Correctness of Vehicle Control Systems – A Case Study

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

Abstract

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

1 Introduction

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2 Hybrid I/O Automaton Model

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The importance of simulations is given by the following theorem.

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

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

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

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

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

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

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

3 Case 1: No Delay or Feedback

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

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

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

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

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

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

It stores the state of the brake in variable b. While braking, the train applies an acceleration that is nondeterministically chosen at every point but is constrained to be an integrable function with range in the interval $[\ddot{c}_{\min}, \ddot{c}_{\max}]$. While not braking, the train has acceleration exactly 0. The variable *now* represents the current time; when using assertions to reason about the timing behavior of systems, it is convenient to have an explicit state variable that records the current time. At this point in [26], we prove various fundamental facts about the mechanics of the train. Most of these facts relate the initial state and final states of a trajectory. Here, we give two examples of such lemmas. The first bounds change in velocity and position by change in time. The second bounds change in position by change in velocity. (Notation: If s and s' are states and x is a variable, we often write x for $s.x$ and x' for $s'.x$ when s and s' are understood.)

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

Table 1. The TRAIN automaton.

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

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

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

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

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

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

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

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

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

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

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

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

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

We define some more constants:

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

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

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

Table 2. The ONE-SHOT automaton

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

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

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

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

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

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

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

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

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

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

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

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

■

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

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

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

Proof: By induction. ■

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

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

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

4 Case 2: Delay and No Feedback

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

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

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

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

Table 3. The BUFFER **automaton.**

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

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

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

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

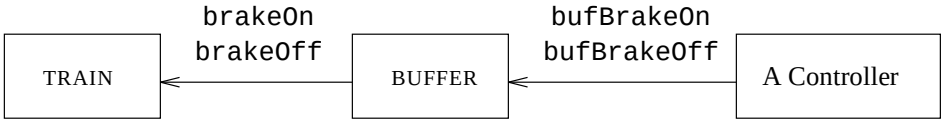

Figure 1. Overview of Delay Deceleration Model

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

5 Case 3: Feedback and No Delay

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

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

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

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

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

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

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

6 Case 4: Delay and Feedback

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

7 Conclusion

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

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

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

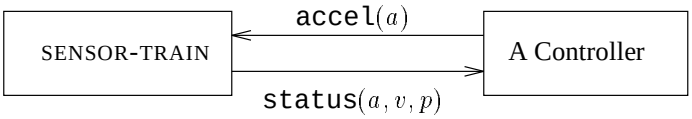

Figure 2. Overview of Feedback Deceleration Model

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

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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