Verification of Automated Vehicle Protection Systems

(extended abstract)

H.B. Weinberg * Nancy Lynch † Norman Delisle ‡

Abstract. We apply specification and verification techniques based on the *timed I/O automaton* model of Lynch and Vaandrager to a case study in the area of *automated transit*. The case study models and verifies selected safety properties for automated Personal Rapid Transit (PRT) systems such as PRT 2000^{TM}, a system currently being developed at Raytheon. Due to their safety critical nature, PRT 2000^{TM} and many other automated transit systems divide the control architecture into *operation* and *protection* subsystems. The operation system handles the normal control of vehicles. The protection system maintains safety by monitoring and possibly taking infrequent but decisive action. In this work, we present both a high-level treatment of a generic protection system and more detailed examinations of protection systems that enforce speed limits and vehicle separation.

1 Introduction

This paper presents a case study in the application of formal methods from computer science to the modeling and verification of hybrid systems. Our group's work is based on the Lynch-Vaandrager *timed I/O automaton* model [1] and uses a combination of several representation and verification methods. This is the second in a series of case studies undertaken by our group that focus on automated transit systems. An overview of our methods, the series of case studies, and our group's long term goals appears in this volume [2]. In the process of conducting these case studies, we extended the timed I/O automaton model to allow the precise description of continuous behavior. We call these extensions the *hybrid I/O automaton* model; a complete formal definition of this model appears in this volume [3].

Raytheon engineers are currently working on the design and development of a new PRT system called PRT 2000^{TM}. This system uses 4-passenger vehicles that

*`hbw@theory.lcs.mit.edu`. MIT Laboratory for Computer Science, Cambridge, MA 02139.
Research supported an NSF Graduate Fellowship and the grants and contracts below.

†`lynch@theory.lcs.mit.edu`. Research supported by NSF Grant 9225124-CCR, U.S. Department of Transportation Contract DTRS95G-0001-YR.8, AFOSR-ONR Contract F49620-94-1-0199, and ARPA Contracts N00014-92-J-4033 and F19628-95-C-0118.

‡`Norman_M_Delisle@ccmail.ed.ray.com`. Raytheon Company, 1001 Boston Post Road, Marlborough, MA 01752.

travel on an elevated guideway with Y-shaped diverges and merges. Passengers on this system board at stations and travel directly to their desired destination stations without intermediate stops. Compared to conventional transit systems, a PRT system can provide shorter average trip times and shorter average waiting times with equivalent passenger throughput. These performance improvements are achieved because the vehicles are separated on the guideway by only a few seconds, instead of the minutes typical of a conventional transit system. The vehicles are controlled by a distributed network of computers, which receive data from sensors on the vehicles and in the tracks. An important feature of the design is its absolute safety requirements: for example, vehicles must never collide and they must never exceed designated maximum speed limits, regardless of the behavior of the control software and hardware. These stringent design criteria have led to the complete separation of system functionality into two parallel components: Automated Vehicle Operation (AVO) and Automated Vehicle Protection (AVP). AVO is responsible for the normal control of the vehicles and can be composed of complex software and hardware. AVP is responsible for emergency control of vehicles and is designed to be simple and reliable. In ordinary operation, AVP is not supposed to take any action – it merely monitors the behavior of the vehicles, awaiting some potentially dangerous situation. However, AVP must monitor and react strongly enough to guarantee that, *regardless* of the behavior of AVO, basic safety is maintained. Note that this requirement includes the possibility that AVO contains errors.

The separation of operation and protection functions is a generally recognized engineering paradigm for the design of safety critical systems. In the transportation area, this structure was initially used in the design of railroad systems. Automatic safety systems were added to human-controlled railroad systems to protect against human error and mechanical malfunctions. As railroad and mass transit systems have evolved to become more automated, this division of labor has been retained, in the form of Automatic Train Operation and Automatic Train Protection systems. This paradigm occurs in most existing automated train systems, including the Washington Metro, the Miami People Mover, the O'Hare People Mover, the Detroit People Mover, and systems in Toronto, Vancouver, and Jacksonville. The use of this split migrated to automated vehicle systems with the pioneering Morgantown PRT system in the late sixties; this system has been in continuous active use for over 20 years with no serious accidents.

Our goal in this paper is to formalize the desired safety properties of an AVP system such as the one being developed by Raytheon. Specifically, we have examined overspeed protection and collision protection on straight tracks and during merges. These are by no means the only safety rules that the Raytheon AVP system enforces, but they are among the most complex. In each case, AVP receives frequent sensor data and takes some actions when parameters are getting too close to "bad values". Too close, in each case, means that it is possible that it will actually (eventually) reach a bad value if no action is taken at the present time. We view the relationship between AVP and the rest of the system as adversarial — the AVP system must maintain safety despite Byzantine faults in the rest of the system. Interesting modeling questions arise in deciding what powers the adversary has (it is certainly limited by the basic physics of motion, but what other limits are there?) and what control AVP can exert.

In this paper, we present a preliminary treatment of these AVP functions using hybrid I/O automata. First, in Section 2, we give an informal problem statement. Next, in Section 3, we present a high-level model of the adversarial relationship between physical system and AVP. In Section 4, we derive a generic theorem about the correctness of a class of AVP systems. In the remaining sections, we specialize this general model to two examples of particular AVP functions: overspeed protection and safe separation enforcement. The correctness of each of the specializations is a corollary of the general correctness theorem of Section 3. The proof methods we employ are predominantly assertional, with operational reasoning used in certain individual cases within inductive proofs, usually for describing the physics of the system. The inductive structure provides a convenient framework for the proofs, while allowing the use of standard types of reasoning about continuous functions where it is convenient. In this extended abstract we state only the final results and omit the intermediate lemmas and proofs.

2 Informal Problem Statement

A protection system is a subsystem that monitors the physical state for hazards and averts mishaps. The problem can be viewed as a "game" between AVP and an *adversary* that controls the physical system. The game proceeds by turns — on each turn AVP receives sensor information, takes some action, and the adversary chooses some evolution for the physical system. The game can be made more complex by introducing delays and uncertainty into the turn sequence; however, in this paper we will consider only the simple turn sequence that results from reliable, periodic, and timely communication. AVP cannot rely on the correct functioning of other computer systems, such as the AVO system. For this reason, AVO will not even figure in our models; rather, we will assume that AVP can be sure only that the system will not exceed its inherent physical limitations and those specifically imposed by AVP itself. In the interest of making protection systems robust, designers keep them simple; instead of having complex control abilities, protection systems depend only on the correct functioning of a few decisive emergency commands. We formalize this notion of limited, simple emergency controls by defining the powers of AVP to be *monotonically constraining*.

By "monotonic", we mean that the action AVP takes is irreversible. A typical action available to AVP is to activate an emergency brake; our monotonicity assumption means that, once engaged, the brake cannot be disengaged. Of course, in a real system some method to reverse the action is necessary, but typically this requires manual intervention and we do not model it in this paper.

By "constraining", we mean that AVP commands do not enable some behavior that was previously unavailable, but rather simply limit the possible behaviors of the system. For example, if a vehicle can exert a certain braking force in response to a command to brake, then it is reasonable to believe that it can achieve that force *without* the command. In terms of the game, the adversary cannot gain new abilities as the result of actions taken by AVP — in fact it usually loses them.

3 Generic Physical System

In this section, we present an abstract model of a generic physical system. The model is abstract because it does not specify any particular properties for the physical system, not even that there are vehicles or tracks. We also define what it means for an AVP subsystem (AVPS) responsible for averting a given mishap to be correct. This section is organized as follows: we introduce some notation, present a formal hybrid I/O automaton (HIOA) description of the physical system, and define correctness for an AVPS.

3.1 Notation

Let $\mathcal{P}(S)$ denote the set of all subsets of an arbitrary set S. Given an automaton, we write $s \xrightarrow{a} s'$ to mean that discrete action a in state s can lead to state s' in the automaton. Similarly, we say "s' is reachable from s " and write $s \leadsto s'$ to mean that there exists an execution fragment of the automaton which begins in s and ends in s'. If R is a predicate on states of an automaton, then we say "s' is R-reachable from s" and write $s \leadsto_R s'$ to mean that s' is reachable from s and R holds on all states in the execution fragment. For any $f : A \rightarrow B$, we define extensions of f mapping $\mathcal{P}(A) \rightarrow B$, and overload the symbol f so that it refers to both functions. If $f : A \rightarrow B$ and we write $f \equiv b$ for $b \in B$, then f is the constant function whose value is b everywhere. If R is a predicate on the valuations of some subset Q of the variables of an automaton, then we extend it to states s of the automaton via projection: $f(s)$ if and only if $f(s\lceil Q)$. If s is a state of an automaton and x is a variable of that automaton, then $s.x$ is the value of the variable in state s. If f is a function to states of an automaton, then $f.x$ is the projection of f onto the variable x. When defining the trajectory set of an automaton we use the symbol w to quantify over all the possible trajectories and the symbol I for the domain of w. The trajectories of an automaton are all the w what satisfy the conditions given in the trajectory set definition.

3.2 Physical System Automata

The physical system is modeled as three automata: GP (generic plant), SENSOR, and ACTUATOR.
The GP **Automaton:** We do not define GP explicitly but rather give a set of properties that it must satisfy. The restrictions on GP are of two types: we specify GP's signature and give an axiom that GP must satisfy. Our specification

of the signature describes the relationship between GP and other components; the axiom formalizes the notion of constraint.

The signature of an HIOA consists of its state variables and actions and a partition of them into three groups: input, output, and internal. GP has exactly one input variable *con* (discussed below); it has no internal variables; and it may have arbitrary output variables, except that they must include the usual current time variable *now*. GP may have arbitrary input, output, and internal actions. We denote a single state of GP by p. Furthermore, P denotes a set of states; $\mathbf{P}$ denotes the set of all states. We use $p\lceil L$ to denote the projection of a state of GP to its local state — in other words, the state minus the *con* variable.

The input variable *con* models the current constraint set imposed on the physical system by the protection system's actuators. The powers of the protection system are modeled as a set of constraints, $\mathcal{C}$. The variable *con* takes values over $\mathcal{P}(\mathcal{C})$.

The GP must also satisfy the following axiom:

> **Constraint Axiom** For all $p, q \in \mathbf{P}$, if $p\lceil L = q\lceil L$ and $p.con \supseteq q.con$ then:
>
> 1. For all $p' \in \mathbf{P}$ and for all discrete actions a, if $p \xrightarrow{a} p'$ and $p.con = p'.con$, then there exists $q' \in \mathbf{P}$ such that $q \xrightarrow{a} q'$, $p'\lceil L = q'\lceil L$ and $q.con = q'.con$.
> 2. For all closed trajectories $w : [0, t] \rightarrow \mathbf{P}$, if $w(0) = p$ and $w.con \equiv p.con$ then there exists a trajectory $y : [0, t] \rightarrow \mathbf{P}$ such that $y(0) = q$, $y\lceil L = w\lceil L$, and $y.con \equiv q.con$.

The axiom says that if a local state is reachable in one step or trajectory under a certain constraint, then the same local state is reachable under any weaker constraint. A consequence of this axiom is a similar condition involving multi-step executions instead of just single steps.

The SENSOR and ACTUATOR automata: Due to space considerations we do not present these simple automata, but merely describe them informally. The SENSOR automaton has all of GP's state variables as inputs. Fix a constant δ. At time zero and every δ time units thereafter, the SENSOR outputs this entire state p through the discrete action `snapshot`(p).

The ACTUATOR automaton has input actions `constrain`(C) for $C \subseteq \mathcal{C}$ and output variable *con*. The input actions model the sequence of constraints added by the AVPS; these constraints accumulate in *con* via set union. This captures the "monotonic" part of monotonically constraining, i.e. the set of active constraints never shrinks. Note that to compose multiple automata with `constrain` output actions, we will require a separate `constrain`$(C)_i$ action for each source where i varies over some index set of the sources. We omit the subscripts when convenient.

3.3 Definition of AVPS

We define an AVPS to be an automaton with no input or output variables and exactly the following input and output actions: the set of input actions is $\{\texttt{snapshot}(p) \mid p \in \mathbf{P}\}$ and the set of output actions is $\{\texttt{constrain}(C)_i \mid C \subseteq \mathcal{C}$ and $i \in N\}$ where N is a finite subset of $\mathbb{N}$. The set N allows for an AVPS to be composed of multiple automata that output the `constrain` action. An AVPS may have arbitrary internal variables and internal actions. The composition of two compatible AVPSs yields an AVPS. We will usually ignore the subscripting of the `constrain` actions and assume that subscripts are assigned in a way that makes the AVPSs we wish to compose compatible.

We define a notion of correctness for an AVPS. We characterize certain states as "bad": these are the states that the AVPS is supposed to protect against. We also qualify (i.e. weaken) the claim of correctness of an AVPS by saying that the AVPS only protects against bad states when GP starts out in a certain set of local states and remains in a certain (possibly different) set of local states.

Let A be an AVPS; let `bad`, S, and R be predicates on the local state of GP; let ALL be the composition of GP, SENSOR, ACTUATOR, and A. We define that *A averts* `bad` *in* GP *starting from S under invariant R*, when no execution of ALL that begins in an S state and contains only R states leads to a `bad` state. If R or S is just "true" then we omit it.

This definition of correctness leads to two useful theorems about the composition of AVPSs. The first addresses independent AVPSs; the second, a one-way dependence among AVPSs.

Theorem 3.1 *If AVPSs A_1 and A_2 are compatible, A_1 averts* $\texttt{bad}_1$ *in* GP *starting from S_1 under invariant R_1, and A_2 averts* $\texttt{bad}_2$ *in* GP *starting from S_2 under invariant R_2, then $A_1||A_2$ averts* $\texttt{bad}_1 \vee \texttt{bad}_2$ *in* GP *starting from $S_1 \wedge S_2$ under invariant $R_1 \wedge R_2$.*

Theorem 3.2 *If AVPSs A_1 and A_2 are compatible, A_1 averts* $\texttt{bad}_1$ *in* GP *starting from S_1 under invariant R_1, and A_2 averts* $\texttt{bad}_2$ *in* GP *starting from S_2 under invariant $\neg\texttt{bad}_1 \wedge R_2$, then $A_1||A_2$ averts* $\texttt{bad}_1 \vee \texttt{bad}_2$ *in* GP *starting from $S_1 \wedge S_2$ under invariant $R_1 \wedge R_2$.*

4 A Generic Protection System: PROTECTOR

Fix predicates `bad` and R on the local state of GP. In Figure 1, we give an example AVPS called PROTECTOR and below a predicate `safe`, such that PROTECTOR averts `bad` in GP starting from `safe` under the invariant R. The PROTECTOR automaton receives each `snapshot` and immediately responds with an appropriate constraint. The heart of PROTECTOR is the pair of functions $\bar{C}$ and `future` and the predicate `next-safe`. These are defined as follows:

$\bar{C} : \mathbf{P} \to \mathbf{P}$, where $C \subseteq \mathcal{C}$, defined by

> $\bar{C}(p) = p'$ where $p'\lceil L = p\lceil L$ and $p'.con = p.con \cup C$.
> This function mimics the effect on GP of a $\texttt{constrain}(C)$ action. We extend it to sets of states as follows: $\bar{C}(P) = \{p' | p \in P \text{ and } p' = \bar{C}(p)\}$

$\texttt{future} : (\mathbb{R}^{\geq 0} + \infty) \times \mathbf{P} \rightarrow \Pi$, defined by

$\texttt{future}(t, p) \equiv \{p' | (p \rightsquigarrow_R p' \text{ with } con \text{ constant}) \wedge (p'.now - p.now \leq t)\}$

This function returns the set of states R-reachable from state p in t time with the *con* input variable held constant. We extend the function to sets of states as follows: $\texttt{future}(t, P) = \bigcup_{p \in P} \texttt{future}(t, p)$

The definition of **`next-safe`** requires some auxiliary functions:

$\texttt{tick} : \mathbf{P} \rightarrow \Pi$, defined by $\texttt{tick}(p) \equiv \texttt{future}(\delta, p)$

This function returns the set of states R-reachable from p in δ time or less. If p is a state reported in a `snapshot` action, then this function returns the set of states R-reachable from state p up to and including the time of the next snapshot. We extend it to sets of states: $\texttt{tick}(P) = \bigcup_{p \in P} \texttt{tick}(p)$

$\texttt{panic} : \mathbf{P} \rightarrow \Pi$, defined by $\texttt{panic}(P) \equiv \texttt{future}(\infty, \bar{\mathcal{C}}(\texttt{future}(0, P)))$

This functions returns the set of all states R-reachable from state p if all constraints are applied before any time passes. Note that the inner use of `future` allows some discrete actions to occur before the full constraint set is applied. We extend `panic` to sets of states: $\texttt{panic}(P) = \bigcup_{p \in P} \texttt{panic}(p)$

Given these auxiliary functions we define the **`safe`** predicate which is necessary both for the definition of **`next-safe`** and as the restriction on the initial states of GP. A safe state is one where if the protection system can act before any more time passes, it can avoid R-reachable bad states. Because of the Constraint Axiom, this is equivalent to saying that applying all the constraints before any more time passes would avoid all R-reachable bad states for the rest of the evolution of the system.

$\texttt{safe} : \mathbf{P} \rightarrow bool$, defined by $\texttt{safe}(p) \equiv \neg \texttt{bad}(\texttt{panic}(p))$

We extend `safe` to sets of states as follows: $\texttt{safe}(P) \equiv \bigwedge_{p \in P} \texttt{safe}(p)$

Finally, we define the **`next-safe`** predicate. A next-safe state is one where, if the protection system takes no action for δ time, then the system will be safe from now up to and including that time. Usually we are examining the states that are reported in a `snapshot` action, in which case a next-safe state is one where if the protection system takes no action until the time of the next snapshot, then the system will be safe from now up to and including that time. Every next-safe state is itself a safe state. A state which is safe but not next-safe is hazardous.

$\texttt{next-safe} : \mathbf{P} \rightarrow bool$, defined by $\texttt{next-safe}(p) \equiv \texttt{safe}(\texttt{tick}(p))$

We extend `next-safe` to sets of states as follows:

$\texttt{next-safe}(P) \equiv \bigwedge_{p \in P} \texttt{next-safe}(p)$

This completes the definition of PROTECTOR.

One can imagine a "trivial" AVPS that immediately applies the entire constraint set. Such a controller would correctly avert bad states if the system starts in a safe state. However, we are interested in less restrictive controllers, i.e. controllers that send weaker constraint sets than the complete set when possible. An AVPS may find weaker constraint sets by testing whether a candidate set is sufficient to guarantee the safety of the system *until the next snapshot*. This idea is captured by the next-safe states: our PROTECTOR AVPS identifies hazardous states and imposes any constraint that converts them to next-safe states. An "optimal" version would choose the weakest possible constraint.

Due to space considerations we give only the final result:

Figure 1 PROTECTOR automaton description

Actions:	Input:	$\mathtt{snapshot}(p)$, where $p \in \mathbf{P}$
	Output:	$\mathtt{constrain}(C)$, where $C \subseteq \mathcal{C}$
Variables:	Internal:	$send \in \mathcal{P}(\mathcal{C}) \cup \{\, \mathtt{none} \,\}$, initially $\mathtt{none}$

Transitions:

$\mathtt{snapshot}(p)$
Eff: $send := C$, where $C \subseteq \mathcal{C}$
such that $\mathtt{next\text{-}safe}(\bar{C}(\mathtt{future}(0, p)))$ holds,
if any exists; otherwise $C = \mathcal{C}$.

$\mathtt{constrain}(C)$
Pre: $send = C$
Eff: $send := \mathtt{none}$

Trajectories:
$w.send \equiv \mathtt{none}$

Theorem 4.1 *AVPS* PROTECTOR *averts* $\mathtt{bad}$ *in* GP *starting from* $\mathtt{safe}$ *under invariant R.*

5 Example 1: Overspeed

In this section, we present a model of n vehicles on a single infinite track and an AVPS that stops vehicles from exceeding a speed limit provided that they do not collide. In an actual system, speed limits may vary from one region to another; in this section, we assume a single global speed limit. The model of the physical system, called VEHICLES, conforms to the restrictions on the GP automaton of Section 3. We define an AVPS, called OS-PROT, which enforces the speed limit on all vehicles. This AVPS is the composition of n separate copies of another AVPS called OS-PROT-SOLO, one copy for each vehicle. Each OS-PROT-SOLO$_i$ for $1 \leq i \leq n$ implements the abstract PROTECTOR automaton of Section 3 and enforces the speed limit for one vehicle.

We describe in detail those aspects of the model which were only abstract in Section 3. These include: the bad states; the constraint set C; the unspecified variables, trajectories, and discrete actions of GP. The constraints model AVP's ability to order a vehicle or set of vehicles to "emergency brake"; the unspecified variables model (among other things) the position, velocity, and acceleration of each vehicle; the trajectories model the motion of the vehicles, within physical constraints; there are internal discrete actions of the physical system that model the possibility that vehicles stop suddenly.

To give an implementation of the AVPS, we will introduce closed form redefinitions of the predicates of Section 3. The predicates of Section 3 were defined in terms of the possible future states of GP; the analogous predicates in this section will instead be defined in terms of the current state. The proof of correctness relies on the fact that the new versions are conservative approximations of the abstract versions.

5.1 Plant: VEHICLES

There are n vehicles modeled in a single automaton called VEHICLES described in Figure 2. Each vehicle is modeled with three variables, x_i, $\dot{x}_i$, and $\ddot{x}_i$, for i where $1 \leq i \leq n$. These are the position, velocity, and acceleration of each vehicle. The acceleration of a vehicle is bounded above and below: $\ddot{x}_i \in [\ddot{c}_{min}, \ddot{c}_{max}]$, where $\ddot{c}_{min} < 0 < \ddot{c}_{max}$. Furthermore, when braking the vehicles have exactly $\ddot{x}_i = \ddot{c}_{brake}$, where $\ddot{c}_{min} < \ddot{c}_{brake} < 0$. The difference between the minimum acceleration and the braking acceleration reflects a conservative estimate of the effectiveness of the vehicles' braking systems. The velocity is also restricted to be non-negative. The constraint set is $\mathcal{C} = \{1, \ldots, n\}$. When $i \in con$, this means that vehicle i is emergency braking. An internal action called `brick-wall` models the instantaneous stopping of a vehicle — as if it hit a brick wall.

We must specify the bad states for the overspeed protector: these are the states in which a vehicle exceeds the maximum velocity $\dot{c}_{max}$. More formally, the overspeed "bad" predicate is:

`overspeed` : $\mathbf{P} \rightarrow bool$, defined by `overspeed`$(p) \equiv \exists i \;\; p.\dot{x}_i > \dot{c}_{max}$

This predicate is the instantiation of the `bad` predicate from the generic case. It is extended to sets of states in a similar way to `bad`; a set of states is `overspeed` if any element of the set is `overspeed`.

5.2 Protection System: OS-PROT-SOLO$_i$

We can build an AVPS for overspeed protection from a single AVPS for each vehicle. We define the vehicle-wise "bad" predicate `overspeed-solo`$_i$ to be $\dot{x}_i > \dot{c}_{max}$. To construct the corresponding single vehicle protection system, we define "safe" and "next-safe" predicates that only test each vehicle separately.

Figure 3 shows OS-PROT-SOLO$_i$, an example protection system which maintains $\neg$`overspeed-solo`$_i$. It is a special case of PROTECTOR of Section 4. For the automaton definition to be complete we must give definitions for `os-safe-solo`$_i$ and `os-next-safe-solo`$_i$ which are analogous to `safe` and `next-safe`. They are extended to sets of states in the same manner as `safe` and `next-safe`.

`os-safe-solo`$_i$: $\mathbf{P} \rightarrow bool$, defined by `os-safe-solo`$_i(p) \equiv \neg$`overspeed-solo`$_i(p)$

`os-next-safe-solo`$_i$: $\mathbf{P} \rightarrow bool$, defined by `os-next-safe-solo`$_i(p) \equiv$
$(\dot{x}_i \leq \dot{c}_{max} - \delta \ddot{c}_{max})$

This completes the definition of OS-PROT-SOLO$_i$. Let `os-safe` be the conjunction of `os-next-safe-solo`$_i$ for all i. Let OS-PROT be the composition of OS-PROT-SOLO$_i$ for all i.

Corollary 5.1 *AVPS* OS-PROT-SOLO$_i$ *averts* `overspeed-solo`$_i$ *in* VEHICLES *starting from* `os-safe-solo`$_i$.

This result can be proved by showing that the automata of this section are a specialization of those of Section 3 and applying Theorem 4.1.

Corollary 5.2 *AVPS* OS-PROT *averts* `overspeed` *in* VEHICLES *starting from* `os-safe`.

This result follows from Corollary 5.2 and Theorem 3.1.

Figure 2 VEHICLES automaton description

Actions:	Internal:	`brick-wall`(i) for all $i \in \{1, \ldots, n\}$
Variables:	Output:	$x_i, \dot{x}_i, \ddot{x}_i \in \mathbb{R}$ for all $i \in \{1, \ldots, n\}$
		$stopped_i$, boolean, for all $i \in \{1, \ldots, n\}$, all initially `false`
	Input:	$con \subseteq \{1, \ldots, n\}$, the dynamic-type is constant functions

Transitions:

`brick-wall`(i)
  Eff: $stopped_i :=$ `true`
  $\ddot{x}_i := \dot{x}_i := 0$

Trajectories:
  for all $i \in \{1, \ldots, n\}$
    the function $w.\ddot{x}_i$ is integrable
    for all $t \in I$ where $t \neq 0$
      $w(t).stopped_i = w(0).stopped_i$
      $0 \leq w(t).\dot{x}_i$
      $w(t).\dot{x}_i = w(0).\dot{x}_i + \int_0^t w(s).\ddot{x}_i \, ds$
      $w(t).x_i = w(0).x_i + \int_0^t w(s).\dot{x}_i \, ds$
      if $w(t).stopped_i$ then $w(t).\ddot{x}_i = 0$
      else if $i \notin con$ then $w(t).\ddot{x}_i \in [\ddot{c}_{min}, \ddot{c}_{max}]$
        else if $w(t).\dot{x}_i = 0$ then $w(t).\ddot{x}_i = 0$
          else $w(t).\ddot{x}_i = \ddot{c}_{brake}$

Figure 3 OS-PROT-SOLO$_i$ automaton description (subscript i omitted)

Actions:	Input:	`snapshot`(p), where p ranges over the states of VEHICLES
	Output:	`constrain`(C) for $C \subseteq \mathcal{C}$
Variables:	Internal:	boolean $send \in \mathcal{P}(C) \cup \{$ `none` $\}$, initially `none`

Transitions:

`snapshot`(p)
  Eff: if `os-next-safe-solo`(p) then
      $send := \emptyset$
    else
      $send := \{i\}$

`constrain`(c)
  Pre: $send = c$
  Eff: $send :=$ `none`

Trajectories:
  $w.send \equiv$ `none`

6 Example 2: Safe Separation on a Single Track

This section is similar to Section 5; instead of an overspeed protection system, here we model a collision protection system for the same physical system, VEHICLES. This AVPS can apply the same constraints and receives the same snapshot information as that section's OS-PROT. However, the collision protection system relies on the overspeed protection system. As in Section 5 the protector of this section, called CL-PROT, is an instantiation of the generic PROTECTOR from Section 3.

The mishap we wish to avoid is that two vehicles collide. Each vehicle occupies some distance on the track given by the `extent` function:

`extent` $: \mathbb{R} \rightarrow \mathcal{P}(\mathbb{R})$, defined by `extent`$(x) = [x, x + c_{len}]$

It takes as an argument the current position of a vehicle and maps it to the section of track occupied by the vehicle. The positive constant c_{len} captures the minimum allowable separation between vehicles; this includes the length of the vehicle plus any desired extra margin specified by the system designer. Now we define the predicate `collide` which tests if the extents of two vehicles overlap.

`collide` $: \mathbf{P} \rightarrow bool$, defined by `collide`$(P) \equiv$

$\exists i\ \exists j\ \ (i \neq j) \wedge (\texttt{extent}(p.x_i) \cap \texttt{extent}(p.x_j) \neq \emptyset)$

Due to space considerations we do not present CL-PROT, the collision protector. The main ideas are captured in the definitions of the **`cl-safe`** and **`cl-next-safe`** predicates. To define them we first introduce some useful notation and five auxiliary functions. A *configuration* X of a vehicle is a 4-tuple of type $\mathbb{R} \times \mathbb{R} \times bool \times bool$ which represent respectively: position, velocity, whether the vehicle is stopped and whether the vehicle is braking. If X is a configuration then $X.x$, $X.\dot{x}$, $X.stopped$, and $X.brake$ refer respectively to the elements of the configuration. If p is a state of VEHICLES, we use $p.X_i$ to denote the configuration of vehicle i.

`cl-stop-dist`$(\dot{x})$ is the distance required to stop a vehicle with speed $\dot{x}$ assuming $\ddot{c}_{brake}$ deceleration.

`cl-fast-dist`(X) is the maximum distance a vehicle with configuration X can travel in δ time units, assuming maximum acceleration and correct overspeed protection.

`cl-fast-vel`(X) is the maximum velocity achievable in δ time (as in `cl-fast-dist`).

`cl-safe-dist`(X) is the distance in front of a vehicle that the vehicle "owns"; that is, the length of the vehicle plus the distance that it might travel even if braked immediately. If the ownerships of two vehicles overlap then the state is not safe.

`cl-next-safe-dist`(X) is the distance in front of a vehicle that the vehicle "claims"; that is, the length of the vehicles plus the distance that it might travel if left unbraked for δ time units and then braked. If the claims of two vehicles overlap then the state is not next-safe.

Formal definitions of these functions and of the **`cl-safe`** and **`cl-next-safe`** predicates appears in Table 1. A state is safe if there is no ownership overlap; a state is next-safe if there is no claim overlap. The strategy of CL-PROT is to brake the trailing vehicle when two vehicles have a claim overlap.

We give only the final results:

Table 1 Functions used in the definition of CL-PROT.

$$\texttt{cl-stop-dist}(\dot{x}) = -\frac{\dot{x}^2}{2\ddot{c}_{brake}}$$

$$\texttt{cl-fast-dist}(X) = \begin{cases} 0 & \text{if } \mathit{stopped} \\ \dot{x}t + \frac{1}{2}\ddot{c}_{brake}t^2, & \\ \quad \text{where } t = \min(\delta, \frac{-\dot{x}}{\ddot{c}_{brake}}) & \text{if } \neg\mathit{stopped} \text{ and } \mathit{brake} \\ \dot{x}t + \frac{1}{2}\ddot{c}_{max}t^2 + \dot{c}_{max}(\delta - t), & \\ \quad \text{where } t = \min(\delta, \frac{\dot{c}_{max} - \dot{x}}{\ddot{c}_{max}}) & \text{otherwise} \end{cases}$$

$$\texttt{cl-fast-vel}(X) = \begin{cases} 0 & \text{if } \mathit{stopped} \\ \max(0, \dot{x} + \delta(\ddot{c}_{brake})) & \text{if } \neg\mathit{stopped} \text{ and } \mathit{brake} \\ \min(\dot{c}_{max}, \dot{x} + \delta(\ddot{c}_{max})) & \text{otherwise} \end{cases}$$

$$\texttt{cl-safe-dist}(X) = c_{len} + \texttt{cl-stop-dist}(\dot{x})$$

$$\begin{aligned}\texttt{cl-next-safe-dist}(X) = c_{len} &+ \texttt{cl-fast-dist}(X) \\ &+ \texttt{cl-stop-dist}(\texttt{cl-fast-vel}(X))\end{aligned}$$

$$\begin{aligned}\texttt{cl-safe}(p) \equiv \forall i\, \forall j \;\; i \neq j \text{ implies } &[x_i, x_i + \texttt{cl-safe-dist}(X_i)] \\ \text{and } &[x_j, x_j + \texttt{cl-safe-dist}(X_j)] \text{ are disjoint}\end{aligned}$$

$$\begin{aligned}\texttt{cl-next-safe}(p) \equiv \forall i\, \forall j \;\; i \neq j \text{ implies } &[x_i, x_i + \texttt{cl-next-safe-dist}(X_i)] \\ \text{and } &[x_j, x_j + \texttt{cl-next-safe-dist}(X_j)] \text{ are disjoint}\end{aligned}$$

Corollary 6.1 *AVPS* CL-PROT *averts* `collide` *in* VEHICLES *starting from* `cl-safe` *under invariant* ¬`overspeed`.

This follows from Theorem 4.1.

Corollary 6.2 *The composition of* OS-PROT *and* CL-PROT *averts* (`overspeed` ∨ `collide`) *in* VEHICLES *starting from* `os-safe` *and* `cl-safe`.

This follows from Theorem 3.2 and Corollaries 5.2 and 6.1.

7 Conclusions and Future Work

In this work we have demonstrated how hybrid I/O automaton techniques can be applied to the specification and verification of a very general automated transit problem. The specification technique involves refinement of an abstract model. The proof structure permits extensive reuse of reasoning. Compositional properties are exploited to yield a hierarchical decomposition of the protection system that reflects the dependencies among critical components.

This treatment of protection systems is a preliminary case study. We would like to examine less idealized cases, including cases where the communication is unreliable and/or delayed and where vehicle sensor information is reported asynchronously.

The formalization presented in this paper has already had an influence on the PRT system under development at Raytheon. In particular, the formalization of `next-safe` exposed important safety criteria that led to practical methods to handle latencies and uncertainties due to discrete sampling and scheduling of the protection system. For example, the formalization of the overspeed protector revealed that the maximum speed attainable by a vehicle is strictly greater than the minimum speed which the protection system considers a hazard. This precise characterization of the overspeed protector is critical to the safe operation of the system, especially because other components of the system such as the collision protection system rely on guarantees made by the overspeed protection system.

We believe the techniques developed in this paper complement more traditional safety analysis. For example, safety engineers typically perform a fault-tree analysis to identify possible causes of each system hazard and related dependencies among system components. In our work, we use composition of automata to formalize these dependencies: to yield a speed limited system, we compose the physical system with a set of overspeed protectors, one for each vehicle — this formalizes the independence of overspeed protection for separate vehicles; conversely, the collision protection system controls the speed limited system — this formalizes the dependence of collision protection on overspeed protection. We believe a more comprehensive treatment in this style of all the protection subsystems would as a by-product yield a significant subtree of the fault-tree.

This treatment of transit systems overall is a work in progress. Our goal is to develop a general theoretical framework for specifying, verifying, and analyzing transit systems.

Acknowledgments: We thank Steve Spielman and Victor Luchangco for helpful discussions and Ekaterina Dolginov for a careful reading of the manuscript.

References

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

[2] Nancy Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. This volume.

[3] Nancy Lynch, Roberto Segala, Frits Vaandrager, and H.B. Weinberg. Hybrid I/O automata. This volume.