$ `none` $\implies request =$ `none` $\wedge\ \dot{x} \geq \dot{c}_{minf}$
   $\dot{x} + \overline{acc}(\delta^-) + \min(\overline{acc}, \overline{send})(\delta^+ - \delta^-) + \overline{send}(\delta_{sense}) \geq \dot{c}_{minf}$
2. $send =$ `none` $\implies$
   (a) $request =$ `none` $\implies \dot{x} \geq \dot{c}_{minf} \wedge \dot{x} + \overline{acc}(next - now + \delta^+) \geq \dot{c}_{minf}$
   (b) $request \neq$ `none` $\implies$
      i. $now < first(BUFF) \implies \dot{x} \geq \dot{c}_{minf} \wedge$
         $\dot{x} + \overline{acc}(first(BUFF) - now) + \min(\overline{acc}, \overline{request})(\delta^+ - \delta^-) + \overline{request}(\delta_{sense}) \geq \dot{c}_{minf}$
      ii. $now \geq first(BUFF) \implies \dot{x} \geq \dot{c}_{minf} \wedge$
         $\dot{x} + \min(\overline{acc}, \overline{request})(last(OFF) - now) + \overline{request}(\delta_{sense}) \geq \dot{c}_{minf}$

Proof: Long induction. ■

6.5.3 Safety

As in the preceeding section, we give only the major lemma. It is similar to Lemma 5.8.

Lemma 6.5 *The following is an invariant of* buffered-feedback-system*:*

1. $\dot{x} > \dot{c}_{maxf} \implies acc = \ddot{c}_s \wedge send =$ `none` $\wedge \left((x \leq c_f) \implies c_f - x \geq \frac{\dot{c}^2_{maxf} - \dot{x}^2}{2\ddot{c}_s} \right)$
2. $\dot{x} \leq \dot{c}_{maxf} \implies$
   (a) $send \neq$ `none` $\implies request =$ `none` $\wedge$
   $\dot{x} + acc(\delta^-) + \max(acc, send)(\delta^+ - \delta^-) + send(\delta_{sense}) \leq \dot{c}_{maxf}$
   (b) $send =$ `none` $\implies$
      i. $request =$ `none` $\implies \dot{x} + acc(next - now + \delta^+) \leq \dot{c}_{maxf}$
      ii. $request \neq$ `none` $\implies$
         A. $now < first(BUFF) \implies$
            $\dot{x} + acc(first(BUFF) - now) + \max(acc, request)(\delta^+ - \delta^-) + request(\delta_{sense}) \leq \dot{c}_{maxf}$
         B. $now \geq first(BUFF) \implies$
            $\dot{x} + \max(acc, request)(last(OFF) - now) + request(\delta_{sense}) \leq \dot{c}_{maxf}$

Proof: Long induction. ■

7 Conclusions and Future Work

Timed I/O automaton techniques have been extended and applied to a non-trivial hybrid system case-study. Many of the methods and proof techniques used in previous work were useful in this effort. These include: MMT-style notation, invariant assertions, simulations, and automata composition. The case-study begins with a very simple problem and introduces complexity selectively — first communication delay, then periodic feedback, then both. All the cases are highly parameterized and therefore the results general. The proofs are simple and scale well from the simple case to the delay and feedback cases.

This is a work in progress and there are a number of areas for further work. First, we continue to develop complications of the basic case. For example, the model of the braking error is very crude; we would like to develop a vehicle model where the error is more realistic. Second, we would like to prove that the constraints chosen for the parameters are not arbitrary but are instead necessary. Third and finally, we would like to investigate the relationship between our models and those that control theorists use for the same type of problem.

A Formal Hybrid I/O Automaton Model

In this appendix we provide formal definitions for hybrid I/O automata. We state without proof some related theorems that are needed in the text. The *hybrid I/O automaton* model is based on the *timed I/O automaton* model of [1, 2], but includes more explicit treatment of continuous behavior. Specifically, it includes trajectories as primitive objects.

A.1 Automaton Definition

A *hybrid I/O automaton* A consists of:

- a set $states(A)$ of states,
- a nonempty subset $start(A)$ of start states,
- a set $dacts(A)$ of discrete actions, partitioned into *external* and *internal* actions; the external actions are further partitioned into *input* and *output* actions,
- a set $dsteps(A)$ of discrete steps; this is a subset of $states(A) \times dacts(A) \times states(A)$;
- a set $trajs(A)$ of trajectories; each trajectory is a mapping $w : I \to states(A)$, where I is a left-closed interval of $\mathbb{R}^{\geq 0}$ with left endpoint equal to 0.

The components are mostly self-explanatory; the only unusual one is $trajs(A)$, which specifies how states may be associated with times in an interval of real time. We write $s \xrightarrow{\pi}_A s'$ as shorthand for $(s, \pi, s') \in dsteps(A)$.

If $w : I \to states(A)$, then we say that w is an I-trajectory of A. If I consists of of just a single point, we way that I and w are *trivial*. If w is an I-trajectory then define $w.ltime$, the "last time" of w, to be the supremum of I. We define $w.fstate = w(0)$, and if I is right-closed, we also define $w.lstate = w(w.ltime)$. If I is a closed interval, then an I-trajectory w is said to *span* from state s to state s' if $w.w.fstate = s$ and $w.lstate = s'$.

If w is an I-trajectory, where I is right-closed, w' is an I'-trajectory, and if $w.lstate = w'.fstate$, then we define the *concatenation* $w \cdot w'$ to be the least function w'' such that $w''(t) = w(t)$ for all $t \in I$ and $w''(t + w.ltime) = w'(t)$ for all $t \in I'$.

We extend the concatenation operator to a countable sequence of trajectories: if w_i is an I_i trajectory, $1 \leq i < \infty$, 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 $w_1 \cdot w_2 \cdot w_3 \ldots$ to be the least function w such that $w(t + \Sigma_{j<i} w_j.ltime) = w_i(t)$ for all $t \in I_i$.

Let ν be a special symbol denoting (nonzero) time-passage. Let $acts(A)$, the set of *actions*, denote $dacts(A) \cup \{\nu(d) : d \in \mathbb{R}^{>0}\}$. We write $s \xrightarrow{\nu(d)}_A s'$ as shorthand for existence of a nontrivial trajectory that spans from s to s'. Let $steps(A)$ denote $dsteps(A) \cup \{(s, \nu(d), s') : s \xrightarrow{\nu(d)}_A s'\}$. In all these definitions, we sometimes suppress the argument A when no confusion seems likely.

A hybrid I/O automaton must also satisfy the following two axioms:

A1: (Closure under countable concatenation)
If w_i is an I_i trajectory, $1 \leq i < \infty$, where all I_i are right-closed, and $w_i.lstate = w_{i+1}.fstate$ for all i, then $w_1 \cdot w_2 \cdot w_3 \ldots$ is in $trajs(A)$.

A2: (Closure under subinterval)
If $w \in trajs(A)$ is an I-trajectory and J is a left-closed subinterval of I then the function w' given by $w'(t - \min J) = w(t)$ for all $t \in J$ is in $trajs(A)$.

A.2 Timed Executions

In this subsection, we define a notion of "timed execution" for a hybrid I/O automaton. A *timed execution fragment* is a finite or infinite alternating sequence $\alpha = w_0\pi_1 w_1 \pi_2 w_2 \cdots$, where:

1. Each w_i is a trajectory and each π_i is a discrete action.
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 $w_i.lstate \xrightarrow{\pi_{i+1}} w_{i+1}.fstate$.

The trajectories describe the changes of state during the time-passage steps. The last item says that the discrete actions in α span between successive trajectories.

If α is a timed execution fragment then we let $\alpha.ltime$ denote $\Sigma_i w_i.ltime$. We define the initial state of α, $\alpha.fstate$, to be $w_0.fstate$. A *timed execution* is a timed execution fragment α for which the first state, $\alpha.fstate$, is a start state.

In what follows, we will be interested in certain subclasses of the set of timed executions: the *admissible* timed executions and the *finite* timed executions. Thus, we define a timed execution fragment α to be

1. *finite* if α is a finite sequence and the domain of its final trajectory is a right-closed interval,
2. *admissible* if $\alpha.ltime = \infty$.

We use the notation $t\text{-}execs(A)$, $t\text{-}execs^*(A)$ and $t\text{-}execs^\infty(A)$ for the sets of all timed executions, finite timed executions, and admissible timed executions, of A, respectively.

A state of a hybrid I/O automaton is defined to be *reachable* if it is the final state of the final trajectory in some finite timed execution of the automaton.

If α is a finite timed execution fragment with final trajectory w_i, α' is a timed execution fragment with initial trajectory w_0', and $w_i.lstate = w_0'.fstate$ then we define $\alpha \cdot \alpha'$ to be the timed execution fragment obtained by concatenating the sequences α and α', except that the consecutive pair of trajectories w_i and w_0' is replaced by $w_i \cdot w_0'$.

A.3 Timed Traces

In order to describe the problems to be solved by hybrid I/O automata, we require a definition for their visible behavior. We use the notion of *timed traces*. The *timed trace* of any timed execution fragment α, $t\text{-}trace(\alpha)$, is a pair consisting of:

1. the sequence of external discrete events that occur in α, each paired with its time of occurrence, and
2. the final time $\alpha.ltime$.

The *timed traces* of a hybrid I/O automaton A are the timed traces that arise from all the admissible and finite timed executions. The *admissible timed traces* of A are those that arise from all the admissible timed executions, while the *finite timed traces* of A are those that arise from the finite timed executions.

We write $t\text{-}traces(A)$ for the set of timed traces, $t\text{-}traces^*(A)$ for the set of finite timed traces, and $t\text{-}traces^\infty(A)$ for the set of admissible timed traces of A.

If a problem P is formulated as a set of (finite and infinite) sequences of actions paired with times, then a hybrid I/O automaton A is said to *solve* P if all its admissible timed traces are in P. Often, it is natural to express a problem P as the set of admissible timed traces of another hybrid I/O automaton B. Thus, the notion of admissible timed traces induces a preorder on hybrid automata: $A \leq B$ is defined to mean that the set of admissible timed traces of A is a subset of the set of admissible timed traces of B.

A.4 Simulation Relations

In this subsection we present a proof technique for the above mentioned pre-order. A *forward simulation* from hybrid I/O automaton A to hybrid I/O automaton B with the same visible actions is a relation R from $states(A)$ to $states(B)$ satisfying the following conditions:

1. If $s \in start(A)$ then there exists $u \in start(B)$ such that sRu.
2. If $(s, \pi, s') \in dsteps(A)$ and sRu then there exists a timed execution fragment α of B with $\alpha.fstate = u$ and $\alpha.lstate = u'$, such that $s'Ru'$ and $t\text{-}trace(w_s \pi w_{s'}) = t\text{-}trace(\alpha)$, where w_s and $w_{s'}$ are the trivial trajectories with range s and s' respectively.
3. If $w \in trajs(A)$, $s = w.fstate$, and sRu then there exists a timed execution fragment α of B with $\alpha.fstate = u$ and $\alpha.lstate = u'$, such that $s'Ru'$ and $t\text{-}trace(w) = t\text{-}trace(\alpha)$.

The definition says that every step in A has a corresponding execution fragment in B with the same external behavior.

Theorem A.1 *If A and B are hybrid I/O automata with the same external actions and there is a forward simulation from A to B, then*

1. $t\text{-}traces(A) \subseteq t\text{-}traces(B)$.
2. $t\text{-}traces^*(A) \subseteq t\text{-}traces^*(B)$.
3. $t\text{-}traces^\infty(A) \subseteq t\text{-}traces^\infty(B)$.

A.5 Composition

It is easy to extend the usual composition definition for timed automata to hybrid I/O automata, as follows: Let A and B be hybrid I/O automata satisfying the following *compatibility* conditions:

1. A and B have no output actions in common.
2. No internal action of A is an action of B, and vice versa.

Then the *composition* of A and B, written as $A \times B$, is the hybrid I/O automaton defined as follows.

- $states(A \times B) = states(A) \times states(B)$;
- $start(A \times B) = start(A) \times start(B)$;
- $dacts(A \times B) = dacts(A) \cup dacts(B)$; a discrete action is *external* in $A \times B$ exactly if it is external in either A or B, and likewise for *internal* actions; an external action of $A \times B$ is an *output* in $A \times B$ exactly if it is an output in either A or B, and is an *input* otherwise;
- If $\pi \in dacts(A \times B)$ then $(s_A, s_B) \longrightarrow^{\pi}_{A \times B} (s'_A, s'_B)$ exactly if
  1. $s_A \longrightarrow^{\pi}_A s'_A$ if $\pi \in dacts(A)$, else $s_A = s'_A$, and
  2. $s_B \longrightarrow^{\pi}_B s'_B$ if $\pi \in dacts(B)$, else $s_B = s'_B$;
- If $w : I \to states(A \times B)$ then $w \in trajs(A \times B)$ exactly if the following hold:
  1. $w_A \in trajs(A)$, where $w_A : I \to states(A)$ is defined by $w_A(t) = w(t)|states(A)$.
  2. $w_B \in trajs(B)$, where $w_B : I \to states(B)$ is defined by $w_B(t) = w(t)|states(B)$.

Theorem A.2 *$A \times B$ is a hybrid I/O automaton.*

The composition operator is also associative and commutative, up to isomorphism, so can be extended to a k-ary operator for any finite k.

Typically the states of automata are specified as the possible assignments to a set of variables. By convention, when we compose hybrid I/O automata which have the usual *now* variable, we elide the distinction between the *now* variables in the composition. This is well-defined only when the *now* variable has its customary meaning.

Let A be the composition of a finite collection of hybrid I/O automata A_i, $i \in I$. If α is a timed execution of A, then for each $i \in I$, we can define $\alpha|A_i$, the projection of α on i by projecting all trajectories onto the state of A_i, keeping only the actions in $dacts(A_i)$, and merging adjacent trajectories. Note that axiom [A1] is required, to ensure that the merged trajectories are actually trajectories. Also, if β is a timed trace of A, then for each $i \in I$, we can define $\beta|A_i$ by projecting onto the (discrete) external actions of A_i (and keeping the same final time).

We give a theorem that allows projection of timed executions, finite timed executions, and admissible timed executions onto components.

Theorem A.3 *Let A be the composition of a finite collection of hybrid I/O automata A_i, $i \in I$.*

1. *If α is a timed (resp., finite timed / admissible timed) execution of A, then for each $i \in I$, $\alpha|A_i$ is a timed (resp., finite timed / admissible timed) execution of A_i.*
2. *If β is a timed (resp., finite timed / admissible timed) trace of A, then for each $i \in I$, $\beta|A_i$ is a timed (resp., finite timed / admissible timed) trace of A_i.*
3. *If α is a timed execution of A, then for each $i \in I$, $t\text{-}trace(\alpha|A_i) = t\text{-}trace(\alpha)|A_i$.*

Theorem A.4 *Let A be a hybrid I/O automaton. Let B and C be hybrid I/O automata with the same external signatures and both compatible with A.*

1. *If $t\text{-}traces^*(B) \subseteq t\text{-}traces^*(C)$ then $t\text{-}execs^*(A \times B)|A \subseteq t\text{-}execs^*(A \times C)|A$.*
2. *If $t\text{-}traces^\infty(B) \subseteq t\text{-}traces^\infty(C)$ then $t\text{-}execs^\infty(A \times B)|A \subseteq t\text{-}execs^\infty(A \times C)|A$.*

B Formal MMT Automaton Model

This section is based on a similar exposition in [6]. We give a formal definition of an MMT-specification and of the corresponding hybrid I/O automaton.

An *MMT-specification* A consists of six components:

- a set $states(A)$ of states,
- a non-empty set $start(A)$ of start states,
- a set $acts(A)$ of actions,
- a set $steps(A) \subseteq states(A) \times acts(A) \times states(A)$,
- a collection $tasks(A)$ of disjoint subsets of $acts(A)$,
- a function $bounds(A) : tasks(A) \to \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$.

The tasks are sets of actions which share the same timing behavior. The bounds function specifies that behavior by giving a lower and upper time bound for execution of each task. For convenience we write $b_l(C_i)$ and $b_u(C_i)$ for the lower and upper time bounds of task C_i, respectively. An action a is *enabled* in state s when $(s, a, s') \in steps(A)$ for some s. A task C_i is enabled in a state if 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 A_{MMT} be an MMT-specification. Then $time(A_MMT)$ is the hybrid I/O automaton A with the following components:

$states(A)$ are the possible assignments to the following variables: $basic \in states(A_{MMT})$; $now \in \mathbb{R}^{\geq 0}$; and $first(C_i) \in \mathbb{R}^{\infty}$ and $last(C_i) \in \mathbb{R}^{\infty}$ for all $C_i \in tasks(A_{MMT})$.

$start(A)$ are all the elements s of $states(A)$ where $s.basic \in start(A_{MMT})$, $s.now = 0$, and for each $C_i \in tasks(A)$ if C_i is enabled in $s.basic$ then $first(C_i) = b_l(C_i)$ and $last(C_i) = b_u(C_i)$; otherwise, $first(C_i) = 0$ and $last(C_i) = \infty$.

$dacts(A)$ are $acts(A_{MMT})$.

$dsteps(A)$ are all (s, a, s') where:

1. $s'.now = s.now$
2. $(s.basic, a, s'.basic) \in steps(A_{MMT})$
3. for each $C_i \in tasks(A_{MMT})$
   (a) If $a \in C_i$, then $s.first(C_i) \leq s.now$.
   (b) If C_i is enabled in both $s.basic$ and $s'.basic$, and $a \notin C_i$, then $s'.first(C_i) = s.first(C_i)$ and $s'.last(C_i) = s.last(C_i)$.
   (c) If C_i is enabled in $s'.basic$ and either $a \in C_i$ or C_i is not enabled in $s.basic$, then $s'.first(C_i) = s'.now + b_l(C_i)$ and $s'.last(C_i) = s'.now + b_u(C_i)$.
   (d) If C_i is not enabled in $s'.basic$ then $s'.first(C_i) = 0$ and $s'.last(C_i) = \infty$.

$trajs(A)$ are the set of I-trajectories w where for all $t \in I$:

1. $w(t).now = w(0).now + t$
2. $w(t).basic = w(0).basic$
3. for all $C_i \in tasks(A_{MMT})$
   (a) $w(t).now \leq w(0).last(C_i)$
   (b) $w(t).first(C_i) = w(0).first(C_i)$
   (c) $w(t).last(C_i) = w(0).last(C_i)$

In practice, we elide the distinction between the MMT-specification and its corresponding hybrid I/O automaton. The stylized nature of MMT-specifications yields some useful properties:

Theorem B.1 *If A_{MMT} is an MMT-specification and $A = time(A_{MMT})$, then in all reachable states s of A and for all $C_i \in tasks(A_{MMT})$ 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 $s.basic$ then* $0 \leq last(C_i) - now \leq b_u(C_i)$

References

[1] Nancy Lynch and Frits Vaandrager. Forward and backward simulations for timing-based systems. In *Proceedings of REX Workshop "Real-Time: Theory in Practice"*, volume 600 of *Lecture Notes in Computer Science*, pages 397–446, Mook, The Netherlands, June 1991. Springer-Verlag.

[2] Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part II: Timing-based systems. Technical Memo MIT/LCS/TM-487.b, Laboratory for Computer Science, Massachusetts Institute Technology, Cambridge, MA, 02139, April 1993.

[3] 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.

[4] Nancy Lynch. Simulation techniques for proving properties of real-time systems. In deRoever Bakker and Rozenberg, editors, *A Decade of Concurrency: Reflections and Perspectives*, volume 803 of *Lecture Notes in Computer Science*, pages 375–424, REX School/Symposium, Noordwijkerhout, the Netherlands, June 1993. Springer-Verlag.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] Richard A. Brown, Jacob Aizikowitz, Thomas C. Bressoud, Tony Lekas, and Fred Schneider. The trainset railroad simulation. Technical Report TR 93-1329, Cornell University, Department of Computer Science, February 1993.

[13] 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.

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

[15] 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.

[16] Zohar Manna and Amir Pnueli. Verifying hybrid systems. In Robert L. Grossman et al., editor, *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 4–35. Springer-Verlag, 1992.

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

[18] Nancy Lynch and Frits Vaandrager. Forward and backward simulations – Part II: Timing-based systems. Submitted for publication. Also, [2].

[19] Z. Manna and A.Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer-Verlag, 1992.

[20] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1995. To appear.

[21] 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.

[22] 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.

[23] 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.

[24] N. Lynch and M. Tuttle. An introduction to Input/Output automata. Technical Memo MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute Technology, Cambridge, MA, 02139, November 1988.

[25] N. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. Technical Report MIT/LCS/TR-387, Laboratory for Computer Science, Massachusetts Institute Technology, Cambridge, MA, 02139, April 1987.