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

Nancy Lynch

February 15, 1996

1 Introduction

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

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

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

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

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

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

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

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

2 Hybrid Input/Output Automata

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

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

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

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

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

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

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

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

The important fact about a simulation is:

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

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

3 Mathematical Preliminaries

3.1 Assumptions About the Constants

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

3.2 Some Useful Functions

3.2.1 Function f

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

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

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

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

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

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

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

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

3.2.2 Function g

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

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

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

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

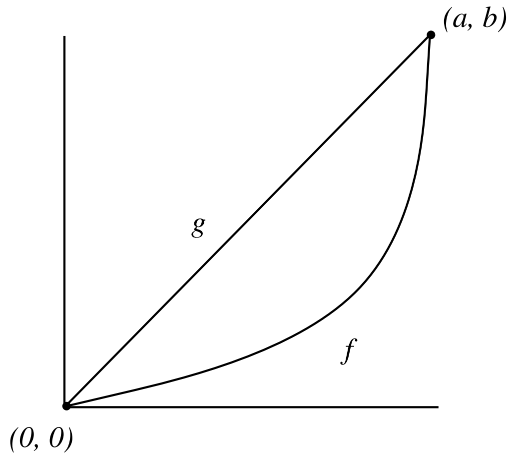

Figure 1: Functions f and g.

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

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

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

3.2.3 Function f_1

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

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

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

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

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

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

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

3.2.4 Function h

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

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

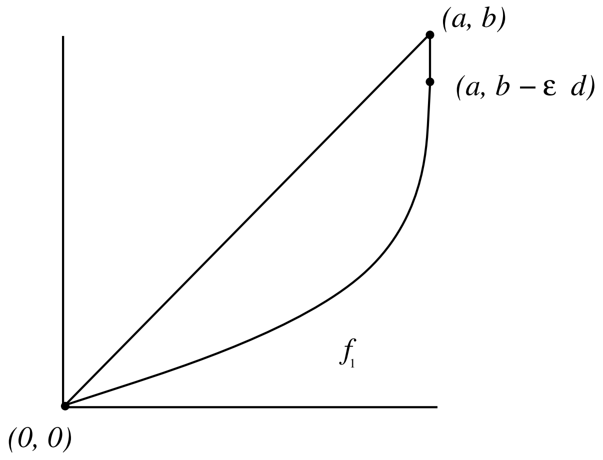

Figure 2: Function f_1.

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

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

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

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

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

4 High Level Specification V

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

4.1 Overview

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

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

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

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

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

4.2 Formal Description

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

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

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

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

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

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

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

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

We give a trivial invariant of V:

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

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

Proof: The proof (as usual for invariants of HIOA's) is by induction on the length, that is, the number of trajectories and discrete steps, in a finite hybrid execution that leads to the state in question. Here (as usual for such proofs), we must show three things: that the property is true in every initial state, that it is preserved by every discrete step, and that it is preserved by every right-closed trajectory. (Note that we need only consider right-closed trajectories, and that we need only show that the property holds in the last state of the trajectory, given that it holds in the first state. We do not need to show anything about the intermediate states in the trajectory.)

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

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

5 Derivative Automaton D

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

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

5.1 Formal Description

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

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

The discrete actions of D are:

Input:	Internal:
none	*reset*
Output:	
none	

D has the following variables:

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

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

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

The trajectories of D are all the mappings w from left-closed subintervals I of $[0, a]$ to states of D such that:

1. $\dot{v}$ is an integrable function in w.[3]
2. For all $t \in I$, the following conditions hold in state $w(t)$.
   (a) $now = w(0).now + t$.
   (b) If $now \neq a$ then $acc = \frac{b-v}{a-now}$. (Otherwise, acc is arbitrary).
   (c) If $now \neq a$ then $\dot{v} \in [acc - \epsilon, acc]$.
   (d) $v = w(0).v + \int_0^t w(x).\dot{v} dx$.

In D, acc points directly at the "goal" (a, b), but $\dot{v}$ reflects an uncertainty of ϵ. The quantity v is simply derived from $\dot{v}$, using integration.

5.2 Some Properties of D

Lemma 5.1 *Let w be any trajectory of D. Then v is a continuous function in w.*[4]

There are some obvious invariants.

Lemma 5.2 *In every reachable state of D, the following are true.*

1. *If $now \neq a$ then $acc = \frac{b-v}{a-now}$.*
2. *If $now \neq a$ then $\dot{v} \in [acc - \epsilon, acc]$.*

Proof: These follow easily from the definition of D. ∎

The following invariant is a little less obvious, but is an easy consequence of Lemma 5.2. The functions f and g used in this lemma are as defined in Section 3.2.

Lemma 5.3 *In every reachable state of D in which $now \neq a$, the following are true.*

1. $\frac{v-f(now)}{a-now} \geq \dot{f}(now) - \dot{v}$.
2. $\frac{g(now)-v}{a-now} \geq \dot{v} - \dot{g}(now)$.

Proof:

1. By definition of f, we have that:

$$\dot{f}(now) = \frac{b - f(now)}{a - now} - \epsilon.$$

[3] More precisely, this means that $w(t).\dot{v}$ is an integrable function of t, where t ranges over the interval I.
[4] This means continuous in the time argument of w.

By Lemma 5.2, we have that:

$$\dot{v} \geq acc - \epsilon = \frac{b - v}{a - now} - \epsilon.$$

Therefore,

$$\dot{f}(now) - \dot{v} \leq \frac{b - f(now)}{a - now} - \epsilon - (\frac{b - v}{a - now} - \epsilon) = \frac{v - f(now)}{a - now}.$$

This is as needed.

2. By definition of g, we have that:

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

By Lemma 5.2, we have that:

$$\dot{v} \leq acc = \frac{b - v}{a - now}.$$

Therefore,

$$\dot{v} - \dot{g}(now) \leq \frac{b - v}{a - now} - \frac{b - g(now)}{a - now} = \frac{g(now) - v}{a - now}.$$

This is as needed.

■

Also, there are limitations on the rate of change of the velocity in D (for contrast, recall there were no such limitations in V).

Lemma 5.4 *Let w be any (right-closed or right-open) trajectory of D whose now values do not include a, and that starts from a reachable state of D. Then:*

1. *The ratio $\frac{v - f(now)}{a - now}$ is monotone nondecreasing in w.*[5]

2. *The ratio $\frac{g(now) - v}{a - now}$ is monotone nondecreasing in w.*

This says that v cannot increase too slowly – its distance from f, weighted by the time remaining, cannot decrease. Likewise, v cannot increase too fast – its distance from g, weighted by the time remaining, cannot decrease.

Proof: In each case, it suffices to show that the first derivative of the ratio is always nonnegative.

1. The first derivative of the ratio is:

$$\frac{(a - now)(\dot{v} - \dot{f}(now)) - (v - f(now))(-1)}{(a - now)^2}$$

$$= \frac{(a - now)(\dot{v} - \dot{f}(now)) + (v - f(now))}{(a - now)^2}.$$

(Here we are using the fact that $\dot{v}$ is the derivative of v – this is justified formally by the integral definition of the variable v.)

[5] This means monotone nondecreasing in the time argument of w.

Since the denominator is always positive, it suffices to show that:

$$= (a - now)(\dot{v} - \dot{f}(now)) + (v - f(now)) \geq 0$$

in all states of w. This is equivalent to saying that:

$$\frac{v - f(now)}{a - now} \geq \dot{f}(now) - \dot{v},$$

in all states of w. But this follows immediately from Lemma 5.3 (using the fact that w starts in a reachable state of D, so all its states are reachable).

2. The derivative of the ratio is:

$$\frac{(a - now)(\dot{g}(now) - \dot{v}) - (g(now) - v)(-1)}{(a - now)^2}$$

$$= \frac{(a - now)(\dot{g}(now) - \dot{v}) + (g(now) - v)}{(a - now)^2}.$$

Since the denominator is always positive, it suffices to show that:

$$(a - now)(\dot{g}(now) - \dot{v}) + (g(now) - v) \geq 0$$

in all states of w. But this is equivalent to saying that:

$$\frac{g(now) - v}{a - now} \geq \dot{v} - \dot{g}(now)$$

in all states of w. This follows from Lemma 5.3.

■

5.3 D Implements V

The main result that we want to show about D is the following:

Theorem 5.5 *If α_D is a hybrid execution of D, then there is a hybrid execution α_V of V having the same hybrid trace.*

Note that the hybrid trace of each of V and D includes just the *now* and v values. Theorem 5.5 implies that the changes in *now* and v that are exhibited by D are allowed, according to the constraints expressed by V. The correspondence does not mention the implementation variables *acc* and $\dot{v}$

We prove Theorem 5.5 using a simulation, as defined informally in Section 2. We define a relation *fsim* from states of D to states of V as follows. If s_D is a state of D and s_V is a state of V, then we say that $(s_D, s_V) \in fsim$ provided that the following hold.

1. $s_D.now = s_V.now$.
2. $s_D.v = s_V.v$.

We show:

Lemma 5.6 *fsim is a simulation from D to V.*

Proof: We show the three conditions in the definition of a simulation. The start condition is straightforward: If s_D is any start state of D and s_V is the unique start state of V, then both states have $now = 0$ and $v = 0$. It follows that $(s_D, s_V) \in fsim$.

Next, we consider discrete steps. Suppose that (s_D, π, s'_D) is any discrete step of D, and that $(s_D, s_V) \in fsim$. Then let the hybrid execution fragment corresponding to this step consist of the trivial trajectory containing exactly one state and no steps. Then both the discrete step and the corresponding fragment have the same hybrid trace, consisting of the values of now and v that appear in s_D. It suffices to show that $(s'_D, s_V) \in fsim$. But this is immediate, because π (a *reset* or e action) does not modify either now or v.

Now we consider trajectories. Suppose that w_D is an I-trajectory w_D of D, where I is right-closed, and suppose that the first state, s_D, of w_D is reachable in D. Suppose that s_V is a reachable state of V such that $(s_D, s_V) \in fsim$. Then let the corresponding hybrid execution fragment of V consists of a single trajectory w_V, where $w_V(t).now = w_D(t).now$ and $w_V(t).v = w_D(t).v$ for all t in the domain of I. It is obvious that the two trajectories have the same hybrid trace. The only interesting thing to show is that w_V is in fact a trajectory of V. By the definition of a trajectory of V, what we must show is that

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

(We must verify these conditions throughout the trajectory, not just at the beginning and end.) The first condition follows immediately from the same condition for w_D and the definition of w_V in terms of w_D. The second condition has two parts, a lower bound and an upper bound.

For the lower bound, since s_V is a reachable state of V, Lemma 4.1 implies that, in s_V, $v \geq f(now)$. By Lemma 5.4 and the definition of w_V in terms of w_D, we know that the ratio $\frac{v-f(now)}{a-now}$ is monotone nondecreasing in w_V, except possibly at the right endpoint of w_V if $now = a$ there. It follows that $v \geq f(now)$ throughout w_V, except possibly at the right endpoint in case $now = a$ there. But since $f(now)$ and v are continuous functions of the time argument of w_V, this inequality must hold at the right endpoint as well.

The upper bound argument is analogous. Since s_V is reachable, Lemma 4.1 implies that, in s_V, $v \leq g(now)$. By Lemma 5.4 and the definition of w_V in terms of w_D, we know that the ratio $\frac{g(now)-v}{a-now}$ is monotone nondecreasing in w_V, except possibly at the right endpoint of w_V if $now = a$ there. It follows that $v \leq g(now)$ throughout w_V, exept possibly at the right endpoint in case $now = a$ there. But since $g(now)$ and v are continuous functions of the time argument of w_V, this inequality must hold at the right endpoint as well. ■

Proof: (of Theorem 5.5)

By Lemma 5.6 and Theorem 2.1. ■

Note that the correspondence between D and V is only one-way. It says, roughly speaking, that everything that D does is allowed by V. It does not say that D has to exhibit *all* the possibilities that are allowed by V. For example, extremely fast increases in v that cannot be achieved by accelerations in the allowed ranges, but that keep v within the allowed envelope, are permitted by V, but do not actually occur in D. Also, note that D performs some activities – here, changes to acc and $\dot{v}$ – that are not explicitly represented in V.

Although Theorem 5.5 is very simple, it does demonstrate, at least in a small way, how one can carry out a correctness proof using invariants and simulations, integrating discrete and continuous reasoning, and coping with some uncertainty.

5.4 An Approximate Result

The lower bound function f is essentially defined as the solution of a differential equation that is extracted from the definition of the trajectories of D. In this case, the differential equation is easy to solve. But what if it were not so easy? In this case, the same methods could still be used, but now the lower bound produced might be a loose bound rather than an exact bound.

For example, suppose that instead of trying to prove a lower bound of f, we only tried to prove a lower bound of h, where h is the function defined in Section 3.2.4. Showing that h is a lower bound essentially requires redefining V to use h instead of f. Proving the simulation now rests on the fact, stated in Section 3.2.4, that

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

for all $t \in [0, a)$. Using this fact, it is easy to obtain the analog to part 1 of Lemma 5.3 for h: that in every reachable state of D,

$$\frac{v - h(now)}{a - now} \geq \dot{h}(now) - \dot{v}.$$

This fact follows as in the proof of part 1 of Lemma 5.3 (but using the inequality above at one step instead of an equality as before). Next, we can prove the analog to part 1 of Lemma 5.4 for h: that the ratio $\frac{v-h(now)}{a-now}$ is monotone nondecreasing in w. This is what is needed to complete the analog to the proof of Lemma 5.6.

6 Modifications to V and D to Incorporate Periodic Feedback

The discussions and results in Sections 4 and 5 have dealt with hypothetical systems with continuous control. But recall from the introduction that in the actual implementation in which we are interested, the sampling outputs and control signals are not continuous but periodic, at intervals of d. It turns out that the abstract automata D and V do not quite provide accurate models of the actual implementation. However, they can be modified easily so that they do.

We believe that providing accurate models for the handling of uncertainties is important. It is not sufficient to give a careful analysis of a situation without uncertainty, then argue informally about the variations in behavior that are introduced by uncertainties. Handling uncertainties correctly requires considering them appropriately at all levels of abstraction.

6.1 Modified High Level Specification V_1

First, we modify V only a tiny bit to get V_1, by changing the lower bound f to the function f_1 defined in Section 3.2.3. The upper bound g remains the same as before. (Of course, we could have written the original V with parameters, so that the modifications in this section would just amount to different parameter settings.)

This modification makes the region of allowable values for v bigger by making the lower bound function smaller. The particular way that we make it smaller amount to simply replacing the "goal" of (a, b) in V with the goal of $(a, b - \epsilon d)$ in V_1, for the lower bound function only. Thus, the value of v at time a will be in the range $[b - \epsilon d, b]$, instead of always being exactly b.

It was not obvious to us at first that the high-level effect of the sampling delays is just this simple change of goal point; we discovered this only through detailed analysis of the behavior of the discrete-sampling system. We do not expect to use any general rule for determining the high-level effect of uncertainties; indeed, we expect that this will usually require some serious work, using results of robust control theory. It is important that the high-level effect of uncertainties be described completely accurately, though the bounds need not be as tight as possible.

6.2 Modified Derivative Automaton D_1

Now we modify D to get D_1, again by modifying the lower bound requirement. Here we do this by introducing uncertainty into acc, allowing it to "point" anywhere from (a, b) (where it points in D) to $(a, b - \epsilon d)$. We still have the same uncertainty as before in $\dot{v}$. Thus, D_1 expresses two different types of uncertainty. We can think of the uncertainty in $\dot{v}$ as representing propulsion system uncertainty and the uncertainty in acc as encompassing the sampling delays.

The modifications are as follows. The states and start states of D_1 are the same as those of D, except for the following changes: The initial value of acc is any value in the interval $[\frac{b-\epsilon d}{a}, \frac{b}{a}]$, and the initial value of $\dot{v}$ is any value in the interval $[acc - \epsilon, acc]$. The *reset* action now changes slightly, to allow changes in acc as well as $\dot{v}$. These changes keep acc and $\dot{v}$ within the desired ranges.

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

The trajectories of D_1 are all the mappings w from left-closed subintervals I of $[0, a]$ to states of D_1 such that:

1. $\dot{v}$ is an integrable function in w.
2. For all $t \in I$, the following conditions hold in state $w(t)$.
   (a) $now = w(0).now + t$.
   (b) If $now \neq a$ then $acc \in [\frac{b-\epsilon d-v}{a-now}, \frac{b-v}{a-now}]$.
   (c) If $now \neq a$ then $\dot{v} \in [acc - \epsilon, acc]$.
   (d) $v = w(0).v + \int_0^t w(x).\dot{v} dx$.

(Again, we could have written the original D with parameters, so that the modifications in this section would amount to different parameter setting.)

6.3 Modified Correctness Proof

Our claim now is that the arguments that worked to show that D implements V can be modified slightly (and systematically) to show that D_1 implements V_1. We give the modified result statements.

Lemma 6.1 *Let w be any trajectory of D_1. Then v is a continuous function in w.*

Lemma 6.2 *In every reachable state of D_1, the following are true.*

1. *If $now \neq a$ then $acc \in [\frac{b-\epsilon d-v}{a-now}, \frac{b-v}{a-now}]$.*
2. *If $now \neq a$ then $\dot{v} \in [acc - \epsilon, acc]$.*

Lemma 6.3 *In every reachable state of D_1 in which $now \neq a$, the following are true.*

1. $\frac{v - f_1(now)}{a - now} \geq \dot{f}_1(now) - \dot{v}$.

2. $\frac{g(now) - v}{a - now} \geq \dot{v} - \dot{g}(now)$.

Proof: We only prove part 1; part 2 is unchanged from the corresponding proof for D. By definition of f_1, we have that:

$$\dot{f}_1(now) = \frac{b - \epsilon d - f_1(now)}{a - now} - \epsilon.$$

By Lemma 6.2, we have that:

$$\dot{v} \geq acc - \epsilon \geq \frac{b - \epsilon d - v}{a - now} - \epsilon.$$

Therefore,

$$\dot{f}_1(now) - \dot{v} \leq \frac{b - \epsilon d - f_1(now)}{a - now} - \epsilon - (\frac{b - \epsilon d - v}{a - now} - \epsilon) = \frac{v - f_1(now)}{a - now}.$$

This is as needed. ■

Lemma 6.4 *Let w be any trajectory of D_1 whose now values do not include a, and that starts from a reachable state of D_1. Then:*

1. *The ratio $\frac{v - f_1(now)}{a - now}$ is monotone nondecreasing in w.*

2. *The ratio $\frac{g(now) - v}{a - now}$ is monotone nondecreasing in w.*

Now define the relation $fsim_1$ from states of D_1 to states of V_1 as follows. If s_{D_1} is a state of D_1 and s_{V_1} is a state of V_1, then we say that $(s_{D_1}, s_{V_1}) \in fsim_1$ provided that the following hold.

1. $s_{D_1}.now = s_{V_1}.now$.
2. $s_{D_1}.v = s_{V_1}.v$.

This definition is essentially the same as that for *fsim*, from D to V.

Lemma 6.5 *$fsim_1$ is a simulation from D_1 to V_1.*

Proof: Similar to the proof of Lemma 5.6. ■

Theorem 6.6 *If α_{D_1} is a hybrid execution of D_1, then there is a hybrid execution α_{V_1} of V_1 having the same hybrid trace.*

Theorem 6.6 says that the changes in *now* and v that are exhibited by D_1 are allowed by V_1.

Note that the modifications we did to include this uncertainty are quite simple and systematic. A good general strategy for constructing proofs for implementations involving uncertainty is to first carry out the development without the uncertainty, then try to incorporate the uncertainty later, by making simple modifications throughout.

7 The Implementation *Impl*

Now we are (finally) ready to describe the actual implementation in which we are interested. This one consists of two components, a *Vehicle* and a *Controller*, interacting by discrete actions. Each component is, formally, an HIOA, and the combination is a composition of HIOA's, interacting via discrete actions only, with the common actions hidden.

7.1 *Vehicle*

The *Vehicle* HIOA represents the motion of the vehicle, including its velocity and acceleration. It reports the velocity (accurately, we assume) every d units of time, starting at time d. It is capable of receiving control signals that set an *acc* variable, representing the desired acceleration. However, the actual acceleration can be slightly less than this – within amount ϵ.

The actions are:

Input:
- $accel(c)$, $c \in \mathsf{R}$

Output:
- $sample(u)$, $u \in \mathsf{R}$

Internal:
- *none*

The variables are the same as those of D_1, with the addition of an internal "deadline variable" *last-sample*. This deadline variable just keeps track of the next (absolute) time at which a *sample* output is scheduled to occur. Also, the initialization of *acc* is more constrained than it is in D_1, reflecting the assumption that the correct acceleration is in effect at the beginning. We can think of the system as if we initialized it with an initial sample output and control signal.

Input:
- *none*

Output:
- $now \in [0, a]$, initially 0
- $v \in \mathsf{R}$, initially 0

Internal:
- $acc \in \mathsf{R}$, initially $\frac{b}{a}$
- $\dot{v} \in \mathsf{R}$, initially any value in $[\frac{b}{a} - \epsilon, \frac{b}{a}]$
- $last\text{-}sample \in \mathsf{R}^{\geq 0}$, initially d

The non-e discrete steps are:

$accel(c)$
- Effect:
  - $acc := c$
  - $\dot{v} :=$ any value $\in [acc - \epsilon, acc]$

$sample(u)$
- Precondition:
  - $now = last\text{-}sample$
  - $u = v$
- Effect:
  - $last\text{-}sample := now + d$

Thus, an *accel* step just sets the *acc* control variable, and resets the actual acceleration $\dot{v}$ accordingly. A *sample* step just announces the current velocity – the only information needed by the controller component. It does so exactly at the time scheduled in *last-sample*. Then it reschedules the sampling time to be exactly d in the future.

The trajectories of *Vehicle* are all the mappings w from left-closed subintervals I of $[0, a]$ to states of *Vehicle* such that:

1. *acc* and *last-sample* are unchanged in w.
2. $\dot{v}$ is an integrable function in w.
3. For all $t \in I$, the following conditions hold in state $w(t)$.
   (a) $now = w(0).now + t$.
   (b) $now \leq last\text{-}sample$.
   (c) $\dot{v} \in [acc - \epsilon, acc]$.
   (d) $v = w(0).v + \int_0^t w(x).\dot{v}dx$.

These trajectories are quite similar to those that are permitted in D_1. The most important difference is that *acc* is now not permitted to change during trajectories; instead, it changes only as a result of discrete inputs (from the controller, presumably). However, $\dot{v}$ can change, as long as it stays within the required bounds. There is also a condition that prevents time from passing beyond the *last-sample* deadline. The following invariants are straightforward to prove.

Lemma 7.1 *In every reachable state of Vehicle, the following are true.*

1. $\dot{v} \in [acc - \epsilon, acc]$.
2. $last\text{-}sample \in [now, now + d]$.

7.2 *Controller*

The *Controller* HIOA represents the controller that decides on the desirable acceleration, i.e., the value that should be placed into *Vehicle*'s variable *acc*. It receives reports from the *Vehicle* of its current velocity v, and uses each such report to calculate a desired new acceleration. It sends this, before any further time passage, to the *Vehicle* in an *accel* action.

The external actions of the *Controller* form the "mirror image" of those of the *Vehicle*:

Input:
  $sample(u)$, $u \in \mathsf{R}$
Output:
  $accel(c)$, $c \in \mathsf{R}$

Internal:
  none

The variables are:

Input:
  none
Output:
  none

Internal:
  $now \in [0, a]$, initially 0
  $sampled\text{-}vel \in \mathsf{R}$, initially 0
  $last\text{-}accel \in \mathsf{R}^{\geq 0} \cup \{\infty\}$, initially ∞

Here, *sampled-vel* is intended to hold the sampled velocity, when the *Controller* receives a report about it. The *last-accel* variable is another deadline variable, intended to keep track of the next scheduled (absolute) time for an *accel* signal. Initially (until the *Controller* receives some velocity report), there is no scheduled signal, so $last\text{-}accel = \infty$.

The non-e discrete steps are:

$sample(u)$
  Effect:
    $sampled\text{-}vel := u$
    $last\text{-}accel := now$

$accel(c)$
  Precondition:
    $last\text{-}accel = now$
    $now \neq a$
    $c = \frac{b - sampled\text{-}vel}{a - now}$
  Effect:
    $last\text{-}accel := \infty$

The *sample* action just records the reported velocity, and schedules an *accel* action to happen before any further real time elapses. (We could alternatively have modelled a system in which there is some bounded delay before the *accel* action occurs.) The *accel* action recalculates the desired velocity, using the same formula as in D – pointing at the desired goal (a, b) – but this time, the calculation is based on the sampled velocity instead of the actual velocity. After the *accel* action, no further *accel* is scheduled, until a new *sample* occurs.

The trajectories of *Controller* are trivial – time just passes up to any time that does not exceed any current deadline. There is no interesting continuous behavior to be modelled. That is, the trajectories are all the mappings w from left-closed subintervals I of $[0, a]$ to states of *Controller* such that:

1. *sampled-vel* and *last-accel* are unchanged in w.
2. For all $t \in I$, the following conditions hold in state $w(t)$.
   (a) $now = w(0).now + t$.
   (b) $now \leq last\text{-}accel$.

7.3 *Impl*

The complete implementation *Impl* is the composition of the two HIOA's *Vehicle* and *Controller*, identifying the *sample* and *accel* actions, and then hiding those actions (making them internal).

We close this section with some properties of *Impl*. The first lemma gives simple invariants about *last-accel*. It says that *last-accel* is only used to schedule an event immediately, and that when it is being used, the recorded and actual velocities are identical.

Lemma 7.2 *In every reachable state of Impl, the following are true.*

1. $\mathit{last\text{-}accel} \in \{now, \infty\}$.
2. *If* $\mathit{last\text{-}accel} = now$ *then* $v = \mathit{sampled\text{-}vel}$.

The next lemma is a key lemma for the simulation proof. It expresses bounds on the *acc* variable, no matter where the reference point is in a sampling interval. The *acc* variable is set accurately initially, and at each sampling time. But in between, the accuracy of the value of *acc* can degrade. Lemma 7.3 gives appropriate guarantees at all times, even within the sampling intervals. Some general statement of this sort is needed for the inductive proof of the simulation of D_1 by *Impl*.

In the statement of Lemma 7.3, the assumption that $\mathit{last\text{-}accel} = \infty$ is used to avoid the case where the implementation automaton is in the middle of processing a new sampling output.

Lemma 7.3 *In every reachable state of Impl, the following are true.*

1. *If* $now \neq a$ *and* $\mathit{last\text{-}accel} = \infty$ *then* $acc \geq \frac{b-\epsilon(now+d-\mathit{last\text{-}sample})-v}{a-now}$.
2. *If* $now \neq a$ *then* $acc \leq \frac{b-v}{a-now}$.

Notice that the lower bound expressed in case 1 varies during each sampling interval. At the beginning of the interval, we have $now + d = \mathit{last\text{-}sample}$, so the bound simplifies to $\frac{b-v}{a-now}$. At the other extreme, at the end of the interval, we have $now = \mathit{last\text{-}sample}$, and the bound simplifies to $\frac{b-\epsilon d-v}{a-now}$. The complete statement fills in guarantees for the intermediate points as well.

Proof:

1. The lower bound is proved by induction on the length of a hybrid execution, as usual. The lower bound claim is true initially, since initially $acc = \frac{b}{a}$, $now = 0$, $\mathit{last\text{-}sample} = d$, and $v = 0$.

   Now consider a discrete step starting from a reachable state. A *sample* step makes $\mathit{last\text{-}accel} = \infty$, which makes the claim vacuously true. On the other hand, an *accel* step explicitly sets *acc* to $\frac{b-\mathit{sampled\text{-}vel}}{a-now}$, which is equal to $\frac{b-v}{a-now}$ by Lemma 7.2, which suffices to show the inequality. (This uses the fact that $\mathit{last\text{-}sample} \leq now + d$, which follows from Lemma 7.1.)

   Finally, consider a $[0,t]$-trajectory w whose first state is reachable. In w, *acc* is unchanged, and $\dot{v} \geq acc - \epsilon$ everywhere, by Lemma 7.1. Therefore,

   $$\frac{w(t).v - w(0).v}{t} \geq acc - \epsilon,$$

   that is,

   $$w(t).v - w(0).v \geq (acc - \epsilon)t.$$

We know by inductive hypothesis that

$$acc \geq \frac{b - \epsilon(w(0).now + d - \mathit{last\text{-}sample}) - w(0).v}{a - w(0).now}.$$

In other words,

$$w(0).v \geq b - \epsilon(w(0).now + d - \mathit{last\text{-}sample}) - acc(a - w(0).now).$$

Adding, we get:

$$w(t).v \geq b - \epsilon(w(t).now + d - \mathit{last\text{-}sample}) - acc(a - w(t).now).$$

In other words,

$$acc \geq \frac{b - \epsilon(w(t).now + d - \mathit{last\text{-}sample}) - w(t).v}{a - w(t).now}$$

This is what we needed to show.

2. For the upper bound, the argument is similar. The upper bound claim is true initially, since initially $acc = \frac{b}{a}$, $now = 0$ and $v = 0$.

   Now consider a discrete step starting from a reachable state. A *sample* step does not change any of the quantities mentioned in the inequality, and so it preserves the inequality. On the other hand, an *accel* step explicitly sets acc to $\frac{b - \mathit{sampled\text{-}vel}}{a - now}$, which is equal to $\frac{b-v}{a-now}$ by Lemma 7.2, which suffices to show the inequality.

   Finally, consider a $[0, t]$-trajectory w whose first state is reachable. In w, acc is unchanged, and $\dot{v} \leq acc$ everywhere, by Lemma 7.1. Therefore,

$$\frac{w(t).v - w(0).v}{t} \leq acc,$$

   that is,

$$w(t).v - w(0).v \leq acc \cdot t.$$

   We know by inductive hypothesis that

$$acc \leq \frac{b - w(0).v}{a - w(0).now}.$$

   In other words,

$$w(0).v \leq b - acc(a - w(0).now).$$

   Adding, we get:

$$w(t).v \leq b - acc(a - w(t).now).$$

   In other words,

$$acc \leq \frac{b - w(t).v}{a - w(t).now}.$$

   This is what we needed to show.

■

7.4 *Impl* Implements D_1

We show that *Impl* implements D_1 (see Theorem 7.5 for the formal statement), using a simulation from *Impl* to D_1.

Define the relation $fsim_2$ from states of *Impl* to states of D_1 as follows. If s_{Impl} is a state of *Impl* and s_{D_1} is a state of D_1, then we say that $(s_{Impl}, s_{D_1}) \in fsim_2$ provided that:

1. $s_{Impl}.now = s_{D_1}.now$.
2. $s_{Impl}.v = s_{D_1}.v$.
3. $s_{Impl}.acc = s_{D_1}.acc$.
4. $s_{Impl}.\dot{v} = s_{D_1}.\dot{v}$.

That is, $fsim_2$ is the identity mapping on all the state components of D_1. Note that all the state components of D_1 are derived from the *Vehicle* state in *Impl*. This is because the abstract system only mentions vehicle behavior, not controller behavior.

Lemma 7.4 *$fsim_2$ is a simulation from Impl to D_1.*

Proof: For the start condition, note that any combination of initial values allowed for all the state components in *Impl* is also allowed in D_1.

Next, consider a discrete step $(s_{Impl}, \pi, s'_{Impl})$ of *Impl*, where s_{Impl} and s_{D_1} are reachable states of *Impl* and D_1, respectively, and $(s_{Impl}, s_{D_1}) \in fsim_2$. There are two cases (again ignoring the trivial e case):

1. π is a *sample* action.

   Then we take the corresponding hybrid execution fragment to be trivial – just the trivial trajectory containing the single state s_{D_1}. It is easy to see that the step and the trivial trajectory have the same hybrid trace. Also, $(s'_{Impl}, s_{D_1}) \in fsim_2$, since this step does not change anything that affects any of the state components of D_1.

2. $\pi = accel(c)$.

   Now we take the corresponding hybrid execution fragment of D_1 to consist of a single *reset* step, $(s_{D_1}, reset, s'_{D_1})$. The state s'_{D_1} is obtained from the state s_{D_1} by modifying the *acc* and $\dot{v}$ components to their values in s'_{Impl}. The two steps have the same hybrid trace. Since π does not modify *now* or v, it should be clear that $(s'_{Impl}, s'_{D_1}) \in fsim_2$. It remains to show that $(s_{D_1}, reset, s'_{D_1})$ is in fact a step of D_1.

   The step of *Impl* causes *acc* to be set to $\frac{b - sampled\text{-}vel}{a - now}$, which is equal to $\frac{b-v}{a-now}$ by Lemma 7.2. It also causes $\dot{v}$ to be set to something in the range $[acc - \epsilon, acc]$. These changes are permitted in a *reset* step of D_1.

Finally, we consider a $[0,t]$-trajectory w_{Impl} whose first state is reachable. We allow this to correspond to a trajectory w_{D_1} of D_1, defined by simply projecting the states of *Impl* on the state components of D_1. The correspondence between the trajectories is then immediate. It remains to show that w_{D_1} is in fact a trajectory of D_1. Specifically, we show:

1. $\dot{v}$ is an integrable function in w_{D_1}.

   This follows from the definition of a trajectory of *Vehicle*.

2. For all $t \in I$, the following conditions hold in state $w(t)$.

(a) $now = w(0).now + t$.
This follows from the definition of a trajectory of *Vehicle*.

(b) If $now \neq a$ then $acc \in [\frac{b-\epsilon d-v}{a-now}, \frac{b-v}{a-now}]$.
The upper bound follows from Lemma 7.3, part 2. For the lower bound, Lemma 7.3, part 1, implies that, throughout w_{Impl} (except possibly at the right endpoint, if $now = a$ there), we have:

$$acc \geq \frac{b - \epsilon(now + d - \textit{last-sample}) - v}{a - now}.$$

(This uses the fact that $\textit{last-accel} = \infty$ throughout a trajectory; this is true because if not, then *last-accel* must be equal to *now* at the beginning of the trajectory, which would not permit time to pass.) Then the fact that $\textit{last-sample} \geq now$, stated in Lemma 7.1, yields the result.

(c) If $now \neq a$ then $\dot{v} \in [acc - \epsilon, acc]$.
This follows from the definition of a trajectory of *Vehicle*.

(d) $v = w(0).v + \int_0^t w(x).\dot{v}dx$.
This follows from the definition of a trajectory of *Vehicle*.

■

Now we can give the basic theorem relating *Impl* to D_1:

Theorem 7.5 *If α_{Impl} is a hybrid execution of Impl, then there is a hybrid execution α_{D_1} of D_1 having the same hybrid trace.*

Theorem 7.5 implies that the changes in *now* and v that are exhibited by *Impl* are allowed by D_1. The theorem does not mention the values of the other variables of D_1, *acc* and $\dot{v}$, but of course those correspond as well. We could have obtained this conclusion simply by regarding *acc* and $\dot{v}$ as output variables instead of internal variables.
Proof: By Lemma 7.4 and Theorem 2.1. ■

We can combine the results stated in Theorems 7.5 and 6.6 to obtain the following result, which relates the implementation *Impl* to the high-level specification automaton V_1. This is the main result of the paper.

Theorem 7.6 *If α_{Impl} is a hybrid execution of Impl, then there is a hybrid execution α_{V_1} of V_1 having the same hybrid trace.*

Theorem 7.6 implies that the changes in *now* and v that are exhibited by *Impl* are allowed by V_1.
Proof: By Theorem 7.5 and Theorem 6.6. ■

8 Discussion

We have described a simple vehicle deceleration maneuver as a composition *Impl* of hybrid I/O automata. In this maneuver, deceleration is accomplished using a controller that receives accurate velocity information at equally spaced times, and instantly responds with control signals containing the desired acceleration. However, there is some uncertainty, in that the proposed acceleration might not be exhibited exactly by the vehicle.

We have also given a correctness specification for the range of allowed velocities at various times, as another HIOA V_1. V_1 gives, in a simple closed form, an "envelope" that includes the allowed velocities. The envelope is sufficiently large to encompass the effects of both the acceleration uncertainty and the sampling delays.

We have verified, using extensions of standard computer science techniques (methods for reasoning about discrete systems), that the implementation *Impl* meets the specification V_1. In particular, our proof uses invariants and levels of abstraction. Invariants involve real-world quantities such as the velocity and acceleration, as well as state components of the controller. Our proof interposes an additional level of abstraction between the implementation and the specification, in which the system's behavior is represented using differential equations; uncertainty is included at this level also. Again, the representation is sufficient to encompass the effects of both acceleration uncertainty and sampling delay. Ideas from differential equations and from discrete analysis fit clearly into the appropriate places in the proof.

Our proof that *Impl* satisfies the specification V_1 is broken down into separate pieces, corresponding to different facts to be shown and different types of mathematical tools. It combines continuous and discrete reasoning cleanly, in a single framework. It gives a completely accurate description of the system's guarantees, including correct handling of the uncertainty and the effects of sampling delays.

Note that some complications of continuous mathematics – definability of derivatives, proper handling of infinities, etc. – arise at the intermediate level only, not at the top and bottom level. The top level just gives an envelope demarcated by explicitly-defined continuous functions. The bottom level gives a discrete algorithm. It is only the intermediate level of abstraction that uses the differential representation, and at which the complications of infinities arise.

Of course, this example is very simplified. It remains to generalize it to cases that include more uncertainty: the sampling times might be only approximately known, or velocity information might be inexact or out-of-date, or the control signal might be sent only after some approximately known delay. We have only considered uncertainty in the lower bound, but of course there could also be uncertainty in the upper bound. None of these cases appears to introduce any ideas that are different in principle, so we expect that the proofs we have given should extend to these cases. Another extension is that the implementation might be subject to a limit on the achievable acceleration (because of physical limitations or passenger comfort). It should be possible to use our techniques to reason about this situation also.

It should also be possible to continue our example by refining further. A natural extension would be to implement the discrete *Controller* using a more complicated algorithm, e.g., a distributed algorithm with its own difficulties of communication, uncertainty, etc. Techniques of discrete reasoning (only) could be used to show the correspondence between the more detailed controller and the more abstract controller of this paper. Then general composition theorems about HIOA's could be used to show that the combination of the new controller implementation and the given *Vehicle* automaton still guarantee the proper behavior of the vehicle, as expressed by V_1.

Our general strategy can be described as: using levels of abstraction to represent the relationship between a derivative and explicit form of a system representation, and also between a discrete and a continuous form, while incorporating uncertainties accurately throughout. It remains to use the same general strategy to model and verify other maneuvers, in particular, more complex ones. These two splits seem likely to be useful in many other examples.

We could use more levels of abstraction to represent more levels of derivatives. For example, if vehicle position at various times were the important consideration, then vehicle position only might be constrained at the top level, with velocity at the next level, acceleration at another level below that, and jerk at a fourth level, below the acceleration level. The correspondence between each successive pair of levels related by differentiation would just use standard methods of reasoning about differential equations (for the continuous parts of the correspondence).

Finally, the sort of reasoning we are doing in this paper admits assistance by mechanical reasoning tools. These could be a combination of a theorem-prover, for carrying out the discrete reasoning, and a tool for manipulating continuous function expressions, such as Mathematica. It is necessary to integrate the two types of tools so that they can be used together, using a single representation of the system.

References

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