Formal Verification of Safety-Critical Hybrid Systems[⋆]

Carolos Livadas and Nancy A. Lynch

Laboratory for Computer Science
Massachusetts Institute of Technology
{clivadas,lynch}@theory.lcs.mit.edu

Abstract. This paper investigates how formal techniques can be used for the analysis and verification of hybrid systems [1, 5, 7, 16] — systems involving both discrete and continuous behavior. The motivation behind such research lies in the inherent similarity of the hierarchical and decentralized control strategies of hybrid systems and the communication and operation protocols used for distributed systems in computer science. This paper focuses on the use of hybrid I/O automata [11, 12] to model, analyze, and verify safety-critical hybrid systems that use emergency control subsystems to prevent the violation of their safety requirements. The paper is split into two parts. First, we develop an abstract model of a *protector* — an emergency control component that guarantees that the physical plant at hand adheres to a particular safety requirement. The abstract protector model specialized to a particular physical plant and a particular safety requirement constitutes the specification of a protector that enforces the particular safety property for the particular physical plant. The correctness proof of the abstract protector model leads to simple correctness proofs of the implementations of particular protectors. In addition, the composition of independent protectors, and even dependent protectors under mild conditions, guarantees the conjunction of the safety properties guaranteed by the individual protectors being composed. Second, as a case study, we specialize the aforementioned abstract protector model to simplified versions of the personal rapid transit system (PRT 2000$^{\text{TM}}$) under development at Raytheon Corporation and verify the correctness of overspeed and collision avoidance protectors. Such correctness proofs are repeated for track topologies ranging from a single track to a directed graph of tracks involving Y-shaped merges and diverges.

⋆ This research was performed at the Theory of Distributed Systems Group of the Laboratory for Computer Science of the Massachusetts Institute of Technology. The research was supported by NSF Grant 9225124-CCR, U.S. Department of Transportation Contract DTRS95G-0001-YR.8, AFOSR-ONR Contracts F49620-94-1-0199 and F49620-97-1-0337, and ARPA Contracts N00014-92-J-4033 and F19628-95-C-0118.

1 Introduction

The recent trend of system integration and automation has encouraged the study of hybrid systems — systems that combine continuous and discrete behavior. Although the individual problems of continuous and discrete behavior have been extensively analyzed by control theory and formal analysis, respectively, their combination has recently been aggressively studied. In particular, the automation in various safety-critical systems, such as automated transportation systems, has indicated the need for formal approaches to system analysis, design, and verification. Automated highway systems [2,8], personal rapid transit systems [6,17], and air traffic control systems [9,15] have served as benchmark problems for the development of techniques to analyze, design, and verify hybrid systems.

Many of the safety-critical systems in use today abide by the engineering paradigm of using an emergency control, or protection, subsystem to prevent the violation of the system's safety requirements. In this paper we present a formal framework for the analysis of systems that adhere to this engineering paradigm. The framework is used to prove the correctness of such protection subsystems in an effort to provide indisputable system safety guarantees. The formal approach to the analysis of such systems has several advantages. Formal analysis yields a precise specification of the system and its safety requirements, provides insight as to the location of possible design errors, and minimizes the duplication of verification effort when such errors are corrected. The technique of system validation through exhaustive testing lacks the insightful feedback and requires full-fledged regression testing when design errors are detected.

In this paper, we use *hybrid I/O automata* [11,12] — an extension of *timed I/O automata* [4,14] — to define an abstract model of a *protector* — a subsystem that guarantees that the physical plant adheres to a particular safety requirement. The abstract protector model is parameterized in terms of the physical plant, the safety requirement, and several other quantities. The instantiation of the abstract protector, obtained by specifying the abstract protector's parameters, constitutes the specification of a protector that guarantees a particular safety property for a particular physical plant model. The proof of correctness of the abstract protector model minimizes the effort in verifying the correct operation of a particular protector implementation. In fact, such correctness proofs get reduced to simple simulations from the protector implementations to the particular instantiation of the abstract protector model. As a case study, we apply the formal framework developed towards the verification of overspeed and collision protection subsystems for simplified models of the personal rapid transit system (PRT 2000™) under development at Raytheon Corporation. The case studies presented in this paper extend the work of Weinberg, Lynch, and Delisle [17] by introducing a powerful formal framework that allows more complete system models to be used. The actual PRT 2000™ system is comprised of 4-passenger vehicles that travel on an elevated guideway of tracks involving Y-shaped merges and diverges and provide point-to-point passenger transportation. In this treatment, we verify the correct operation of overspeed and collision avoidance protectors for track topologies ranging from a single track to a directed graph of tracks involving Y-shaped merges and diverges. A detailed treatment of the work presented in this paper can be found in Ref. 6.

2 Hybrid I/O Automata

A hybrid I/O automaton A is a (possibly) infinite state model of a system involving both discrete and continuous behavior. The automaton $A = (U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, \mathcal{D}, \mathcal{W})$ consists of three disjoint sets U, X, and Y of variables (*input*, *internal*, and *output* variables, respectively), three disjoint sets Σ^{in}, Σ^{int}, and Σ^{out} of *actions* (*input*, *internal*, and *output* actions, respectively), a nonempty set Θ of *initial states*, a set $\mathcal{D}$ of *discrete transitions* and a set $\mathcal{W}$ of trajectories over V, where $\Sigma = \Sigma^{in} \cup \Sigma^{int} \cup \Sigma^{out}$ and $V = U \cup X \cup Y$. The initial states, the discrete transitions, and the trajectories of any HIOA A must however satisfy several technical conditions which are omitted here. For a detailed presentation of the HIOA model, the reader is referred to Refs. 11 and 12.

Variables in the set V are typed; that is, each variable $v \in V$ ranges over the set of values *type*(v). A *valuation* of V, also referred to as a *state* of A, is a function that associates to each variable v of V a value in *type*(v). The set of all valuations of V, or equivalently the set of all states of A, is denoted by $\boldsymbol{V}$, or equivalently *states*(A). Letting $v \in V$ and $S_v \subseteq$ *type*(v), we use the notation $v :\in S_v$ to denote the assignment of an arbitrary element of the set S_v to the variable v. Similarly, letting $S_V \subseteq \boldsymbol{V}$, we use the notation $V :\in S_V$ to denote the assignment of an element of the set *type*(v) to the variable v, for each v in V, such that the resulting valuation of V is an arbitrary element of the set S_V. Letting s be a state of A, *i.e.*, $s \in \boldsymbol{V}$, and $V' \subseteq V$, we define the *restriction* of s to V', denoted by $s\lceil V'$, to be the valuation s' of the variables of V' in s. Letting $\boldsymbol{X} \subseteq \boldsymbol{V}$, we say that $\boldsymbol{X}$ *is* V'*-determinable* if for all $x \in \boldsymbol{X}$ and $s \in \boldsymbol{V}$, such that $x\lceil V' = s\lceil V'$, it is the case that $s \in \boldsymbol{X}$. The continuous time evolution of the valuations of the variables in V is described by a *trajectory* w over V; that is, a function $T_I \to \boldsymbol{V}$, where T_I is a left-closed interval of $\mathbb{R}^{\geq 0}$ with left endpoint equal to 0. The *limit time* of w, denoted by $w.ltime$, is defined to be the supremum of the domain of w, *dom*(w). We define the *first state* of a trajectory w, denoted by $w.fstate$, to be the state $w(0)$. Moreover, if the domain of a trajectory w is right-closed, then we define the *last state* of w, denoted by $w.lstate$, to be the state $w(w.ltime)$.

A *hybrid execution fragment* α of A is a finite or infinite alternating sequence $w_0 a_1 w_1 a_2 w_2 \cdots$, where $w_i \in \mathcal{W}$, $a_i \in \Sigma$, and if w_i is not the last trajectory of α then w_i is right-closed and the discrete transition $(w_i.lstate, a_{i+1}, w_{i+1}.fstate)$ is in $\mathcal{D}$, or equivalently $w_i.lstate \xrightarrow{a_{i+1}}_A w_{i+1}.fstate$. If $w_0.fstate \in \Theta$ then α is a *hybrid execution* of A. A hybrid execution α of A is *finite* if it is a finite sequence and the domain of its final trajectory is a right-closed interval and *admissible* if $\alpha.ltime = \infty$. If $R \subseteq$ *states*(A) and $s, s' \in R$, then s' is R*-reachable from* s provided that there is a hybrid execution fragment of A that starts in s, ends in s', and all of whose states are in the set R.

The *hybrid trace* of a hybrid execution fragment α of A, denoted by $h\text{-}trace(\alpha)$, is the sequence obtained by projecting α onto the external variables of A and subsequently removing all inert internal and environment actions. The set of *hybrid traces* of A, denoted by $h\text{-}traces(A)$, is the set of hybrid traces that arise from all the finite and admissible hybrid executions of A.

A *superdense time* in an execution fragment α of A is a pair (i, t), where $t \leq w_i.ltime$. We totally order superdense times in the execution fragment α lexicographically. An *occurrence of* a state s *in* an execution fragment α of A is a triple (i, t, s) such that (i, t) is a superdense time in α and $s = w_i(t)$. State occurrences in α are ordered according to their superdense times. If S is a set of states of A and α is an execution fragment of A, then $past(S, \alpha)$ is the set of state occurrences (i, t, s) in α such that either $s \in S$ or there is a previous state occurrence (i', t', s') in α with $s' \in S$.

Two HIOA A_1 and A_2 are *compatible* if $X_i \cap V_j = Y_i \cap Y_j = \Sigma_i^{int} \cap \Sigma_j = \Sigma_i^{out} \cap \Sigma_j^{out} = \emptyset$, for $i, j \in \{1, 2\}, i \neq j$. If A_1 and A_2 are compatible then their *composition* $A_1 \times A_2$ is defined to be the tuple $A = (U, X, Y, \Sigma^{in}, \Sigma^{int}, \Sigma^{out}, \Theta, \mathcal{D}, \mathcal{W})$ given by $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$, $X = X_1 \cup X_2$, $Y = Y_1 \cup Y_2$, $\Sigma^{in} = (\Sigma_1^{in} \cup \Sigma_2^{in}) - (\Sigma_1^{out} \cup \Sigma_2^{out})$, $\Sigma^{int} = \Sigma_1^{int} \cup \Sigma_2^{int}$, $\Sigma^{out} = \Sigma_1^{out} \cup \Sigma_2^{out}$, $\Theta = \{s \in \boldsymbol{V} \mid s \lceil V_1 \in \Theta_1 \ \wedge \ s \lceil V_2 \in \Theta_2\}$ and sets of discrete transitions $\mathcal{D}$ and trajectories $\mathcal{W}$ each of whose elements projects to discrete transitions and trajectories, respectively, of A_1 and A_2.

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

3 Protected Plant Systems

A *protected plant system* is modeled abstractly as a *physical plant* interacting with a *protection system*. The protection system is modeled as the composition of a set of *protectors* each of which is supposed to enforce a particular safety requirement of the physical plant. Our model is abstract in the sense that it does not specify any of the details or safety requirements of the physical plant.

The physical plant and each of the protectors are modeled as HIOA. The *physical plant* PP is an automaton that is assumed to be interacting with the protectors through the set J of communication channels, which are referred to as *ports*. The input action set Σ_{PP}^{in}, the output action set Σ_{PP}^{out}, and the input variable set U_{PP} are partitioned into subsets $\Sigma_{PP_j}^{in}$, $\Sigma_{PP_j}^{out}$, and U_{PP_j}, respectively, one for each port j. We use the letter p to denote a state of PP and P to denote a set of states of PP. A *protector* A for the physical plant PP and the port set $K \subseteq J$ is an automaton that is compatible with PP and whose output actions are exactly the input actions of PP on ports in K, whose output variables are exactly the input variables of PP on ports in K, and all of whose input actions and input variables are outputs of PP. It can easily be shown that the composition of two distinct protectors is itself a protector.

Letting S, R, and G be particular sets of states of PP, a protector automaton A for PP and ports K *guarantees* G *in* PP *from* S *given* R provided that every finite execution of the composition $PP \times A$ starting in a state in S that only involves states in R ends in a state in G regardless of the inputs that arrive at PP on ports other that those in K. Two protectors are *dependent*, if the correct operation of one relies on the correct operation of the other, and *independent*, otherwise. The following theorems express the *substitutivity* condition — the condition under which the implementation of a protector is *correct* with respect to its specification — and the *compositional* conditions — conditions under which the composition of independent or dependent protectors guarantees the conjunction of the safety properties guaranteed by the protectors being composed.

Theorem 1 (Substitutivity). *Let A_1 and A_2 be two protector automata for the same port set K of a physical plant automaton PP, and suppose that $A_1 \leq A_2$. If A_2 guarantees G in PP from S given R, then A_1 guarantees G in PP from S given R.*

Theorem 2 (Independent Protector Composition). *Suppose that $A_1, A_2, \ldots, A_k$ are protector automata for a physical plant automaton PP, with respective port sets $K_1, K_2, \ldots, K_k$, where $K_i \cap K_{i'} = \emptyset$, for all $i, i' \in \{1, \ldots, k\}, i \neq i'$. Suppose that each of the protectors A_i, for all $i \in \{1, \ldots, k\}$, guarantees G_i from S_i given R_i. If the protectors $A_1, A_2, \ldots, A_k$ are compatible, then their composition $\prod_{i \in \{1,\ldots,k\}} A_i$ is a protector for PP that guarantees $\bigcap_{i \in \{1,\ldots,k\}} G_i$ from $\bigcap_{i \in \{1,\ldots,k\}} S_i$ given $\bigcap_{i \in \{1,\ldots,k\}} R_i$.*

Theorem 3 (Dependent Protector Composition). *Suppose that $A_1, A_2, \ldots, A_k$ are protector automata for a physical plant automaton PP, with respective port sets $K_1, K_2, \ldots, K_k$, where $K_i \cap K_{i'} = \emptyset$, for all $i, i' \in \{1, \ldots, k\}, i \neq i'$. Suppose that each of the protector automata A_i, for all $i \in \{1, \ldots, k\}$, guarantees G_i from S_i given $R_i \bigcap \left(\bigcap_{i' \in \{1,\ldots,k\}, i' \neq i} G_{i'} \right)$.*

Assume that α is any finite execution of the system $PP \times \prod_{i \in \{1,\ldots,k\}} A_i$ starting from a state in $\bigcap_{i \in \{1,\ldots,k\}} S_i$ and all of whose states are in the set $\bigcap_{i \in \{1,\ldots,k\}} R_i$. Then, one of the following holds:

1. *Every state in α is in $\bigcap_{i \in \{1,\ldots,k\}} G_i$.*
2. *The finite execution α can be written as $\alpha_1 \frown \alpha_2$, where*
   (a) *all state occurrences in α_1, except possibly the last, are in the set of states $\bigcap_{i \in \{1,\ldots,k\}} G_i$,*
   (b) *if the last state occurrence in α_1 is in $\overline{G_i}$, for some $i \in \{1, \ldots, k\}$, then there exists $i' \in \{1, \ldots, k\}, i' \neq i$, such that the last state occurrence in α_1 is in $\overline{G_{i'}}$, and*
   (c) *all state occurrences in α_2, except possibly the first, are in the set of states $\bigcap_{i \in I} past(\overline{G_i}, \alpha)$, for some $I \subseteq \{1, \ldots, k\}$, where $|I| \geq 2$.*

In loose terms, Theorem 3 states that the composition of dependent protectors guarantees the conjunction of the safety properties guaranteed by the protectors being composed provided a single action or trajectory of the composed system can cause the violation of *at most* one of the safety properties guaranteed by the protectors being composed.

4 An Abstract Protector

The abstract protector automaton is parameterized in terms of the automaton PP, the subsets R, G, and S of the states of PP, the port index j, and the positive real-valued sampling period d. The PP automaton represents the physical plant being modeled. The set R, also referred to as the set of *reliance*, is the set of states to which we restrict the states of the PP automaton while considering a particular protector. This set is usually comprised of states satisfying a particular property of the physical plant that is required by the protector under consideration. The set G, also referred to as the set of *guarantee*, is the set of states to which the protector is designed to constrain the PP automaton. The set S is a set of states from which the protector under consideration is said to guarantee G given R; that is, given that the states of the PP automaton are restricted to the set R, the protector guarantees that every finite execution starting from an initial state in S ends in a state in G. The port index j and the sampling period d denote the port and the sampling period with which the abstract protector interacts with the PP automaton. Thus, an instantiation of the abstract protector automaton $Abs(PP, S, R, G, j, d)$ is obtained by specifying the parameters PP, *etc.*

To begin, we define several functions and sets that are useful in the definition of the abstract protector $Abs(PP, S, R, G, j, d)$. Although, formal definitions of these functions and sets are presented in Table 1, their informal interpretations follow. First, we define a function, $future_{PP,R,j}$, that yields the set of states of PP that are R-reachable from the given subset of R within an amount of time in the given subset of $\mathbb{R}^{\geq 0}$, under the constraint that no input actions arrive on port j of the PP automaton. We define a function, $no\text{-}op_{PP,R,j}$, which yields, for a given state in R, the set of input actions on port j of the PP automaton that do not affect the state of the PP automaton, provided they are executed prior to either time-passage, or other input actions on port j. For any state p in R, the input actions in the set $no\text{-}op_{PP,R,j}(p)$ are referred to as no-op input actions on port j of PP for the state p. We define a set, $very\text{-}safe_{PP,R,G,j}$, which is comprised of the states of PP that satisfy R and from which all R-reachable states of PP with no input actions on port j are in G. The set $very\text{-}safe_{PP,R,G,j}$ may be interpreted as the set consisting of the states from which the PP automaton is bound to remain within the set G provided that it remains within the set R and the protector on port j does not retract or issue additional protective actions. We define a set, $safe_{PP,R,G,j}$, which is comprised of the states of PP that satisfy R and from which the protector on port j has a "winning protective strategy"; that is, for any state p in $safe_{PP,R,G,j}$ there exists an input action on port j of the PP automaton whose immediate execution — its execution prior to any

Table 1 Terminology for the abstract protector $Abs(PP, S, R, G, j, d)$.

$future_{PP,R,j} : \mathcal{P}(R) \times \mathcal{P}(\mathbb{R}^{\geq 0}) \to \mathcal{P}(R)$, defined by:

$p \in future_{PP,R,j}(P, T)$, where $P \subseteq R$ and $T \subseteq \mathbb{R}^{\geq 0}$, if and only if p is R-reachable from some $p' \in P$ via a finite execution fragment α of PP with no input actions on port j and with $\alpha.ltime \in T$.

$no\text{-}op_{PP,R,j} : R \to \mathcal{P}(\Sigma^{in}_{PP_j})$, defined by:

$\pi \in no\text{-}op_{PP,R,j}(p)$ if and only if π is an input action on port j of PP such that for all $p', p'' \in R$ satisfying $p' \in future_{PP,R,j}(p, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' = p'$.

$very\text{-}safe_{PP,R,G,j} \subseteq R$, defined by:

$p \in very\text{-}safe_{PP,R,G,j}$ if and only if $future_{PP,R,j}(p, \mathbb{R}^{\geq 0}) \subseteq G$.

$safe_{PP,R,G,j} \subseteq R$, defined by:

$p \in safe_{PP,R,G,j}$ if and only if both of the following hold:

1. $future_{PP,R,j}(p, 0) \subseteq G$.
2. There exists an input action π on port j, such that for every $p', p'' \in R$ satisfying $p' \in future_{PP,R,j}(p, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' \in very\text{-}safe_{PP,R,G,j}$.

$safe_{PP,R,G,j} : \Sigma^{in}_{PP_j} \to \mathcal{P}(R)$, defined by:

$p \in safe_{PP,R,G,j}(\pi)$ if and only if both of the following hold:

1. $future_{PP,R,j}(p, 0) \subseteq G$.
2. For every $p', p'' \in R$ such that $p' \in future_{PP,R,j}(p, 0)$ and $p' \xrightarrow{\pi}_{PP} p''$, it is the case that $p'' \in very\text{-}safe_{PP,R,G,j}$.

$delay\text{-}safe_{PP,R,G,j} : \mathbb{R}^{\geq 0} \to \mathcal{P}(R)$, defined by:

$p \in delay\text{-}safe_{PP,R,G,j}(t)$ if and only if both of the following hold:

1. $future_{PP,R,j}(p, [0, t]) \subseteq G$.
2. $future_{PP,R,j}(p, t) \subseteq safe_{PP,R,G,j}$.

time-passage with the possibility that its execution follows an arbitrary number of discrete actions other than input actions on port j — guarantees that all subsequent R-reachable states of PP with no input actions on port j are in G; that is, the state following the execution of the particular input action of PP on port j is in the set $very\text{-}safe_{PP,R,G,j}$. We overload the notation $safe_{PP,R,G,j}$ by defining a function, $safe_{PP,R,G,j}$, which yields the states of PP that satisfy R and for which the immediate execution of the given input action on port j — its execution prior to any time-passage with the possibility that its execution follows an arbitrary number of discrete actions other than input actions on port j — guarantees that all subsequent R-reachable states of PP with no input actions on port j are in G; that is, the state following the execution of the given input action on port j is in the set $very\text{-}safe_{PP,R,G,j}$. Finally, we define a function, $delay\text{-}safe_{PP,R,G,j}$, which yields the set of states of PP that satisfy R and for which all states R-reachable within the given amount of time and with no input actions on port j are in G, and all states R-reachable in exactly the given amount of time and with no input actions on port j are in $safe_{PP,R,G,j}$.

We proceed by stating the various assumptions made about the physical plant PP and the abstract protector $Abs(PP, S, R, G, j, d)$. We assume that the

Fig. 1 *Sensor*(PP, S, R, G, j, d) automaton definition.

```
Actions:        Input:     $e$, the environment action
                Output:    $\texttt{snapshot}(y)_j$, for each valuation $y$ of $Y_{PP}$
Variables:      Input:     $u \in type(u)$, for all $u \in Y_{PP}$,
                              initially $u \in type(u)$, for each $u \in Y_{PP}$
                Internal:  $now_j \in \mathbb{R}^{\geq 0}$, initially 0
                           $next\text{-}snap_j \in \mathbb{R}^{\geq 0}$, initially 0
Discrete Transitions:
  $e$
      Eff: $Y_{PP} :\in \boldsymbol{Y_{PP}}$
  $\texttt{snapshot}(y)_j$
      Pre: $next\text{-}snap_j = now_j$
           $y$ is current valuation of $Y_{PP}$
      Eff: $Y_{PP} :\in \boldsymbol{Y_{PP}}$
           $next\text{-}snap_j := now_j + d$
Trajectories:
    for all $u \in Y_{PP}$
        $u$ assumes arbitrary values in $type(u)$ throughout $w$
    $next\text{-}snap_j$ is constant throughout $w$
    for all $t \in T_I$
        $w(t).now_j = w(0).now_j + t$
        $w(t).now_j \leq w(t).next\text{-}snap_j$
```

PP automaton has no input variables on port j, for all $j \in J$; that is, the protectors control the state of the physical plant only through input actions. A consequence of this assumption is that the environment action of the *PP* automaton is stuttering. Moreover, we assume that the *PP* automaton has no output actions on port j, for all $j \in J$. The physical plant is modeled as a passive system in the sense that the protectors observe the state of the plant only through output variables. We assume that there exist no-op input actions on port j for every state of the *PP* automaton in the set R. We assume that membership of a state of the *PP* automaton in the set $safe_{PP,R,G,j}$ is determinable from the output variables of the *PP* automaton, *i.e.*, the set $safe_{PP,R,G,j}$ is Y_{PP}-determinable. Moreover, we assume that for any state in the set $safe_{PP,R,G,j}$, an appropriate action to guarantee safety can be determined from the output variables of the *PP* automaton, *i.e.*, the variables in Y_{PP}. For any valuation y of the output variables Y_{PP} of the *PP* automaton, we use the notation $y \in safe_{PP,R,G,j}$ to denote the existence of a state $p \in safe_{PP,R,G,j}$ such that $p \lceil Y_{PP} = y$. We assume that the state information provided by the output variables of the *PP* automaton is sufficient to determine membership of any state of the *PP* automaton in the sets R and G, *i.e.*, the sets R and G are Y_{PP}-determinable. Moreover, we assume that the set of start states S is a subset of the set $safe_{PP,R,G,j}$.

The protector is defined as the composition of a *sensor automaton* (Figure 1) and a *discrete controller automaton* (Figure 2). Both the sensor and the discrete controller are described abstractly in terms of *PP*, S, R, G, j, and d and are respectively denoted *Sensor*(PP, S, R, G, j, d) and *DC*(PP, S, R, G, j, d). At intervals of d time units, the sensor automaton samples the output variables of the *PP* automaton. The discrete controller automaton is rather nondeterminis-

Fig. 2 *DC*(PP, S, R, G, j, d) automaton definition.

```
Actions:        Input:      $e$, the environment action (stuttering)
                            $\texttt{snapshot}(y)_j$, for each valuation $y$ of $Y_{PP}$
                Output:     $\pi$, for all $\pi \in \Sigma^{in}_{PP_j}$
Variables:      Internal:   $send_j \in \Sigma^{in}_{PP_j} \cup \{null\}$, initially $null$
Discrete Transitions:
  $e$
      Eff: None
  $\pi$
      Pre: $send_j = \pi$
      Eff: $send_j := null$
  $\texttt{snapshot}(y)_j$
      Eff: if $y \in safe_{PP,R,G,j}$ then
               $send_j :\in \{\phi \in \Sigma^{in}_{PP_j} \mid$
                   $\forall\, p, p', p'' \in R$ such that
                   $p \lceil Y_{PP} = y$, $p' \in future_{PP,R,j}(p, 0)$,
                   and $p' \xrightarrow{\phi}_{PP} p''$,
                   it is the case that
                   $p'' \in delay\text{-}safe_{PP,R,G,j}(d)\}$
           else
               $send_j :\in \Sigma^{in}_{PP_j}$
Trajectories:
  $w.send_j \equiv null$
```

tic. Based on the output state information of the *PP* automaton sampled by the sensor automaton, the discrete controller automaton issues protective actions so as to guarantee that (i) the *PP* automaton remains within the set G up to the next sampling point, and (ii) the state of the *PP* automaton at the next sampling point is in the set $safe_{PP,R,G,j}$. The nondeterminism in the description of the *DC*(PP, S, R, G, j, d) automaton allows the freedom to choose any response that satisfies the given conditions — however, in a discrete controller automaton implementation, a response that least restricts the future states of the physical plant automaton *PP* would be preferred because it would represent a weaker protective action.

Theorem 4. *Abs*(PP, S, R, G, j, d) *guarantees* G *in PP from* S *given* R.

The correctness proof of a particular protector implementation involves defining the particular protector's specification as the instantiation of the abstract protector for particular definitions of *PP*, *etc.* and showing that the particular protector implementation is correct with respect to the particular instantiation of the abstract protector. The first step simply involves specifying the parameters *PP*, *etc.* The second step is simplified by choosing the protector implementation to be the composition of the sensor automaton *Sensor*(PP, S, R, G, j, d) and a discrete automaton that is chosen so as to guarantee the effect clause of the $\texttt{snapshot}(y)_j$ action in *DC*(PP, S, R, G, j, d). Thus, the correctness proof of the implementation is reduced to a simulation from the implementation of the discrete controller automaton to its specification.

5 Modeling the PRT 2000™

In this section, we present a model for a simplified version of the PRT 2000™ whose track topology involves a single track. The model, VEHICLES, which is presented in Figure 3, is a HIOA that conforms to the restrictions and assumptions made about the *PP* automaton in Sections 3 and 4. It involves n vehicles of identical dimensions and acceleration/deceleration capabilities traveling on a single track. Its state variables include the position x_i, the velocity $\dot{x}_i$, and the acceleration $\ddot{x}_i$ of each vehicle i in the set of vehicles I and several other variables that record whether each vehicle has collided *into* each other vehicle ($\mathit{collided}(i, i')$, for $i' \in I, i' \neq i$), whether each vehicle is braking ($\mathit{brake}(i)$, for $i \in I$), and whether each protector j in the set of protectors J is requesting each particular vehicle to brake ($\mathit{brake\text{-}req}(i, j)$, for $i \in I$ and $j \in J$). Several properties of the physical plant are enforced by restricting the states of the VEHICLES automaton to the set *VALID* (Appendix A). In particular, we assume that the vehicles occupy non-overlapping sections of the track, the vehicles are only allowed to move forward on the track, the non-malfunctioning vehicle acceleration/deceleration capabilities to be within the interval $[\ddot{c}_{min}, \ddot{c}_{max}]$, and the non-malfunctioning braking deceleration to be given by $\ddot{c}_{brake}$, if the vehicle is moving forward, and 0, otherwise.

The formal definitions of the derived variables and sets of the VEHICLES automaton are shown in Appendix A. For brevity, we only give informal definitions of the key derived variables. Each of the variables E_i, for $i \in I$, denotes the *extent* of the vehicle i; that is, the section of the track *occupied* by the vehicle i. It is defined as the section of track ranging from the position of the rear of the vehicle i to the point on the track that is a distance of c_{len} downstream of the rear of the vehicle i — a distance that specifies the minimum allowable separation between vehicles, *i.e.*, $E_i = [x_i, x_i + c_{len}]$, for $i \in I$. Each of the variables O_i, for $i \in I$, denotes the section of the track that the vehicle i *owns*; that is, the range extending from the current position of the rear of the vehicle i to the point on the track that the vehicle can reach even if it is braked immediately. Each of the variables $C_i(t)$, for $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, denotes the section of the track that the vehicle i *claims* within t time units; that is, the range extending from the current position of the rear of the vehicle i to the point on the track that the vehicle i can reach if it is braked after t time units and assuming worst-case vehicle behavior up to the point in time when it is braked. Moreover, each of the variables $\mathit{collided}(*, i, *)$, for $i \in I$, denotes whether the vehicle i has ever been involved in a collision. Some auxiliary sets for the vehicles automaton that will be used in the following sections are defined in Appendix B.

The input actions of the VEHICLES automaton are the environment action e and the actions `brake`$(i)_j$ and `unbrake`$(i)_j$, for $i \in I$ and $j \in J$. Since the VEHICLES automaton has no input variables, the environment action e is stuttering. Each of the actions `brake`$(i)_j$ and `unbrake`$(i)_j$, for $i \in I$ and $j \in J$, correspond to actions performed by the protector j instructing the vehicle i to apply or release its “emergency” brake, respectively. Each `brick-wall`(i) action, for $i \in I$, models the instantaneous stopping of the vehicle i — as if it hit a brick wall.

Fig. 3 The VEHICLES automaton.

```
Actions:
  Input:
    $e$, the environment action (stuttering)
    brake$(i)_j$, for all $i \in I, j \in J$
    unbrake$(i)_j$, for all $i \in I, j \in J$

  Internal:
    colliding-pair$(i, i')$,
        for all $i, i' \in I, i' \neq i$
    collision-effects$(i)$, for all $i \in I$
    brick-wall$(i)$, for all $i \in I$

Variables
  Internal:
    $\ddot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\ddot{x}_i \in \mathbb{R}$
    $brake(i) \in$ Bool,
        for all $i \in I$, initially False
    $brake\text{-}req(i, j) \in$ Bool,
        for all $i \in I, j \in J$,
        initially False
  Output:
    $x_i \in \mathbb{R}$, for all $i \in I$, initially $x_i \in \mathbb{R}$
    $\dot{x}_i \in \mathbb{R}$, for all $i \in I$, initially $\dot{x}_i \in \mathbb{R}$
    $collided(i, i') \in$ Bool,
        for all $i, i' \in I, i' \neq i$,
        initially False
  subject to $VALID$

Discrete Transitions:
  $e$
    Eff: None

  brake$(i)_j$
    Eff: $brake\text{-}req(i, j) :=$ True
         if $\neg brake(i)$ then
           $brake(i) :=$ True
           if $\dot{x}_i = 0$ then $\ddot{x}_i := 0$
                       else $\ddot{x}_i := \ddot{c}_{brake}$
  unbrake$(i)_j$
    Eff: $brake\text{-}req(i, j) :=$ False
         if $brake(i)$
           $\wedge(\neg \vee_{k \in J}\ brake\text{-}req(i, k))$
         then
           $brake(i) :=$ False
           $\ddot{x}_i :\in [\ddot{c}_{min}, \ddot{c}_{max}]$

  colliding-pair$(i, i')$
    Pre: $\neg collided(i, i')$
         $\wedge(E_i \cap E_{i'} \neq \emptyset)$
         $\wedge(x_i < \min(E_i \cap E_{i'}))$
    Eff: $collided(i, i') :=$ True

  collision-effects$(i)$
    Pre: $collided(*, i, *)$
    Eff: $\dot{x}_i :\in \mathbb{R}^{\geq 0}$
         $\ddot{x}_i :\in \mathbb{R}$

  brick-wall$(i)$
    Pre: True
    Eff: $\dot{x}_i := 0$
         if $brake(i)$ then
           $\ddot{x}_i := 0$
         else
           $\ddot{x}_i :\in [0, \ddot{c}_{max}]$

Trajectories:
  for all $i, i' \in I, i \neq i'$, $collided(i, i')$ is constant throughout $w$
  for all $i \in I$ and $j \in J$, $brake(i)$ and $brake\text{-}req(i, j)$ are constant throughout $w$
  for all $i, i' \in I, i \neq i'$
      the function $w.\ddot{x}_i$ is integrable
      for all $t \in T_I$
          $w(t).\dot{x}_i = w(0).\dot{x}_i + \int_0^t w(s).\ddot{x}_i\ ds$
          $w(t).x_i = w(0).x_i + \int_0^t w(s).\dot{x}_i\ ds$
          if $\neg w.collided(i, i')$
            $\wedge(w(t).E_i \cap w(t).E_{i'} \neq \emptyset)$
            $\wedge(w(t).x_i < \min(w(t).E_i \cap w(t).E_{i'}))$
          then
            $t = w.ltime$
  subject to $VALID$
```

Thereafter however, the vehicle i is allowed to reinitiate forward motion. Each `colliding-pair`(i, i') action, for $i, i' \in I, i \neq i'$, records the fact that the vehicle i has collided into the vehicle i'. Since the trailing vehicle is the only vehicle that can prevent the collision through braking, a collision is recorded only by the trailing vehicle as if the trailing vehicle were the only vehicle liable for the particular collision. Each `collision-effects`(i) action, for $i \in I$, models the adverse effects of a collision involving the vehicle i and may be executed, even repeatedly, at any instant of time following the first collision involving the vehicle i. Thus, the malfunctioning apparatus of any vehicle i, for $i \in I$, is modeled by succeeding each of the discrete actions with a `collision-effects`(i) action for the malfunctioning vehicle.

The trajectories of the VEHICLES automaton model the continuous evolution of the state of the VEHICLES automaton. If during a trajectory a vehicle i collides into a vehicle i' for the first time, the trajectory is stopped so that the collision can be recorded.

6 Example Overspeed and Collision Avoidance Protectors

6.1 Example 1: Overspeed Protection System

In this section, we present a protector, called OS-PROT, that prevents the vehicles of the VEHICLES automaton from exceeding a prespecified global speed limit $\dot{c}_{max}$, provided that they do not collide among themselves. The protector OS-PROT is defined to be the composition of n separate copies of another protector called OS-PROT-SOLO$_i$, one copy for each vehicle $i \in I$. Each of the OS-PROT-SOLO$_i$ protectors, for $i \in I$, guarantees that the vehicle i, does not exceed the speed limit $\dot{c}_{max}$, provided that no collisions among any of the vehicles occur. The braking strategy of the OS-PROT-SOLO$_i$ protector is to instruct the vehicle i to brake if it is capable of exceeding the speed limit $\dot{c}_{max}$ within the time until the next sampling point.

Let PP_i be the VEHICLES automaton of Figure 3, the port j_i and the sampling period d_i be the port and sampling period with which the protector OS-PROT-SOLO$_i$ communicates with the VEHICLES automaton, the set R_i be the set of states in which none of the vehicles have ever collided, *i.e.*, $R_i = P_{\textit{not-collided}}$ (Appendix B), the set G_i be the set of states in which the vehicle i is at or below the speed limit, *i.e.*, $G_i = \textit{VALID} - P_{\textit{overspeed}(i)}$ (Appendix B), and the set S_i be the set $\textit{safe}_{PP_i, R_i, G_i, j_i}$. We define the OS-PROT-SOLO$_i$ automaton to be the composition of $\textit{Sensor}(PP_i, S_i, R_i, G_i, j_i, d_i)$ and the discrete controller automaton of Figure 4.

Lemma 1. *The protector* OS-PROT-SOLO$_i$ *guarantees* G_i *in* VEHICLES *starting from* S_i *given* R_i.

Corollary 1. *The protector* OS-PROT $= \prod_{i \in I}$ OS-PROT-SOLO$_i$ *for the* VEHICLES *automaton guarantees* $\bigcap_{i \in I} G_i$ *in the* VEHICLES *automaton starting from* $\bigcap_{i \in I} S_i$ *given* $P_{\textit{not-collided}}$.

Corollary 1 follows directly from Lemma 1 and Theorem 2.

Fig. 4 Discrete controller automaton for the protector OS-PROT-SOLO$_i$.

```
Actions:       Input:     e, the environment action (stuttering)
                          snapshot(y)_j, for each valuation y of Y_VEHICLES
               Output:    brake(i)_j
                          unbrake(i)_j
Variables:     Internal:  send_j ∈ {brake, unbrake, null}, initially null
Discrete Transitions:
    e                                       brake(i)_j
        Eff: None                               Pre: send_j = brake
                                                Eff: send_j := null
    snapshot(y)_j
        Eff: if (y.ẋ_i ≤ ċ_max − d c̈_max) then    unbrake(i)_j
                 send_j := unbrake              Pre: send_j = unbrake
             else                               Eff: send_j := null
                 send_j := brake

Trajectories:
    w.send_j ≡ null
```

6.2 Example 2: Collision Avoidance on a Single Track

In this section, we present a protector, called CL-PROT, that prevents the vehicles of the VEHICLES automaton from colliding among themselves, provided that they are all abiding by the speed limit $\dot{c}_{max}$. The protector CL-PROT is defined to be the composition of n separate copies of another protector called CL-PROT-SOLO$_i$, one copy for each vehicle $i \in I$. Each of the OS-PROT-SOLO$_i$ protectors, for $i \in I$, guarantees that the vehicle i does not collide into any of the vehicles it trails, provided that all the vehicles in the VEHICLES automaton are abiding by the speed limit and that all other vehicles $i' \in I, i' \neq i$, do not collide into any of the vehicles they respectively trail. The braking strategy of the CL-PROT-SOLO$_i$ protector is to instruct the vehicle i to brake if it has a d_i time unit claim overlap with any of the vehicles it trails. The rationale behind this braking strategy is that a collision between two vehicles in the VEHICLES automaton can only be prevented by instructing the trailing vehicle to brake.

Let PP_i be the VEHICLES automaton of Figure 3, the port j_i and the sampling period d_i be the port and sampling period with which the protector CL-PROT-SOLO$_i$ communicates with the VEHICLES automaton, and the set G_i be the set of states in which the vehicle i has not collided into any of the other vehicles, *i.e.*, $G = \mathit{VALID} - P_{\mathit{collided}(i)}$ (Appendix B). Moreover, let the set R_i be the set of states in which all of the vehicles are abiding by the speed limit and in which each of the other vehicles has never collided into any other vehicle, *i.e.*, $R_i = P_{\mathit{not\text{-}overspeed}} \bigcap \left(\bigcap_{i' \in I, i' \neq i} G_{i'} \right)$ (Appendix B), and the set S_i be the set $\mathit{safe}_{PP_i, R_i, G_i, j_i}$. We define the CL-PROT-SOLO$_i$ automaton to be the composition of $\mathit{Sensor}(PP_i, S_i, R_i, G_i, j_i, d_i)$ and the discrete controller automaton of Figure 5.

Fig. 5 Discrete controller automaton for the protector CL-PROT-SOLO$_i$.

Actions:	Input:	e, the environment action (stuttering)
		$\texttt{snapshot}(y)_j$, for each valuation y of Y_{VEHICLES}
	Output:	$\texttt{brake}(i)_j$
		$\texttt{unbrake}(i)_j$
Variables:	Internal:	$send_j \in \{\texttt{brake}, \texttt{unbrake}, null\}$, initially $null$

Discrete Transitions:

```
e
    Eff: None

snapshot(y)_j
    Eff: if $\exists\, i' \in I, i' \neq i$ such that
           $y \notin disjoint\text{-}claimed\text{-}tracks(i, i', d)$
           $\wedge(y.x_i < y.x_{i'})$
         then
           $send_j := \texttt{brake}$
         else
           $send_j := \texttt{unbrake}$

brake(i)_j
    Pre: $send_j = \texttt{brake}$
    Eff: $send_j := null$

unbrake(i)_j
    Pre: $send_j = \texttt{unbrake}$
    Eff: $send_j := null$
```

Trajectories:
$w.send_j \equiv null$

Lemma 2. *The protector* CL-PROT-SOLO$_i$ *guarantees* G_i *in* VEHICLES *starting from* S_i *given* R_i.

Lemma 3. *The protector* CL-PROT $= \prod_{i\, \in\, I}$ CL-PROT-SOLO$_i$ *for the* VEHICLES *automaton guarantees* $\bigcap_{i\, \in\, I} G_i$ *in the* VEHICLES *automaton starting from* $\bigcap_{i \in I} S_i$ *given* $P_{not\text{-}overspeed}$.

Lemma 3 is shown by combining Lemma 2 and Theorem 3 and realizing that the second condition of Theorem 3 does not hold.

6.3 Example 3: Collision Avoidance on Merging Tracks

In this section, we present a protector, called MERGE-PROT, that guarantees that none of the n vehicles that are traveling on a track involving a Y-shaped merge collide, provided that they are all abiding by the speed limit $\dot{c}_{max}$. The MERGE-PROT protector is defined as the composition of $n(n-1)/2$ separate copies of another protector called MERGE-PROT-PAIR$_{\{i,i'\}}$, one copy for each unordered pair of vehicles $\{i, i'\}$, where $i, i' \in I, i \neq i'$. Each of these MERGE-PROT-PAIR$_{\{i,i'\}}$ protectors, for $i, i' \in I, i \neq i'$, guarantees that the vehicles i and i' do not collide into each other, provided that all the vehicles are abiding by the speed limit and the vehicles of all other vehicle pairs do not collide between themselves.

We augment the VEHICLES automaton to involve a track topology consisting of a Y-shaped merge. This is done by replacing the position component of a vehicle's state with a location component — a component that specifies the track on which the vehicle is traveling and the vehicle's position with respect to the merge point — and update the definitions of the discrete steps and the trajectories of the VEHICLES automaton to handle the location variables. Furthermore, we replace the `brake` and `unbrake` input actions of the VEHICLES automaton with `protect` input actions which allow single protectors to instruct sets of vehicles to apply their "emergency" brakes. Finally, we augment the definitions of the discrete actions pertaining to vehicle collisions such that the blame for a particular collision is assigned to either only the trailing vehicle, if one vehicle collides into the other vehicle from behind, or both vehicles, if the vehicles collide sideways while merging. The resulting physical plant automaton is henceforth referred to as MERGE-VEHICLES.

Let $PP_{\{i,i'\}}$ be the MERGE-VEHICLES automaton. Let the port $j_{\{i,i'\}}$ and the sampling period $d_{\{i,i'\}}$ be the port and sampling period with which the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ communicates with the MERGE-VEHICLES automaton. Let $G_{\{i,i'\}}$ be the set of states in which the vehicles i and i' have not collided into each other, *i.e.*, $G_{\{i,i'\}} = \mathit{VALID} - P_{collided(i,i')} - P_{collided(i',i)}$ (Appendix B). Let $R_{\{i,i'\}}$ be the set of states of the MERGE-VEHICLES automaton in which all the vehicles are abiding by the speed limit and in which the vehicles of all other vehicle pairs have not collided into each other, *i.e.*, $R_{\{i,i'\}} = P_{not\text{-}overspeed} \bigcap \left(\bigcap_{i'',i''' \in I, i'' \neq i''', \{i'',i'''\} \neq \{i,i'\}} G_{\{i'',i'''\}} \right)$ (Appendix B). Finally, let $S_{\{i,i'\}}$ be the set $\mathit{safe}_{PP_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j_{\{i,i'\}}}$.

We define the protector MERGE-PROT-PAIR$_{\{i,i'\}}$ to be the composition of $\mathit{Sensor}(PP_{\{i,i'\}}, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j_{\{i,i'\}}, d_{\{i,i'\}})$ and a discrete controller automaton whose braking strategy is as follows. The discrete controller automaton is allowed to brake the vehicles i and i' only if the sections of the track they claim in time $d_{\{i,i'\}}$ overlap. Given that the vehicles i and i' are indeed involved in such a claim overlap, there are two possible scenarios depending on whether the locations of the vehicles i and i' are comparable, or not. If their locations are comparable, then the vehicle i is instructed to brake if it trails the vehicle i'; otherwise, the vehicle i' is instructed to brake. On the other hand, if the vehicle locations are not comparable, the vehicle i is instructed to brake either if *only* the vehicle i' owns the merge point, or if both or neither vehicles own the merge point and the vehicle i is traveling on the `left` branch of the merge; otherwise, the vehicle i' is instructed to brake. In the latter case, we choose to brake the vehicle traveling on the `left` branch for no particular reason. In fact, it is plausible to brake either or both of the vehicles involved in the claim overlap.

Lemma 4. *The protector* MERGE-PROT-PAIR$_{\{i,i'\}}$ *guarantees that the* MERGE-VEHICLES *automaton remains within* $G_{\{i,i'\}}$ *starting from* $S_{\{i,i'\}}$ *given* $R_{\{i,i'\}}$.

Lemma 5. *The protector* MERGE-PROT $= \prod_{i,i' \in I, i \neq i'}$ MERGE-PROT-PAIR$_{\{i,i'\}}$ *for the* MERGE-VEHICLES *automaton guarantees* $\bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$ *in* MERGE-VEHICLES *starting from* $\bigcap_{i,i' \in I, i \neq i'} S_{\{i,i'\}}$ *given* $P_{not\text{-}overspeed}$.

Lemma 5 is shown by combining Lemma 4 and Theorem 3 and realizing that the second condition of Theorem 3 does not hold.

6.4 Example 4: Collision Avoidance on a General Graph of Tracks

In this section, we present a protector, called GRAPH-PROT, that guarantees that none of the n vehicles traveling on a directed graph of tracks comprised of Y-shaped merges and diverges collide, provided that they are all abiding by the speed limit $\dot{c}_{max}$. As in Section 6.3, the GRAPH-PROT protector is defined as the composition of $n(n-1)/2$ separate copies of another protector called GRAPH-PROT-PAIR$_{\{i,i'\}}$, one copy for each unordered pair of vehicles $\{i, i'\}$, where $i, i' \in I, i \neq i'$. Each of the GRAPH-PROT-PAIR$_{\{i,i'\}}$ protectors, for $i, i' \in I, i \neq i'$, guarantees that the vehicles i and i' do not collide into each other, provided that all the vehicles are abiding by the speed limit and the vehicles of all other vehicle pairs do not collide between themselves.

We augment the MERGE-VEHICLES automaton to involve a general track topology consisting of a directed graph G of Y-shaped merges and diverges. All the edges of the graph G are assumed to be of sufficient length to rule out collisions among vehicles that are neither on identical, nor on contiguous edges and all cycles of the graph G are assumed to have at least three edges. Moreover, in order to brake the topological symmetry in merge situations, we associate with each edge of the track topology a unique and totally ordered priority index. The resulting physical plant automaton is henceforth referred to as GRAPH-VEHICLES.

Letting $PP_{\{i,i'\}}$, $S_{\{i,i'\}}$, $R_{\{i,i'\}}$, $G_{\{i,i'\}}$, $j_{\{i,i'\}}$, and $d_{\{i,i'\}}$ be as defined in Section 6.3, we define the GRAPH-PROT-PAIR$_{\{i,i'\}}$ automaton to be the composition of $\mathit{Sensor}(PP_{\{i,i'\}}, S_{\{i,i'\}}, R_{\{i,i'\}}, G_{\{i,i'\}}, j_{\{i,i'\}}, d_{\{i,i'\}})$ and a discrete controller automaton whose braking strategy is as follows. The discrete controller automaton is allowed to brake the vehicles i and i' only if the sections of the track they claim in $d_{\{i,i'\}}$ time units overlap. Given that the vehicles i and i' are indeed involved in such a claim overlap, there are two possible scenarios depending on whether the vehicles i and i' are traveling in succession, or on adjacent tracks. If the vehicles are traveling in succession, then the vehicle i is instructed to brake if it trails the vehicle i'; otherwise, the vehicle i' is instructed to brake. On the other hand, if the vehicles i and i' are traveling on adjacent edges, the vehicle i is instructed to brake either if *only* the vehicle i' owns the merge point, or if both or neither vehicles own the merge point and the vehicle i' is traveling on the edge of greater priority; otherwise, the vehicle i' is instructed to brake.

Lemma 6. *The protector* GRAPH-PROT-PAIR$_{\{i,i'\}}$ *guarantees that the* GRAPH-VEHICLES *automaton remains within* $G_{\{i,i'\}}$ *starting from* $S_{\{i,i'\}}$ *given* $R_{\{i,i'\}}$.

Lemma 7. *The protector* GRAPH-PROT $= \prod_{i,i' \in I, i \neq i'}$ GRAPH-PROT-PAIR$_{\{i,i'\}}$ *for the* GRAPH-VEHICLES *automaton guarantees* $\bigcap_{i,i' \in I, i \neq i'} G_{\{i,i'\}}$ *in* GRAPH-VEHICLES *starting from* $\bigcap_{i,i' \in I, i \neq i'} S_{\{i,i'\}}$ *given* $P_{\textit{not-overspeed}}$.

Lemma 7 is shown by combining Lemma 6 and Theorem 3 and realizing that the second condition of Theorem 3 does not hold.

6.5 Composing the Overspeed and Collision Protectors

In the previous sections, we presented example protectors whose correct operation required that the physical plant automaton at hand satisfied particular properties. For example, in the case of the VEHICLES automaton of Section 5, the overspeed protector OS-PROT of Section 6.1 assumes that none of the vehicles collide among themselves and the collision protector CL-PROT of Section 6.2 assumes that none of the vehicles exceed the speed limit. Using Theorem 3 it can be shown that the composition OS-PROT $\times$ CL-PROT is a protector for the VEHICLES automaton that guarantees that the vehicles in the VEHICLES automaton neither exceed the speed limit, nor collide among themselves. In fact, realizing that the OS-PROT protector extends, virtually unchanged, to the MERGE-VEHICLES and GRAPH-VEHICLES automata, such composition results extend to the MERGE-VEHICLES and GRAPH-VEHICLES automata by composing the OS-PROT protector with the MERGE-PROT and GRAPH-PROT protectors, respectively.

7 Conclusions

In this paper, we demonstrate how formal analysis techniques using the hybrid I/O automaton model can be applied to the specification and verification of hybrid systems whose structure adheres to the protection subsystem paradigm. We propose a parameterized abstract protector model which allows simple specification of an abstract protector for any hybrid system of this form. Such specification is obtained by defining the physical system, the start states, the sets of guarantee and reliance, and the port and sampling period with which the protector communicates with the physical plant. The proof of correctness of the abstract model leads to simple correctness proofs of the protector implementations for particular instantiations of the abstract model. Finally, the composition of independent, and even dependent protectors under mild conditions, guarantees the conjunction of the safety properties guaranteed by the individual protectors. The examples presented in this paper show that the proposed formal framework provides a precise and succinct protector specification, involves simple and straight forward proof methodology, and scales to complex hybrid systems through abstraction and modular decomposition.

Acknowledgments

We would hereby like to thank Dr. Steven L. Spielman of Raytheon Corporation and Norman M. Delisle formerly of Raytheon Corporation for helpful discussions regarding the PRT 2000™. We are grateful for having the opportunity to develop our formal modeling framework on the basis of a real application. We would also like to thank the submission's reviewers for their helpful suggestions and constructive comments.

References

1. Michael S. Branicky. *Studies in Hybrid Systems: Modeling, Analysis, and Control.* Doctor of Science Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, June 1995.

2. Ekaterina Dolginova and Nancy A. Lynch. Safety Verification for Automated Platoon Maneuvers: A Case Study. In Oded Maler, editor, *Proc. International Workshop on Hybrid and Real-Time Systems (HART'97)*, volume 1201 of *Lecture Notes in Computer Science*, pages 154–170. Springer-Verlag, 1997. The International Workshop on Hybrid and Real-Time Systems took place in Grenoble, France, in March 1997.
3. Rainer Gawlick, Roberto Segala, Jørgen Søgaard-Andersen, and Nancy A. Lynch. Liveness in Timed and Untimed Systems. Technical Report MIT/LCS/TR-587, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, December 1993.
4. Rainer Gawlick, Roberto Segala, Jørgen Søgaard-Andersen, and Nancy A. Lynch. Liveness in Timed and Untimed Systems. In Serge Abiteboul and Eli Shamir, editors, *Proc. 21st International Colloquium on Automata, Languages and Programming (ICALP'94)*, volume 820 of *Lecture Notes in Computer Science*, pages 166–177. Springer-Verlag, 1994. The 21st International Colloquium on Automata, Languages and Programming (ICALP'94) took place in Jerusalem, Israel, in July 1994. Full version appeared as Ref. 3.
5. Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993. This volume of LNCS was inspired by a workshop on the Theory of Hybrid Systems, held on Oct. 19–21, 1992 at the Technical University, Lyngby, Denmark, and by a prior Hybrid Systems Workshop, held on June 10–12, 1991 at the Mathematical Sciences Institute, Cornell University.
6. Carolos Livadas. Formal Verification of Safety-Critical Hybrid Systems. Master of Engineering Thesis, Dept. of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 1997.
7. John Lygeros. *Hierarchical, Hybrid Control of Large Scale Systems.* Doctor of Philosophy Thesis, Dept. of Electrical Engineering and Computer Sciences, University of California, Berkeley, May 1996.
8. John Lygeros, Datta N. Godbole, and Shankar Sastry. A Verified Hybrid Controller for Automated Vehicles. In *35th IEEE Conference on Decision and Control (CDC'96)*, pages 2289–2294, Kobe, Japan, December 1996.
9. John Lygeros and Nancy Lynch. On the Formal Verification of the TCAS Conflict Resolution Algorithm. In *36th IEEE Conference on Decision and Control (CDC'97)*, San Diego, CA, December 1997. To appear.
10. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O Automata. Technical Memo MIT/LCS/TM-544, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, December 1995.
11. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O Automata. In R. Alur, T. Henzinger, and E. Sontag, editors, *Proc. DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, Hybrid Systems III: Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 496–510. Springer-Verlag, 1996. The DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems took place in New Brunswick, New Jersey, in October 1995.
12. Nancy Lynch, Roberto Segala, Frits Vaandrager, and H. B. Weinberg. Hybrid I/O Automata. Preprint/Work in Progress. Preliminary versions appeared as Refs. 10 and 11, June 1997.
13. Nancy Lynch and Frits Vaandrager. Forward and Backward Simulations — Part II: Timing-Based Systems. Technical Memo MIT/LCS/TM-487.c, Labora-

tory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, April 1995.
14. Nancy Lynch and Frits Vaandrager. Forward and Backward Simulations — Part II: Timing-Based Systems. *Information and Computation*, 128(1):1–25, July 1996. Preliminary version appeared as Ref. 13.
15. George J. Pappas, Claire Tomlin, and Shankar Sastry. Conflict Resolution in Multi-Agent Hybrid Systems. In *35th IEEE Conference on Decision and Control (CDC'96)*, Kobe, Japan, December 1996.
16. Amir Pnueli and Joseph Sifakis, editors. *Special Issue on Hybrid Systems*, volume 138, part 1 of *Theoretical Computer Science*. Elsevier Science Publishers, February 1995.
17. H. B. Weinberg, Nancy Lynch, and Norman Delisle. Verification of Automated Vehicle Protection Systems. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III: Verification and Control*, volume 1066 of *Lecture Notes in Computer Science*, pages 101–113. Springer-Verlag, 1996.

A Derived Variables and Sets of the VEHICLES Automaton

$E_i \in \mathcal{P}(\mathbb{R})$, defined by $E_i = [x_i, x_i + c_{len}]$.

$\mathit{collided}(i,*) \in$ `Bool`, for $i \in I$, defined by $\mathit{collided}(i,*) = \bigvee_{i' \in I, i' \neq i} \mathit{collided}(i,i')$.

$\mathit{collided}(*,i) \in$ `Bool`, for $i \in I$, defined by $\mathit{collided}(*,i) = \bigvee_{i' \in I, i' \neq i} \mathit{collided}(i',i)$.

$\mathit{collided}(*,i,*) \in$ `Bool`, for $i \in I$, defined by $\mathit{collided}(*,i,*) = \mathit{collided}(*,i) \vee \mathit{collided}(i,*)$.

$\mathit{VALID} \subseteq \mathit{states}(\text{VEHICLES})$, defined by

$\mathit{VALID} = \{p \in \mathit{states}(\text{VEHICLES}) \mid$

1. $\nexists\, i, i' \in I, i \neq i'$ such that the set $p.E_i \cap p.E_{i'}$ is a positive length closed interval of $\mathbb{R}$.
2. $p.\dot{x}_i \geq 0$, for all $i \in I$.
3. If $\neg p.\mathit{collided}(*,i,*)$ then $p.\ddot{x}_i \in [\ddot{c}_{min}, \ddot{c}_{max}]$, for all $i \in I$.
4. If $\neg p.\mathit{collided}(*,i,*) \wedge p.\mathit{brake}(i)$ then if $p.\dot{x}_i = 0$ then $p.\ddot{x}_i = 0$ else $p.\ddot{x}_i = \ddot{c}_{brake}$, for all $i \in I$. $\}$

$\mathit{stop\text{-}dist}_i \in \mathbb{R}^{\geq 0}$, for all $i \in I$, defined by

$$\mathit{stop\text{-}dist}_i = -\frac{\dot{x}_i^2}{2\ddot{c}_{brake}}$$

$\mathit{max\text{-}range}_i(t) \in \mathbb{R}^{\geq 0}$, for all $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, defined by

$$\mathit{max\text{-}range}_i(t) = \begin{cases} \dot{x}_i \Delta t + \frac{1}{2}\ddot{c}_{max}\Delta t^2 + \dot{c}_{max}(t - \Delta t), \\ \quad \text{where } \Delta t = \min\left(t, \frac{\dot{c}_{max} - \dot{x}_i}{\ddot{c}_{max}}\right) & \text{if } \dot{x}_i \leq \dot{c}_{max}, \text{ and} \\ \dot{x}_i \Delta t + \frac{1}{2}\ddot{c}_{brake}\Delta t^2 + \dot{c}_{max}(t - \Delta t), \\ \quad \text{where } \Delta t = \min\left(t, \frac{\dot{c}_{max} - \dot{x}_i}{\ddot{c}_{brake}}\right) & \text{otherwise.} \end{cases}$$

$\mathit{max\text{-}vel}_i(t) \in \mathbb{R}^{\geq 0}$, for all $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, defined by

$$\mathit{max\text{-}vel}_i(t) = \begin{cases} \min(\dot{c}_{max}, \dot{x}_i + t\ddot{c}_{max}) & \text{if } \dot{x}_i \leq \dot{c}_{max}, \text{ and} \\ \max(\dot{c}_{max}, \dot{x}_i + t\ddot{c}_{brake}) & \text{otherwise.} \end{cases}$$

$O_i \subseteq \mathbb{R}$, for all $i \in I$, defined by

$$O_i = [x_i, x_i + \mathit{stop\text{-}dist}_i + c_{len}]$$

$C_i(t) \subseteq \mathbb{R}$, for all $i \in I$ and $t \in \mathbb{R}^{\geq 0}$, defined by

$$C_i(t) = \big[x_i, x_i + \mathit{max\text{-}range}_i(t) - \mathit{max\text{-}vel}_i(t)^2/(2\ddot{c}_{brake}) + c_{len}\big]$$

B Auxiliary Sets for the VEHICLES Automaton

$P_{overspeed(i)} \subseteq \mathit{VALID}$, for $i \in I$, defined by

$$P_{overspeed(i)} = \{p \in \mathit{VALID} \mid p.\dot{x}_i > \dot{c}_{max}\}$$

$P_{overspeed} \subseteq \mathit{VALID}$, defined by $P_{overspeed} = \bigcup_{i \in I} P_{overspeed(i)}$.

$P_{not\text{-}overspeed} \subseteq \mathit{VALID}$, defined by $P_{not\text{-}overspeed} = \mathit{VALID} - P_{overspeed}$.

$P_{collided(i,i')} \subseteq \mathit{VALID}$, for $i, i' \in I$, $i \neq i'$, defined by

$$P_{collided(i,i')} = \{p \in \mathit{VALID} \mid p.\mathit{collided}(i,i') = \texttt{True}\}$$

$P_{collided(i)} \subseteq \mathit{VALID}$, defined by $P_{collided(i)} = \bigcup_{i' \in I, i' \neq i} P_{collided(i,i')}$.

$P_{collided} \subseteq \mathit{VALID}$, defined by $P_{collided} = \bigcup_{i \in I} P_{collided(i)} = \bigcup_{i,i' \in I, i \neq i'} P_{collided(i,i')}$.

$P_{not\text{-}collided} \subseteq \mathit{VALID}$, defined by $P_{not\text{-}collided} = \mathit{VALID} - P_{collided}$.

$\mathit{disjoint\text{-}extents}(i,i') \subseteq \mathit{VALID}$, for $i, i' \in I, i \neq i'$, defined by

$$\mathit{disjoint\text{-}extents}(i,i') = \{p \in \mathit{VALID} \mid p.E_i \cap p.E_{i'} = \emptyset\}$$

$P_E \subseteq \mathit{VALID}$, defined by

$$P_E = \bigcap_{i,i' \in I, i \neq i'} \mathit{disjoint\text{-}extents}(i,i')$$

$\mathit{disjoint\text{-}owned\text{-}tracks}(i,i') \subseteq \mathit{VALID}$, for $i, i' \in I, i \neq i'$, defined by

$$\mathit{disjoint\text{-}owned\text{-}tracks}(i,i') = \{p \in \mathit{VALID} \mid p.O_i \cap p.O_{i'} = \emptyset\}$$

$P_O \subseteq \mathit{VALID}$, defined by

$$P_O = \bigcap_{i,i' \in I, i \neq i'} \mathit{disjoint\text{-}owned\text{-}tracks}(i,i')$$

$\mathit{disjoint\text{-}claimed\text{-}tracks}(i,i',t) \subseteq \mathit{VALID}$, for $i, i' \in I, i \neq i'$, and $t \in \mathbb{R}^{\geq 0}$, defined by

$$\mathit{disjoint\text{-}claimed\text{-}tracks}(i,i',t) = \{p \in \mathit{VALID} \mid p.C_i(t) \cap p.C_{i'}(t) = \emptyset\}$$

$P_{C(t)} \subseteq \mathit{VALID}$, for $t \in \mathbb{R}^{\geq 0}$, defined by

$$P_{C(t)} = \bigcap_{i,i' \in I, i \neq i'} \mathit{disjoint\text{-}claimed\text{-}tracks}(i,i',t)$$

$P_{B_{ij}} \subseteq \mathit{VALID}$, defined by

$$P_{B_{ij}} = \{p \in \mathit{VALID} \mid p.\mathit{brake\text{-}req}(i,j) = \texttt{True}\}$$